

Project Thetis
A Low-Cost, Low-Profile Inertial Data Logger

by
Braidan Duffy

Bachelor of Science
Ocean Engineering
Florida Institute of Technology
2021

A thesis
submitted to the College of Engineering and Science
at Florida Institute of Technology
in partial fulfillment of the requirements
for the degree of

Master of Science
in
Ocean Engineering

Melbourne, Florida
December, 2023

© Copyright 2023 Braidan Duffy
All Rights Reserved

The author grants permission to make single copies.

We the undersigned committee
hereby approve the attached thesis

Project Thetis
A Low-Cost, Low-Profile Inertial Data Logger by Braidan Duffy

Stephen L. Wood, Ph.D., P.E.
Professor
Ocean Engineering and Marine Sciences
Major Advisor

Hector M. Gutierrez, Ph.D., P.E.
Professor
Mechanical and Civil Engineering
Outside Committee Member

Robert J. Weaver, Ph.D.
Associate Professor
Ocean Engineering and Marine Sciences

Marius C. Silaghi, Ph.D.
Professor
Electrical Engineering and Computer Science

Richard B. Aronson, Ph.D.
Professor and Department Head
Ocean Engineering and Marine Sciences

Abstract

Title:

Project Thetis

A Low-Cost, Low-Profile Inertial Data Logger

Author:

Braidan Duffy

Major Advisor:

Stephen L. Wood, Ph.D., P.E.

This thesis details the design, testing, calibration, and verification of a nine degree of freedom inertial measurement data logger for use with floating bodies. The instrument was conceived to address limitations of equipment used in classes within the Ocean Engineering department at Florida Institute of Technology. By meeting with several stakeholders and end users, a series of stakeholder requirements, capabilities, and component-level requirements were developed that informed the design constraints. There were several hardware iterations of the board, culminating in Revision F5 which was extensively tested and proven. The design was inspected after testing concluded to determine which capabilities or requirements were met and detail additional efforts for future students. Overall, the design was successful and promises to be a compelling instrument for use in class and laboratory environments. With some additional work, it may become more equivalent to some of its commercial counterparts.

Table of Contents

Abstract	iii
List of Figures	xi
List of Tables	xiv
Abbreviations	xv
Acknowledgments	xviii
Dedication	xix
1 Introduction	1
1.1 Objectives	2
1.2 Hypothesis	2
2 How It Works	3
2.1 Orientation and Rotation	3
2.1.1 Rotation Matrices	4
2.1.2 Quaternions	7
2.2 Sensing	8
2.2.1 Magnetometer	9
2.2.2 Gyroscope	9
2.2.3 Accelerometer	10

2.2.4	MARG Arrays and Inertial Measurement Units	10
2.2.5	Global Positioning System	13
2.3	Sensor Fusion	13
2.3.1	Attitude and Heading Reference System	14
2.3.2	Inertial Calibration	15
2.3.3	Magnetic Calibration	19
2.3.4	Fast Complimentary Filter	20
2.3.5	Kalman Filter	22
2.3.6	Mahony Filter	23
2.3.7	Madgwick Filter	24
2.4	Project Management Techniques	25
2.4.1	Version Control	26
2.4.2	Systems Engineering	27
2.4.3	Agile Methodology	27
2.5	A Review of the State of the Art	28
2.5.1	Industrial Solutions	29
2.5.2	Hobbyist Solutions	33
3	How I Made It Work	35
3.1	Stakeholders	35
3.1.1	Hands-On Users	35
3.2	Design Rationale	36
3.2.1	Problem Description	36
3.2.2	Mission Statement	37
3.2.3	Stakeholder Requirements	38
3.2.4	Risk Identification	39
3.2.5	Quality Functional Deployment	50
3.3	Concept of Operations	54
3.4	Capabilities	54

3.5	Conceptual Design	54
3.5.1	System Requirements	57
3.5.2	Failure Mode, Effect, and Criticality Analysis	62
3.5.3	Functional Block Diagram	68
3.5.4	Functional Flow Diagram	68
3.6	Brief Revision History	70
3.6.1	Revision F1	70
3.6.2	Revision F2	71
3.6.3	Revision F3	72
3.6.4	Revision F4	73
3.7	Final Design	75
3.7.1	Schematic	77
3.7.2	Assembly Technique	78
3.8	Prototype Problems	82
4	Calibration	84
4.1	Methodologies	85
4.1.1	Magnetometer	86
4.1.2	Gyroscope	86
4.1.3	Accelerometer	88
4.1.4	Orientation	88
4.2	Results	89
4.2.1	Magnetometer	90
4.2.2	Gyroscope	91
4.2.3	Accelerometer	92
4.3	Discussion	94
5	Verification and Validation	98
5.1	Threshold Capabilities	99

5.2	Reach Capabilities	105
5.3	Stretch Capabilities	108
5.4	Stakeholder Requirements	110
6	Future Efforts and Potential Applications	113
6.1	Calibration	113
6.2	Hardware Revision F6	114
6.3	Software Improvements	116
6.3.1	Wireless	116
6.3.2	Settings/API	117
6.3.3	Data Storage	118
6.3.4	Sensors	119
6.4	More Verification and Validation	119
6.4.1	Wireless	120
6.4.2	Power	120
6.5	Calibration Machine	121
6.5.1	Requirements	121
6.5.2	Concept of Operations	122
6.6	Floating Body Testing	123
6.6.1	Surfboard	123
6.6.2	Model Barge	123
6.6.3	WEAVE	124
7	Conclusion	125
References	126
A	Voltage Divider	131
A.1	Example: Battery Monitor	132
B	Wheatstone Bridge	134

C Analog Measurement	137
C.1 Analog to Digital Conversion	138
D Digital Communication Methods	140
D.1 UART Explained	140
D.2 I2C Explained	141
D.3 SPI Explained	142
E Gyroscope Coriolis Effect Proof	144
F Rotation Matrices	146
F.1 Three Dimensional Rotation Matrix	148
G Kalman Filter	151
G.1 Introduction	151
G.2 Kalman Filter in One Dimension	152
G.2.1 The Kalman Gain	152
G.2.2 The Estimate Uncertainty Update Equation	153
G.2.3 The Estimate Uncertainty Extrapolation Equation	153
G.2.4 Putting it all Together	154
G.2.5 Process Noise	157
H Traceability Matrix	159
H.1 Requirement Lifecycle	162
I Wave Estimating Algorithm for Vessels Experiment (WEAVE)	164
I.1 Introduction	164
I.2 Background	165
I.2.1 Quaternion Representation	166
I.2.2 Quaternion Application to Rotations	167
I.3 Spectral Analysis Theory	168

I.4	Oscillatory Motion Tracking Theory	171
I.4.1	Filter Design	172
I.5	Methodology	174
I.5.1	For Spectral Analysis	174
I.5.2	For the Oscillatory Motion Tracking Analysis	175
J	Revision Schematics	178
J.1	Revision D1 Schematic	179
J.2	Revision D2 Schematic	180
J.3	Revision E1 Schematic	181
J.4	Revision F1 Schematic	182
J.5	Revision F2 Schematic	183
J.6	Revision F3 Schematic	184
J.7	Revision F4 Schematic	185
J.8	Revision F5 Schematic	186
J.9	Revision F6 Schematic	187
J.10	Revision G1 Schematic	188
J.11	Revision G2 Schematic	189
K	Raw Calibration Data Plots	190
K.1	Magnetometer Data	190
K.2	Gyroscope Data	191
K.3	Accelerometer Data	202
L	Software Installation and Setup	214
L.1	Installing the Development Environment	214
L.1.1	Step 1: Install VS Code and git VCS	215
L.1.2	Step Two: Install Extensions for VSCode	216
L.1.3	Step Three: Clone the Thetis-Firmware Repository to a Directory	217
L.1.4	Step Four: Check Proper PIO and Firmware Installation	218

L.2	Advanced Development Environment Setup	218
L.2.1	Configure Git Parameters	219
L.2.2	Fork Repositories and Reconfigure Submodules	219
M	User Manual	221

List of Figures

2.1	Body rotations	4
2.2	Illustration of the difference between the global and local reference frames . .	5
2.3	Illustration of rotation between coordinates frame about an axis	5
2.4	Gimbal lock demonstration	7
2.5	Magnetometer block diagram	9
2.6	Gyroscope block diagram	10
2.7	Accelerometer block diagram	11
2.8	ST124-M Outline	12
2.9	GPS diagram	14
2.10	AHRS block diagram	15
2.11	Correction of magnetic field readings by removing hard and soft iron distortion	21
2.12	Fast Complimentary Filter	22
2.13	Kalman Filter Process	23
2.14	V-Method Diagram	28
2.15	Sparton AHRS-8	29
2.16	Xsens Mti-100	30
2.17	VectorNav VN-200	30
2.18	SwiftNav Duro	31
2.19	Aceinna INS401	31
2.20	x-io Technologies x-IMU3	32
2.21	Kauai Labs NavX2 Micro	33

2.22	SparkFun OpenLog Artemis	34
2.23	Holybro Pixhawk 6X	34
3.1	Thetis RevF5 block diagram	68
3.2	Base Firmware flow diagram	69
3.3	Thetis Revision F1 PCB render	70
3.4	Thetis Revision F2 PCB render	72
3.5	Thetis Revision F3 PCB render	73
3.6	Thetis Revision F4 PCB render	74
3.7	Thetis Revision F5 PCB render	76
3.8	Comparison of through hole and SMT soldering	78
3.9	Solder deposition for stencil PCB assembly	79
3.10	Component placement and reflow for a Thetis Revision F5 PCB	80
3.11	PCB inspection after assembly	81
3.12	PCB firmware loading and testing after inspection	82
4.1	Calibration apparatuses	85
4.2	Calibration cube aligned with an external compass.	89
4.3	Magnetometer calibration process	90
4.4	Raw and windowed gyroscope data	92
4.5	Cumulative gyroscope error	92
4.6	Raw and windowed accelerometer data	93
4.7	Cumulative accelerometer error	93
5.1	ROV setup	101
5.2	Surfboard setup	101
5.3	GPS measurement deviations	107
6.1	Thetis Revision F6 PCB render	116
6.2	A three axis gimbal	121

A.1	A typical voltage divider circuit	131
A.2	A simple battery monitor circuit for 3.3V logic level microcontrollers.	133
B.1	A generalized wheatstone bridge	134
B.2	A linearized wheatstone bridge	135
C.1	Analog versus digital waveforms	137
C.2	Analog to digital converter waveform	139
C.3	A gated latch circuit used to store 1 bit of memory.	139
D.1	UART protocol diagram	141
D.2	I2C protocol diagram	142
D.3	SPI protocol diagram	143
F.1	2 dimensional rotation of a vector, A , to a new set of coordinates, A'	147
F.2	A vessel encounters a current	148
F.3	A plane experiences gravitational acceleration while at some attitude	149
G.1	Kalman Filter Diagram	154
H.1	Lifecycle of a requirement	163
I.1	Illustration of rotation between coordinates frame about an axis	166
I.2	Digital filter bode plots	173
L.1	Installation of extensions in VSCode	216
L.2	Opening git bash in Windows File Explorer	217
L.3	Output from a successful PlatformIO compile	218

List of Tables

2.1	Common definitions for IMUs of varying degrees of freedom.	11
2.2	Industry Solutions Comparison	32
3.1	A summary of stakeholders for the Thetis device	36
3.2	Stakeholder traceability matrix	38
3.12	House of Quality	53
3.13	Threshold capabilities	55
3.14	Reach capabilities	56
3.15	Stretch capabilities	56
4.1	Comparison errors between the magnetometers onboard Thetis and the x-IMU3	95
5.1	Verification and validation of threshold capabilities	100
5.2	Verification and validation of reach capabilities	106
5.3	Verification and validation of stretch capabilities	108
5.4	Verification and validation of stakeholder requirements	110
H.1	Description of requirement statuses	162

List of Symbols, Nomenclature or Abbreviations

Symbol	Meaning	Definition
ϕ	Roll	Typically, any rotation about the center or longitudinal (x) axis
θ	Pitch	Typically, any rotation about the side or transverse (y) axis
ψ	Yaw	Typically, any rotation about the vertical (z) axis
MEMS	Microelectromechanical Systems	Small electromechanical systems that perform a function and are micrometers in size
MARG	Magnetic, Angular Rate, and Gravity	A collection of tri-axial magnetometers, gyroscopes, and accelerometers in a sensor package
IMU	Inertial Measurement Unit	A collection of gyroscopes and/or accelerometers (optionally with magnetometers) in a sensor package
DOF	Degree(s) of Freedom	A measurement or movement axis in a body
GPS	Global Positioning System	A collection of satellites in Earth orbit that allows receivers to triangulate their position

ToF	Time of Flight	The time it takes a signal (typically electromagnetic) to be emitted, reflected, and return back to the emitter
AHRS	Attitude and Heading Reference System	An algorithm (or collection of algorithms) that report body attitude and heading from sensor data
VCS	Version Control Software	Software that tracks changes made to files and keeps a detailed history
SWaP	Size, Weight, and Power	Three of the most important metrics for considering the design of an embedded system
SCWaP	Size, Cost, Weight, and Power	Four of the most important metrics for considering the design of an embedded system
COTS	Commercial Off The Shelf	Components that can be routinely purchased from stores, retailers, suppliers, catalogs, etc.
API	Application Program Interface	A protocol that allows for standard interactions between different software packages
PCB	Printed Circuit Board	A (typically) fiberglass substrate with etched copper planes that condense electronics designs
IC	Integrated Circuit	A circuit that is integrated into a small board mountable package
DDS	Distributed Data Service	A method for sending messages between different software packages
ROS	Robot Operating System	A framework for handling DDS interactions between nodes in a software package
HAT	Hardware Attached on Top	A daughter board that can be attached onto a Raspberry Pi mini computer to expand its capability

UDP	User Datagram Protocol	A basic message protocol that sends a data packet (datagram) to a target specified in the header without considering a handshake or receipt acknowledgement
IP	Internet Protocol	A series of communication techniques that allow devices on a network to communicate with standard interfaces

Acknowledgements

I would like to acknowledge and thank Dr. Sebastian Madgwick, his company xio-Technologies, and his staff for their efforts in developing small embedded data loggers. The x-IMU3 became a major inspiration when it was announced earlier in 2023 and Seb was willing to chat with me about Thetis's design and its flaws. He has also been open to me using the x-IMU3 API on my board and the x-IMU3 user manual as a basis for Thetis's manual. His contributions and advice directly influenced version 2.0 of the firmware and the hardware revision F6. I am grateful for his correspondence and look forward to continuing our discussions on a variety of ideas.

Secondly, I would like to thank the students of the classes I taught in Fall 2022 and Spring 2023. They were instrumental in helping me formalize some of my notes for this thesis and helped test some early ideas. I am especially grateful for the students who deployed Thetis in their Surf Engineering Analysis class and for their patience when Thetis did not work as intended. I wish it worked better for them, but it gave invaluable insight into the potential application of Thetis.

Finally, I would like to thank my committee members, especially Drs. Wood and Weaver for keeping me on the path. Their advice and support kept me on the path, even during its most twisting and trying times. I also appreciate Dr. Gutierrez's consul in making sure Thetis was actually a usable product and motivating me to try and continually refine my design into the best it could be.

Dedication

This thesis is dedicated to you, the reader. May you find useful insight or information contained within. Or, at least have sweet dreams when you fall asleep by Chapter 4.

Chapter 1

Introduction

We all view the world through different lenses. Biologists look at plants and animals and seeks to understand their interactions; similarly, a scientist considers the universe around us and ponders what drives and constrains it; a psychologist examines human behavior and analyzes its origin and meaning. Humanity has always tried to study the way things work and figure out why they happen. That is the natural product of our curiosity and one of the many things that distinguishes us from other animals.

The fundamental problem that is considered throughout this thesis is sensing the inertial characteristics of a body and determining its orientation. The solutions to this problem are not new. It has been well-studied, well-documented, thoroughly proven out, and is even available to the masses at every moment of every day - even though they may not be aware they are using it. The purpose of the content herewithin is not to unveil something new or revolutionary. Rather, it is the culmination of years of viewing this problem through the engineer's lense and pondering the questions, "how does this work?" and "how can I make it work for me?" This is a guide into how an engineer can look at a problem and tinker with it and research it and develop something for it that enhances their understanding. While the final product may not be itself an innovation, the skills and knowledge required to get there can always lead to greater things.

1.1 Objectives

Based on the questions posed before, I set out to design an all-in-one, low-cost, open source inertial and positional data logger that could be used by students in their projects. The data logger will be able to replace instruments that are ill-suited for their applications in some classes, and to make some experiments in laboratory sections easier for students to analyze. Additionally, the device will be easy to reproduce, making it highly-attributable for students that will inevitably break the device during testing. Its open-source nature will also allow endeavoring students to expand the device's capabilities in both hardware and software as research projects or interests demand.

1.2 Hypothesis

The hypothesis considered for this thesis is that multiple tri-axial sensors could be integrated with a global positioning system radio and a WiFi-capable microcontroller to create a low-cost data logger that is on par with more expensive, state of the art solutions, while adding additional capabilities for students and being more attributable than existing solutions.

Chapter 2

How It Works

Measuring the movement of bodies in three dimensional space is not a new field of study. Especially in our current age of electronics, one can simply grab an accelerometer and microcontroller off of the Internet, have it delivered in less than 96 hours, and know the accelerations felt on a roller coaster or be able to orient a 3D rendering of a bunny¹. What makes today exciting is that improvements in manufacturing and computational density have drastically increased the capabilities of small embedded systems. Now, every cell phone has a sophisticated suite of sensors built into them that have the simple task of detecting whether the phone is in a landscape or portrait mode. Some applications, like Google Maps, have gone further and fused the sensor data, along with GPS information and computer vision, into a robust pedestrian navigation system².

2.1 Orientation and Rotation

A body can be rotated in 3D space along the x-, y-, and z-axis. **Roll** (ϕ) defines rotation about the x-axis; **pitch** (θ) defines rotation about the y-axis; and **yaw** (ψ) defines rotation

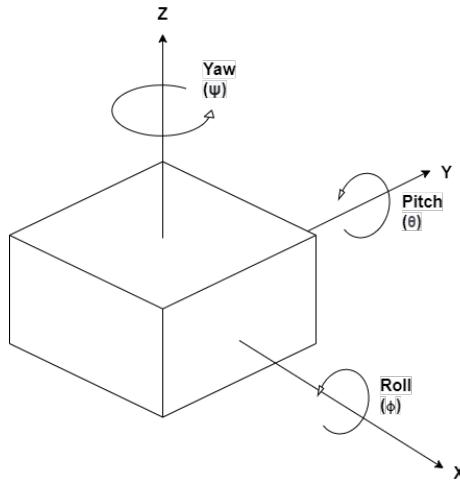
¹<https://learn.adafruit.com/adafruit-bno055-absolute-orientation-sensor/overview>

²<https://gizmodo.com/i-tried-google-maps-experimental-walking-directions-of-1833225629>

about the z-axis. These three rotations are known as Euler angles.

A body's orientation is always relative to another coordinate frame. Consider a person standing straight and still on the Earth's surface. Each of their appendages could be considered at some orientation relative to their body. At rest and in their local coordinate frame, the person's body may not be considered moving or rotating. But, when the scope is expanded to encompass the entire body and the Earth's frame, that person is now rotating about the Earth coordinate frame at almost 17,000 MPH.

Figure 2.1: Basic look at a 3D body and the axes of rotation in the body reference frame.



2.1.1 Rotation Matrices

A rotation matrix is a mathematical model for translating one body's inertial reference frame to another, e.g. local body to the global body. These are used for Eulerian transformations of vectors. The matrix is an $N \times N$ orthogonal matrix where N is the number of dimensions in the vector; for a three dimensional vector, the rotation matrix to go from one coordinate frame to another would be a 3×3 matrix.

In Figure 2.3, we can see a base coordinate frame, A , and a different coordinate frame, B that is rotated relative to the base frame about the axis, \mathbf{r} . To express the rotation from the base to the local frame, we can borrow notation from Craig [8]. The “from” or “base” frame is the preceding superscript while the “to” or “local” frame is the preceding subscript,

Aside: Notation and Nomenclature for Rotations

Bodies on their own typically use the “x, y, z” notation that is ubiquitous for the Cartesian coordinate system. However, when referring to planetary bodies like the Earth, this nomenclature typically changes to North, East, and Down (NED). This occurs because on the surface of a large spherical body, we can assume the local area is a plane tangential to the surface. To keep the broader scope in mind, we arbitrarily associate the x-axis with East, the y-axis with North, and the z-axis with Down towards the Earth’s core (perpendicular to the surface).

Figure 2.2: Illustration of the difference between the global and local reference frames

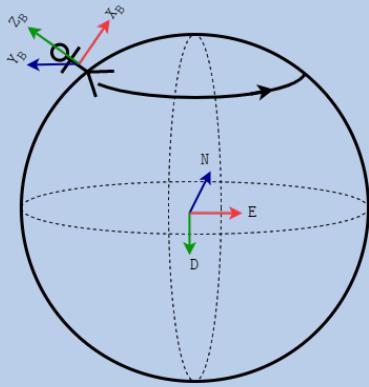
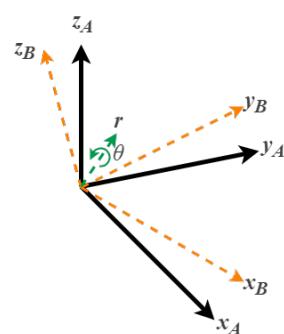


Figure 2.3: Illustration of the rotation from a base coordinate frame to a local coordinate frame about the axis, \mathbf{r}



as shown below:

$${}^A_B R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} = \begin{bmatrix} {}^A_B x & {}^A_B y & {}^A_B z \end{bmatrix}$$

When we want to rotate a vector between frames, we must choose an order in which to do so. For example, if we wanted to rotate from frame A to frame B using a yaw-pitch-roll rotation order, we can construct the following rotation matrix from Appendix F.1:

$$\begin{aligned} {}_B^A R &= R_z(\psi)R_y(\theta)R_x(\phi) = \begin{bmatrix} \cos \psi & -\sin \psi & 0 \\ \sin \psi & \cos \psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos \theta & 0 & \sin \theta \\ 0 & 1 & 0 \\ -\sin \theta & 0 & \cos \theta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \phi & -\sin \phi \\ 0 & \sin \phi & \cos \phi \end{bmatrix} \\ &= \begin{bmatrix} \cos \psi \cos \theta & \cos \psi \sin \theta \sin \phi - \sin \psi \cos \theta & \cos \psi \sin \theta \cos \phi + \sin \psi \sin \phi \\ \sin \psi \cos \theta & \sin \psi \sin \theta \sin \phi + \cos \psi \cos \phi & \sin \psi \sin \theta \cos \phi - \cos \psi \sin \phi \\ -\sin \theta & \cos \theta \sin \phi & \cos \theta \cos \phi \end{bmatrix} \end{aligned}$$

Then, we can rotate the vector from A to B via:

$${}_B^A \mathbf{v} = {}_B^A R \cdot {}^A \mathbf{v}$$

Gimbal Lock When two axes become parallel, Eulerian changes to either axis become irrelevant, ergo, the system loses a degree of freedom. This condition is called “gimbal lock”. When in gimbal lock, rotations yield discontinuities that need to be mitigated by changing the rotation order, or moving two or three axes at once. This can create strange pathways for the body and cause unexpected behavior. Monitoring for and breaking gimbal lock are also computationally expensive as the program must perform multiple matrix multiplications and then have logic to determine if a) it is in a gimbal lock condition and b) what rotation order would be required to break the condition.

The figure below shows a body’s rotation while in gimbal lock. The rotation order is roll-pitch-yaw. First, it is pitched 90 degrees upwards to align the roll and yaw axes, [0, 90, 0] (Figure 2.4b). If we then want it’s orientation to be [90, 0, 0] (Figure 2.4e), the body would be rolled left 90 degrees and pitched down 90 degrees. However, due to gimbal lock, the body

ended up in the orientation $[0, 0, 90]$ (Figure 2.4d)! To break the gimbal lock and get to the correct orientation, the rotation order would have to be changed to pitch-yaw-roll, or the roll and yaw axes driven simultaneously.

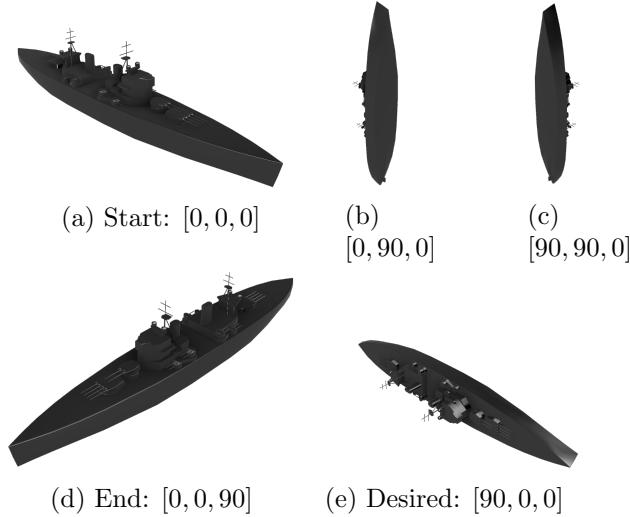


Figure 2.4: Demonstration of gimbal lock on a rotating 3D body. Despite only pitching and rolling the body, the end result of the transformation is an effective yaw. This is because the first pitch aligns the roll and yaw axes, effectively making them the same. In the given rotation order, any roll rotation would be equivalent to a yaw rotation when the object is pitched 90° . 3D model courtesy of “printable models” from Free3D.com.

2.1.2 Quaternions

In the 19th century, an Irish mathematician was contemplating the problem of Euler angles and analyzing three dimensional geometry. Sir William Hamilton considered the problem from within the complex or imaginary space and found that a three dimensional vector could not be expressed with a complex triplet. Instead, Hamilton discovered that the problem could be solved with a complex quadruplet, he called a quaternion [21], which could be used as a more mathematically pure tool to understand three dimensional geometry. For more detailed notes, see Kuipers [26]

A quaternion exists in four dimensional Hamiltonian space and has three complex and one real component in its vector. Like a rotation matrix, it describes the rotation from one reference frame to another about some axis, \mathbf{r} , as shown in Figure 2.3. But, the quaternion

is not susceptible to gimbal lock as when two of its axes align, it still retains three degrees of freedom. Additionally, it is not limited to the range of 0 to 360 (or -180 to 180) like Euler angles and therefore has no discontinuities when looping from one end of the range to the other. We can express the quaternion from frame A to frame B as:

$${}^A_B \mathbf{q} = \begin{bmatrix} q_1 & q_2 & q_3 & q_4 \end{bmatrix} = \begin{bmatrix} w & x & y & z \end{bmatrix} = w + x\hat{i} + y\hat{j} + z\hat{k} \quad (2.1)$$

Since the quaternion is just a vector, this drastically simplifies any rotation calculations and improves computational performance. In order to rotate a vector using a quaternion, we need to set the quaternion to be a unit quaternion by:

$$\mathbf{q} = \frac{\mathbf{q}}{\|\mathbf{q}\|} \quad (2.2)$$

Then, we can calculate the inverse quaternion, \mathbf{q}^{-1} (or conjugate $\mathbf{q}^* = \mathbf{q}^{-1}$, if $\|\mathbf{q}\| = 1$), using:

$$\mathbf{q}^{-1} = \frac{q_1 - q_2\hat{i} - q_3\hat{j} - q_4\hat{k}}{\|\mathbf{q}\|^2} \quad (2.3)$$

Then, we can rotate a three dimensional vector, ${}^A \mathbf{v}$, using the equation:

$${}^A \mathbf{v} = {}^A_B \mathbf{q} {}^A \mathbf{v} {}^A_B \mathbf{q}^{-1} \quad (2.4)$$

In an inertial measurement unit or attitude and heading reference system, we will need to rotate freely between readings taken in the body's local frame to that in the global frame, e.g. when calculating linear acceleration. Using the quaternion method above simplifies the calculation and can allow us to more easily determine body attitude from measurements.

2.2 Sensing

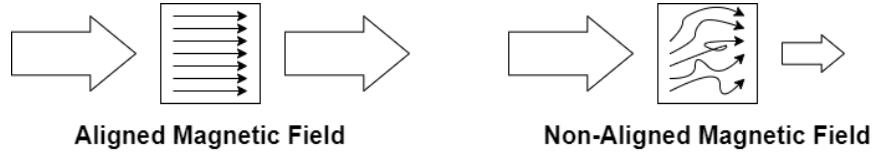
Now that we have a basic understanding of some of the mathematical relationships present in inertial tracking, we need to start sensing our environment. So, how can we compute the

position, velocity, acceleration, and attitude of a body in space? In order to answer this question, we must first examine the different sensors that are available.

2.2.1 Magnetometer

Magnetometers use magnetoresistive elements that change their effective resistance in the presence of a magnetic field [7]. Atoms within a magnetoresistive element change their orientations with the local magnetic field. The new orientation can hinder or aid the path of free electrons moving through the element, thus changing the resistance. By measuring this value and correlating it to a measurement scale, the local strength and direction of a magnetic field can be determined.

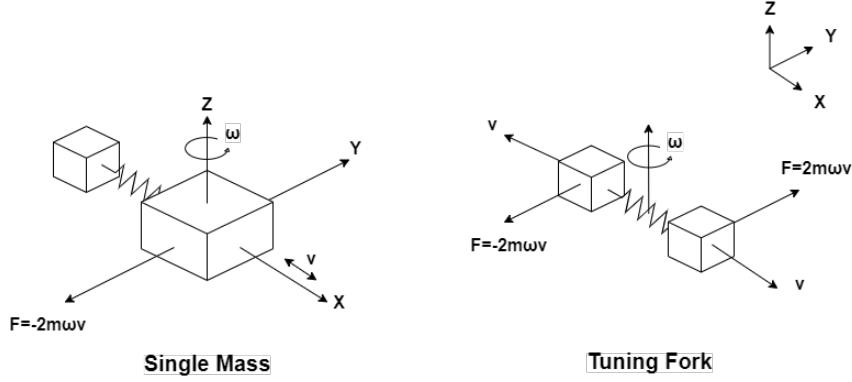
Figure 2.5: Basic block diagram of a single-axis magnetometer where electrons are flowing through the magnetoresistive material. The left diagram shows the condition when the magnetic field is aligned (minimal resistance); the right shows the non-aligned magnetic field condition which increases the resistance the electrons face passing through the material.



2.2.2 Gyroscope

A gyroscope is an inertial sensor that measures the angular velocity of a rotating body. MEMS-based gyroscopes measure this value by applying the Coriolis effect on a microscopic mass [7]. As shown in Figure 2.6, an oscillation is induced on the x-axis using a driving circuit. While oscillating, if an angular velocity (ω) is imparted on the z-axis, the suspended mass will experience a force in the y-axis that is proportional to ω (Appendix E). Newton's Second Law can be applied to directly correlate the force experienced to a displacement and therefore a change in resistance or capacitance.

Figure 2.6: Basic block diagram of a three-axis gyroscope where the masses are suspended from springs. The left diagram shows a single mass configuration; the right shows a tuning fork configuration which is twice as sensitive as the singular mass.



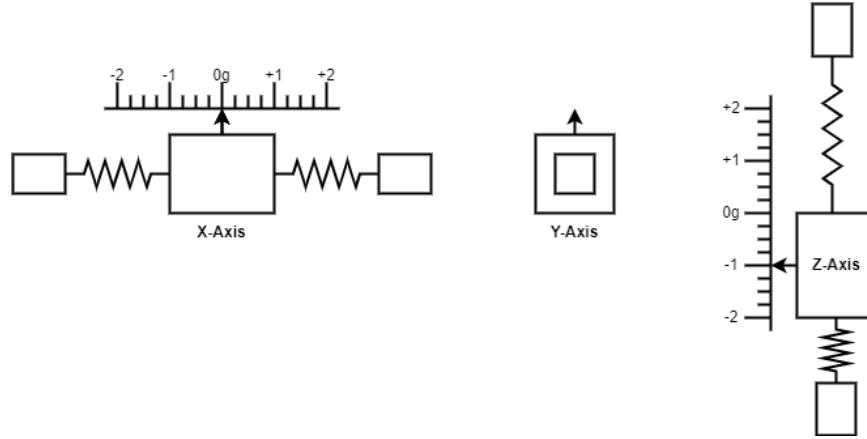
2.2.3 Accelerometer

Accelerometers measure a change in velocity over time (acceleration). An accelerometer is comprised of a mass suspended in an axis of motion by springs of a known K-constant [7]. From Newton's Second Law as applied to a spring-mass system, $\frac{-K}{m}x = a$, we can directly correlate the displacement of the mass to the magnitude of the acceleration along the measurement axis. Typically, this scale is electrically resistive or capacitative and creates an analog change in a driving (exciting) voltage which can be measured in a circuit using a Wheatstone Bridge (Appendix B) and microcontroller. A basic representation of a three-axis accelerometer is shown in Figure 2.7.

2.2.4 MARG Arrays and Inertial Measurement Units

Now, with the basics of each sensor in mind, we can combine them into a single package, called a Magnetic, Angular Rate, and Gravity (MARG) array. Each of the above diagrams represent a single sensing axis, or Degree of Freedom (DOF). In order for a MARG array to be useful in a 3D environment, it needs to have three sensing axes orthogonal to each other. Each DOF will measure the x-, y-, and z-axis, respectively with the positive sensing direction according to the right hand rule. By combining multiple tri-axial arrays, we can define different types of Inertial Measurement Units (IMUs), as shown in Table 2.1. Typically,

Figure 2.7: Basic block diagram of a three-axis accelerometer where the masses are suspended from springs.



the more sensing axes, the more accurate the array will be (depending on the performance of the sensor fusion algorithm). Note that a MARG array is an IMU with an integrated tri-axial magnetometer .

Table 2.1: Common definitions for IMUs of varying degrees of freedom.

DOFs	Accelerometer	Gyroscope	Magnetometer	Barometer
3-DOF	3-axis ³	3-axis ³	—	—
6-DOF	3-axis	3-axis	—	—
9-DOF	3-axis	3-axis	3-axis	—
10-DOF	3-axis	3-axis	3-axis	1-axis

According to VectorNav [50], a producer of industrial-grade IMUS, the cheapest, least precise, and least accurate IMUs are considered “consumer-grade”. Every day smartphones, cheap commercial breakout boards, and even shipping crates have these devices on-board. Consumer-grade IMUs can be bought for cents, dollars, or tens of dollars per unit in bulk and are ideal for mass production spaces where quantity is king over quality. A step up from these sensors are “industrial-grade” IMUs. These are tens to hundreds of dollars per unit but are an order of magnitude more accurate and precise than their consumer counter parts. This makes them desirable for the automotive and industrial sectors as they can assist in automation,

³can be either or, but not both.

Aside: MEMS Technology

During the Apollo program, the ST124-M inertial measurement unit was developed that fed the Saturn V rocket's inertial characteristics to the main flight computer [47]. The IMU consisted of a tri-axial gyroscope array, redundant tri-axial accelerometer arrays, pendulums, and other sensors. While it was a technological marvel at the time, it was the size of a basketball and weighed about 45-65 kilograms. Modern cellphones with the same capability are orders of magnitude smaller, lighter, and cheaper; so, what happened?

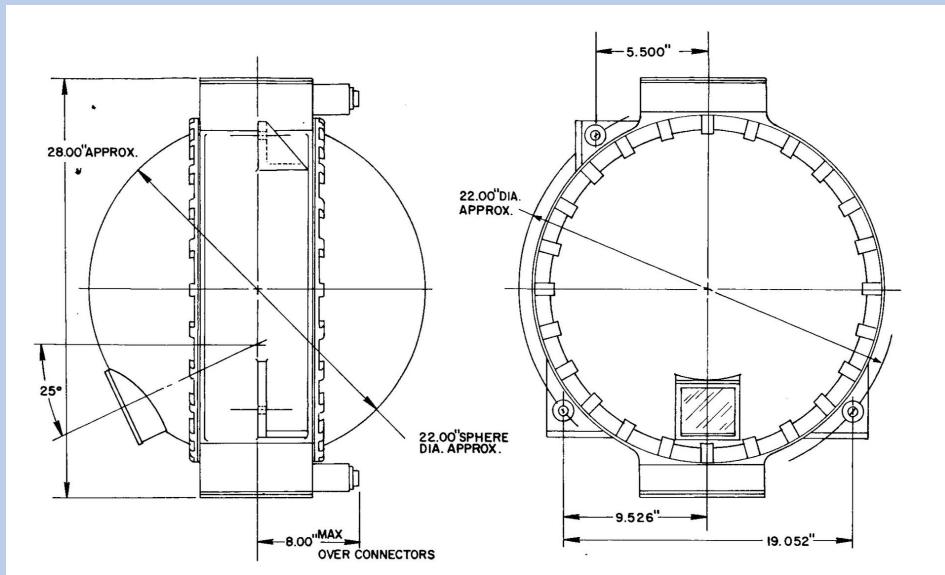


Figure 2.8: ST124-M inertial measurement unit courtesy of NASA [47]

Modern manufacturing methods have enabled a new technology called Microelectromechanical Systems, or MEMS. MEMS are microscopic devices that perform an electromechanical function such as sensing acceleration, rotation rate, or magnetic fields [35]. They are the most common technology in modern sensor development and revolutionized the technology space by shrinking components down to the micro- and nanometer scales. Due to the advent of MEMS technology, devices such as accelerometers, magnetometers, gyroscopes, barometers, hygrometers, etc. have been shrunk down from large mechanical masterpieces to mass-producible products that fit within a few square millimeters of epoxy. This dramatically cheapened these devices and has allowed more products to integrate smart sensors into their designs.

control, and monitoring of expensive unmanned systems. The “tactical-grade” sensors are even more robust and accurate than the previous tiers and have an appropriate military-industrial complex price tag to match. These devices will typically be used in applications where war fighters need precise guidance for munitions, or need to navigate their way through hazardous and/or GPS-denied terrain. Finally, the last major tier are the “navigation-grade” sensors. These sensors are extremely precise, extremely accurate, and cost more than some middle-class families will make in a year. Primarily, these will be used in survey missions or on underwater vehicles where absolute precision and knowledge of their location is necessary.

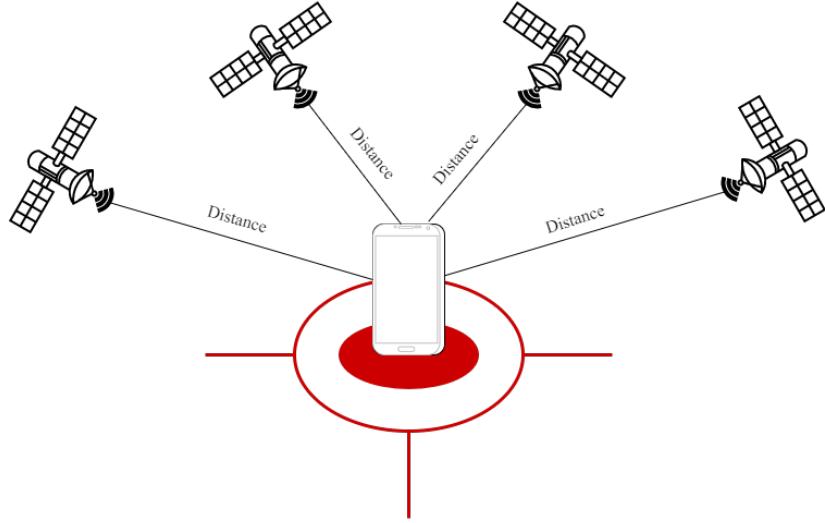
2.2.5 Global Positioning System

The Global Position System, or GPS, is a constellation of high-altitude satellites that service most of the globe [22]. First developed by the US military for large scale maneuvers on the battlefield, GPS is a ubiquitous technology that is available in almost every device from smartphones to cars. Each GPS satellite in orbit transmits the current time measured by their internal atomic clocks. A GPS receiver on the ground can synchronize its own internal clock to the GPS time and wait for a satellite’s transmission. When the satellite time is received, the device can determine the difference between its clock and the satellite’s report called the Time of Flight (ToF). Since the Time of Flight can be assumed to be the constant speed of light, the GPS receiver can determine its distance to a satellite in a known orbit. Repeat for at least three satellites, and a GPS receiver can triangulate its position to a reasonable circle of error of about 3-meters.

2.3 Sensor Fusion

While having each individual sensor will give some data, they offer an incomplete picture of a body’s orientation and movement through space. An accelerometer can detect acceleration and determine pitch and roll using basic vector math, but it cannot determine yaw or heading. Data from a gyroscope can be integrated over time to determine the sensor’s attitude, but this method will drift over time and accumulate errors, it also cannot detect movement.

Figure 2.9: Basic representation of how GPS communicates with a device to calculate its position on Earth. GPS satellites in orbit broadcast a time and using trigonometry and algebra, a device can calculate the distance to various satellites and triangulate its position.



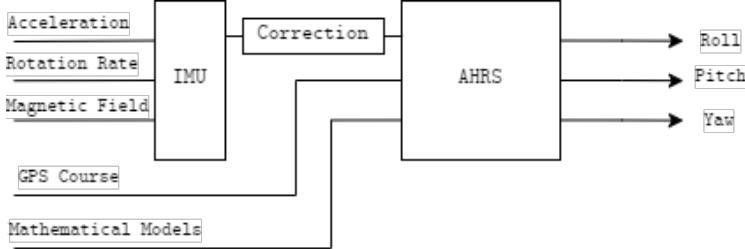
Magnetometers in a weak magnetic field, like Earth's, are not accurate enough to determine roll and pitch, but can easily provide heading. Finally, GPS readings will provide position and velocity vectors, but typically update slowly and have a large circle of error and cannot determine attitude. To make these sensors more effective for an inertial sensing application, we need to fuse the data feeds into a unified output that emphasizes the strength of each sensor, while mitigating their limitations.

2.3.1 Attitude and Heading Reference System

An Attitude and Heading Reference System (AHRS) is an IMU equipped with an accelerometer, gyroscope, and/or a magnetometer in all three axes. The data streams from the sensors can be fused together with external information like GPS data and mathematical models to estimate a body's inertial orientation in three dimensional space. The block diagram for this operation is shown in Figure 2.10. Many algorithms exist to do this sensor fusion such as the the Fast Complimentary Filter [52], Kalman filter [24], the Mahony filter [33], and the Madgwick filter [32].

Before measurements are fed into a sensor fusion algorithm, the data should be pre-

Figure 2.10: Basic block diagram of an AHRS. The acceleration, rotation rates, and magnetic field readings are fused together in the IMU the filtered using calibration data. The AHRS can then apply a bias from the GPS course and corrections from a mathematical model of the system.



filtered based on known calibration data. This will improve the sensor fusion performance and accuracy of the resultant values.

2.3.2 Inertial Calibration

From Madgwick [32], in order to calibrate an IMU, we first need to create a model of the data. For the accelerometer and gyroscope, we can create an inertial model defined as:

$$\mathbf{i}_c = M S (\mathbf{i}_u - \mathbf{b}) \quad (2.5)$$

where \mathbf{i}_c is the calibrated inertial measurements,

M is the misalignment matrix,

S is the sensitivity identity matrix,

\mathbf{i}_u is the uncalibrated inertial measurements, and

\mathbf{b} is the bias or offset vector

The three calibration values, M , S , and \mathbf{b} , represent a set of correction factors that make the measured values more accurate. The inertial measurement is generalized here to represent either accelerometer or gyroscope data. Each sensor will have its own set of the calibration values.

Bias Vector The bias, or offset, vector is the average of the inertial readings while the sensor is in a stable, known orientation. For example, when the gyroscope is at rest, we would expect the measurement output to be $[0, 0, 0]$ [deg/sec]. However, we may measure an average of $[0.89, -0.66, 0.31]$ [deg/sec] instead. By subtracting this bias vector from the measurements, we eliminate any unintentional offset from our readings. The bias can be calculated using the equation below:

$$\mathbf{b}_g = \frac{1}{N} \sum_{n=0}^N \mathbf{g}_n \quad (2.6)$$

where \mathbf{b}_g is the bias vector,

N is the number of collected samples to average, and

\mathbf{g}_n is the uncalibrated gyroscope measurement vector

For accelerometers, the bias is calculated differently. Each measurement axis must be exposed to $\pm 1g$ of acceleration by placing the instrument vertical on each of the measurement axes in both the positive and negative directions. The bias for each axis can then be determined by taking the average value as shown below:

$$b_a = \frac{1}{2} \left[\frac{1}{N} \sum_{n=0}^N a_{n,+g} + \frac{1}{M} \sum_{m=0}^M a_{m,-g} \right] \quad (2.7)$$

where b_a is the bias for a measurement axis,

N is the number of samples taken in the $+1g$ orientation,

a_{+g} is the average axis measurement when exposed to $+1g$,

M is the number of samples taken in the $-1g$ orientation, and

a_{-g} is the average axis measurement when exposed to $-1g$

Sensitivity Matrix The sensitivity matrix is a diagonal matrix that accounts for minor errors with variations in manufacturing process and sensor material. These errors can make

each uniaxial sensor in a triaxial array sense the environment slightly differently. While bias covers this area when at rest or in specific orientations, the sensitivity error will change depending on the motion and orientation. To account for this, we need to expose each sensor to a known reference stimulus and calculate the sensitivity based on the average magnitude of the measurement vector during that time. For a gyroscope, this value can be an arbitrary, but constant, rotation rate, ω . For an accelerometer, this value will be $\pm 1g$. The equations for each gyroscope and accelerometer axis are provided below:

$$s_{g,\omega} = \frac{\|g_{+\omega}\| + \|g_{-\omega}\|}{2\omega} \quad (2.8)$$

where $s_{g,\omega}$ is the sensitivity value for each gyroscope axis when exposed to ω ,

$g_{+\omega}$ is the average gyroscope axis reading when exposed to $+\omega$,

$g_{-\omega}$ is the average gyroscope axis reading when exposed to $-\omega$, and

ω is the reference rotation rate

$$s_{a,g} = \frac{\|a_{+g}\| + \|a_{-g}\|}{2} \quad (2.9)$$

where $s_{a,g}$ is the sensitivity value for each axis when exposed to $1g$,

a_{+g} is the average axis measurement when exposed to $+1g$, and

a_{-g} is the average axis measurement when exposed to $-1g$

After calculating these values for each axis, we can form the sensitivity matrix for the gyroscope and accelerometer like so:

$$S_g = \begin{bmatrix} s_{g,x} & 0 & 0 \\ 0 & s_{g,y} & 0 \\ 0 & 0 & s_{g,z} \end{bmatrix} \quad (2.10)$$

$$S_a = \begin{bmatrix} s_{a,x} & 0 & 0 \\ 0 & s_{a,y} & 0 \\ 0 & 0 & s_{a,z} \end{bmatrix} \quad (2.11)$$

While this calibration value accounts for some of the intrinsic sensor error, it does not account for axes misalignment or non-orthogonality within the sensor. To accomplish this, we must incorporate the misalignment matrix.

Misalignment Matrix The misalignment matrix is the final, but most complex, calibration value that we can calculate for the inertial sensors. It reduces error from multiple sources such as non-orthogonality between the measurement axes, the misalignment from the measurement axes to the actual sensor packaging, and misalignment from the package onto the application board. In order to calculate the misalignment matrix, we must consider it as the solution to a non-linear problem. First, we must define an objective function for the solution space. Since the goal of the misalignment matrix is to reduce error, then we can define the objective function as the Root Mean Square Error (RMSE) of the measurement vector, the misalignment vector, and the reference vector:

$$RMSE = \sqrt{\frac{1}{N} \sum_{n=0}^N \|\mathbf{i}_{n,u} \cdot M^* - \mathbf{i}_{n,\text{ref}}\|^2} \quad (2.12)$$

where N is the number of samples in the dataset,

$\mathbf{i}_{n,u}$ is the uncalibrated measurement vector with the bias removed,

M^* is the guessed 3×3 matrix that corrects $\mathbf{i}_{n,u}$ to be near \mathbf{i}_{ref} , and

\mathbf{i}_{ref} is the reference stimulus vector for the measurement

Then, given a dataset of measurement vectors and their expected reference vectors, we can use a non-liner solver⁴ to determine the value of M^* . Assuming the sensitivity was not

⁴Like the `scipy.optimize.minimize` toolbox

removed from the measurement signal, the M^* matrix happens to include it as its diagonal. By extracting the diagonal and normalizing the original matrix, we are left with the sensitivity and misalignment matrix calibration values.

$$S = \text{diag}(M^*) \quad (2.13)$$

$$M = M^* \cdot S^{-1}$$

We can now apply the misalignment matrix, sensitivity matrix, and bias vector to the sensor readings according to the model in Equation 2.5. This should yield outputs that are close to the real values, which is examined in Chapter 4.

2.3.3 Magnetic Calibration

The magnetometer in a MARG array needs a different sensor model than the inertial sensors because the ambient magnetic field introduces different errors. There are two types: soft iron, W , and hard iron, \mathbf{v} . A hard iron distortion is introduced by magnetic materials placed near the sensor. This can occur when the sensor is placed in a casing with permanent magnets or near a speaker, as shown in Tuupola [48]. Soft iron distortion is typically less severe than hard iron distortion and is caused by materials near the sensor that distort the local magnetic field. The soft iron distortion can also account for misalignment and sensitivity errors. The sensor model for a calibrated magnetometer is given below:

$$\mathbf{m}_c = W^{-1}(\mathbf{m}_u - \mathbf{v}) \quad (2.14)$$

where \mathbf{m}_c is the calibrated magnetometer vector,

W^{-1} is the inverse soft iron matrix,

\mathbf{m}_u is the uncalibrated magnetometer vector,

\mathbf{v} is the hard iron bias vector

With a calibrated sensor in an ideal environment, we would expect a sphere centered on the origin with a radius equal to the magnitude of the local magnetic field. Due to the hard and soft iron distortions, this will rarely be the case. Hard iron distortion can be removed from the signal by determining the average of the minimum and maximum values of each axis, as shown below:

$$\mathbf{v} = \frac{1}{2} \begin{bmatrix} \max(m_x) + \min(m_x) \\ \max(m_y) + \min(m_y) \\ \max(m_z) + \min(m_z) \end{bmatrix} \quad (2.15)$$

Once the hard iron distortion is removed from the signal, we should see the readings more closely adhere to the correct spherical shape. Then, we can apply Li's ellipsoid fitting algorithm [27] to characterize the fit of the optimal ellipsoid for the data, expressed as the symmetrical matrix, A . Ozyagcilar [39] builds a relationship between the ellipsoid fit and the soft iron matrix, summarized by:

$$W^{-1} = A^{1/2} \quad (2.16)$$

With these calibration parameters calculated, they can be used in a sensor fusion algorithm to increase the accuracy of the magnetometer. However, these parameters must be recalibrated whenever the sensor enters a new magnetic environment since either the hard or soft iron distortion could be different than previously calculated.

2.3.4 Fast Complimentary Filter

A complimentary filter take advantage of the problems inherent with the accelerometer and gyroscope measurements such they *compliment* each other. By integrating gyroscopic measurements, we can quickly obtain the body's attitude but the errors in the gyroscope sensor cause this value to quickly drift away from the "true" value. Accelerometer measurements on the other hand can use trigonometry on the gravitational acceleration vector to determine pitch and roll (tilt). These measurements are affected by the body's motion and can drift over time

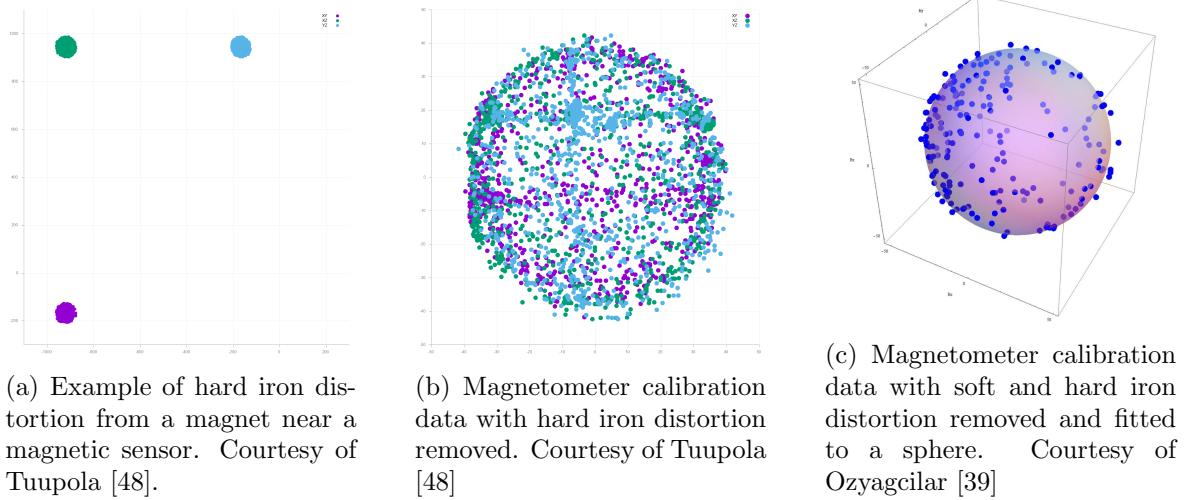


Figure 2.11: Correction of magnetic field readings by removing hard and soft iron distortion

as the body experiences other accelerations besides gravity. An accelerometer also cannot report the yaw rotation, so must be coupled with a tri-axial magnetometer which can also be influenced by external magnetic fields.

In the frequency space, we can consider the gyroscope attitude estimate to drift with a high frequency, and the accelerometer-magnetometer signal to drift with a low frequency. To compensate for these drifts, we can establish a complimentary or band-pass filter that attenuates the drift signal and outputs the best estimate of orientation. This filter typically uses a hyperparameter, γ , to control the attenuation of each of the drifts, i.e. the weight of each estimation like so:

$${}^G_B \mathbf{q}_{est} = (1 - \gamma) {}^G_B \mathbf{q}_{lf} + \gamma {}^G_B \mathbf{q}_{hf} \quad (2.17)$$

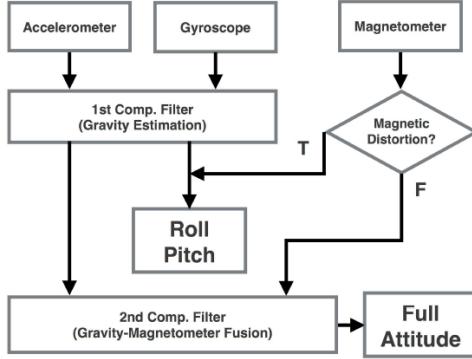
where ${}^G_B \mathbf{q}_{est}$ is the best estimate of the body quaternion relative to the global frame,

γ is the controlling hyperparameter,

${}^G_B \mathbf{q}_{lf}$ is the body quaternion estimate from the low frequency drift source, and

${}^G_B \mathbf{q}_{hf}$ is the body quaternion estimate from the high frequency drift source.

Figure 2.12: A simplified block diagram for the Fast Complimentary Filter. Courtesy of Wu et al. [52]

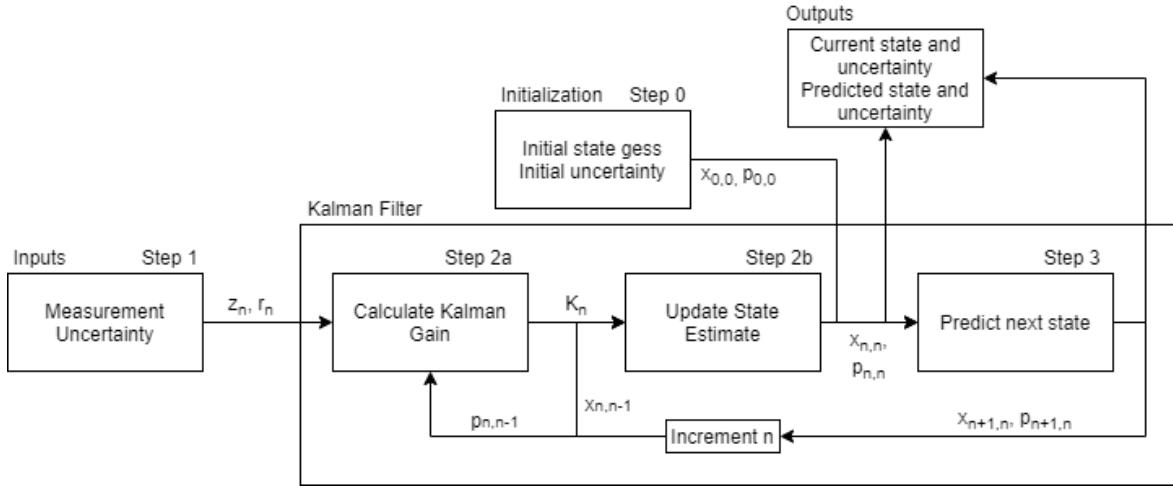


Wu et. al [52] propose a complimentary filter solution that reports being on par with more computationally expensive fusion algorithms like the Kalman Filter and Madgwick Filter. This algorithm computes the gravity vector using accelerometer and gyroscope measurements. By itself, this can yield the tilt attitude of the body. Then, if there is not severe magnetic distortion detected by the magnetometer, the full attitude can be acquired by fusing together the tilt attitude and that estimated from the magnetometer. The basic block diagram for the filter is in Figure 2.12.

2.3.5 Kalman Filter

The Kalman Filter [24] was introduced in the 1960's as a method to estimate and track a series of explicit and hidden states within a system. In a sensor fusion or AHRS application, the Kalman Filter can be used to estimate the orientation state (quaternion) of the system based on the gyroscope, accelerometer, and magnetometer readings. The Kalman Filter also estimates the uncertainty and noise associated with attitude estimate to give more insightful feedback on the usefulness of the estimate. At each timestep, the filter uses a built-in mathematical model of the system to predict the current system state and its associated uncertainty. Then, the filter can predict the state and uncertainty in the next time step and uses that to adjust a correction factor, called the Kalman Gain. With each iteration of the filter, the Kalman Gain converges on a minimum value and the estimate uncertainty decreases. The

Figure 2.13: A simplified process block diagram for the Kalman Filter that includes the steps required for each iteration.



convergence speed and accuracy can be tuned using various hyper parameters like process noise and sensor uncertainty. The process for the Kalman Filter is shown in Figure 2.13 and a more detailed explanation can be found in Appendix G.

2.3.6 Mahony Filter

The Mahony Filter [33] is a deterministic kinematic observer on the Special Orthogonal Group 3 that is driven by instantaneous attitude and angular velocity measurements. That is, the filter exploits a special relationship between observers to decouple gyroscope measurements from attitude estimates and then does on-line gyroscope bias compensation to increase accuracy. The filter proposes a model for the gyroscope in the body frame as:

$$\mathbf{q} = \mathbf{g}_u + \mathbf{b}_\omega + \boldsymbol{\mu}_\omega \quad (2.18)$$

where \mathbf{g} is the true angular velocity,

\mathbf{g}_u is the uncalibrated measured angular velocity,

\mathbf{b}_ω is the gyroscope bias vector, and

$\boldsymbol{\mu}_\omega$ is the gyroscope measurement noise

Additionally, the filter defines the accelerometer measurement as the instantaneous linear acceleration, (i.e. accelerations without the gravitational acceleration vector), with corresponding bias and noise vectors. The model for the accelerometer is given below:

$$\mathbf{a} = {}_B^G R (\mathbf{a}_u - \mathbf{G}_0 + \mathbf{b}_a) + \boldsymbol{\mu}_a \quad (2.19)$$

where \mathbf{a} is the true linear acceleration in the body frame,

${}_B^G R$ is the rotation matrix from the global to the body frame,

\mathbf{G}_0 is the gravitational acceleration vector,

\mathbf{a}_u is the uncalibrated measured angular velocity,

\mathbf{b}_a is the accelerometer bias vector, and

$\boldsymbol{\mu}_a$ is the accelerometer measurement noise

These measurements are used to express the body kinematics in an Explicit Complementary Filter in quaternion form and then rearranged to solve for the change in the body attitude over time, $\dot{\mathbf{q}}$. This estimate can be integrated over the sampling time to get the best estimated attitude via:

$${}^G_B \mathbf{q}_t = {}^G_B \mathbf{q}_{t-1} + {}^G_B \dot{\mathbf{q}}_t \Delta t \quad (2.20)$$

where ${}^G_B \mathbf{q}_t$ is the body attitude from the global frame to the local frame at time, t ,

${}^G_B \dot{\mathbf{q}}_t$ is the time rate of change for the body attitude at time, t , and

Δt is the sample interval

2.3.7 Madgwick Filter

The Madgwick Filter [32] considers body attitude as an optimization problem that can be solved by a gradient descent algorithm (GDA). The filter tries to optimize the difference (deviation) between a predefined reference vector in the global frame, ${}^G \mathbf{v} = [0, v_x, v_y, v_z]$ with

its corresponding measurement vector in the local frame, ${}^B\mathbf{s} = [0, s_x, s_y, s_z]$. This deviation is expressed in the objective function:

$$f(\mathbf{q}, {}^G\mathbf{v}, {}^B\mathbf{s}) = \mathbf{q} {}^G\mathbf{v} \mathbf{q}^{-1} - {}^B\mathbf{s} \quad (2.21)$$

The GDA can compute the solution to the objective function by utilizing an initial guess, ${}^G\mathbf{q}_0$ and a step-size, μ . This yields the estimate for the quaternion gradient, $\mathbf{q}_{\nabla,t}$. The filter then calculates the time rate of change in the body quaternion, $\dot{\mathbf{q}}_t$, as measured from the gyroscopes, $\mathbf{q}_{\omega,t}$. By removing the gyroscope error, β , from the estimated quaternion error, $\dot{\mathbf{q}}_{\epsilon,t}$ along the gradient, and numerically integrating the value over the sampling interval, Δt , the filter yields a quaternion estimate via:

$$\mathbf{q}_t = \mathbf{q}_{t-1} + \dot{\mathbf{q}}_t \Delta t \quad (2.22)$$

$$= \mathbf{q}_{t-1} + (\dot{\mathbf{q}}_{\omega,t} - \beta \dot{\mathbf{q}}_{\epsilon,t}) \Delta t \quad (2.23)$$

$$= \mathbf{q}_{t-1} + \left(\dot{\mathbf{q}}_{\omega,t} - \beta \frac{\nabla f}{\|\nabla f\|} \right) \Delta t \quad (2.24)$$

While the driving principle behind this filter is straight forward, the GDA has a convergence time that can lead to a high initial error during the algorithm start. The convergence time can be tuned using the β value and the overall error reduced by implementing the sensor models in Equations 2.5 and 2.14. The Madgwick filter is also computational and memory intensive as it requires many matrix multiplication operations and Jacobian calculations. On low-power, embedded platforms this can cause performance issues that may not be present with simpler models like the Fast Complimentary Filter [52].

2.4 Project Management Techniques

Any engineering project requires careful consideration to the design methodology and management structure. The methodology and structure dictates how quickly a project will

progress and how flexible it is to changes. Over the course of this thesis, several questions were considered: “how will the project adjust to design changes?” and “how can the project be structured such that it can be easily picked up in the future?”

2.4.1 Version Control

One of the first things considered was tracking changes in software and hardware using a version control system (VCS). Version control allows a team to track changes in source files and restore previous editions, or branch out to experiment with new features. Git⁵ is an industry standard VCS and was used extensively for the software and hardware of this project. Online services like GitHub allow developers to manage complex repositories of source code with version control, branches, issue tracking, and more. Typically, features are introduced as an issue ticket in the source repository, and then a specific branch is created to implement that feature. When completed and tested, the feature can then be introduced into the main code base by a pull request⁶. The same practice could be followed for bugs in the software, ensuring that everything is tracked and fixes did not balloon in size and break the working code. GitHub also provided a project planning tool⁷ that is able to plan out firmware releases and testing.

The hardware was designed in Autodesk Fusion360⁸ which has a built-in version control system more simplistic than Git. This was still utilized extensively and as a safety precaution, the final design revisions were also backed up to a GitHub repository.

With version control, the project was able to manage many different versions of firmware that were developed over years and various hardware platforms. The tools available made

⁵<https://docs.github.com/en/get-started/using-git/about-git>

⁶<https://docs.github.com/en/repositories/creating-and-managing-repositories/best-practices-for-repositories>

⁷<https://github.blog/2022-02-11-getting-started-with-project-planning-on-github/>

⁸<https://www.autodesk.com/products/fusion-360/overview?term=1-YEAR&tab=subscription>

design changes easier to implement and track and allows for new team members to easily come on-board in the future and have ready access to the latest firmware and hardware revisions, and the entire design history.

2.4.2 Systems Engineering

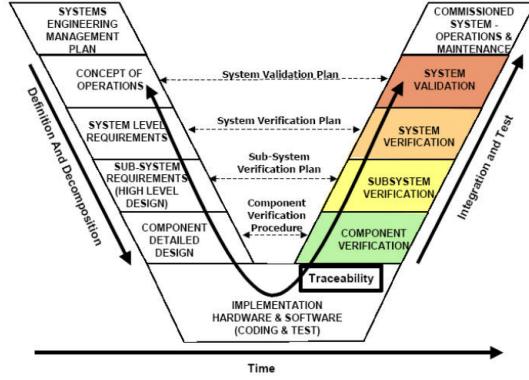
The project was designed using a hybrid systems engineering process that blended the V-methodology and the design spiral. The purpose of these two methods is to constantly iterate on a product until it has reached completion (i.e. the stakeholder requirements are met and the customer is satisfied). At the start of the process, the key problem has to be identified and then requirements drawn up for the system. At the beginning, these requirements start out broad and are best identified with stakeholders (the people who will be directly impacted by the project). Then, it is up to a systems engineer to expand the overarching requirements into capabilities and then subsystem- and component-level requirements.

The V-Methodology details a process from design to integration to verification and validation (Figure 2.14). Each step on the left side of the “V” (the start of the project) is verified by the same steps on the right side (the end of the project). The idea is that by building good requirements at the start and verifying them at the end, the project will be “successful” with clear expectations outlined at the beginning and a traceability matrix (Appendix H) to track requirement progression and achievement. The engineering and management teams can follow the V from left to right and ensure that their product is on-track while mitigating “feature-creep”. As the product is developed, the design volleys between the left and right sides, slowly iterating back up to the top of the V shape. This process is described in more detail in Allouis et al [3].

2.4.3 Agile Methodology

The Agile methodology is a relatively new concept introduced in the mid-1990’s that provided a radical new method for doing software development. Instead of working in siloed teams based on a broad set of requirements and a large, expansive Gantt chart, programmers were

Figure 2.14: The system engineering V-method showing the flow of the project. Courtesy of Allouis et. al [3].



encouraged to start working together in smaller teams to tackle smaller issues and constantly test and present their progress. Team leaders would have an easier time managing the smaller teams and workloads, and project managers would have a better way of tracking product progress. The scrum methodology was also introduced that radically changed the team dynamic and has evolved to improve efficiency in a wide variety of industries [44].

The Agile methodology was implemented on the project so that tasking loading remained light and features could be implemented sprint-by-sprint. The testing recommended by the Agile method meant that products could be more quickly verified and validated against, allowing for quicker iteration. This meant the project could adapt quickly and provide a framework for new people to start contributing to the project, if necessary.

2.5 A Review of the State of the Art

Inertial sensing is not a new problem to be solved. Over the last few decades many solutions have been released with increasingly better performance and efficiency. The advent of MEMS technology drastically shrunk sensor sizes and made them cheaper to use. We shall examine two different categories of solutions and compare solutions within those categories.

Figure 2.15: Sparton AHRS-8. Courtesy of Sparton [43]



2.5.1 Industrial Solutions

Industrial solutions are provided by companies to customers that require performance and reliability for their sensors. They will typically be employed in the industrial robotics or automotive industries and integrated within a larger system. Thus, they cannot provide standalone services, and do not satisfy the objectives for this thesis. But, they represent the cutting edge techniques for sensor fusion. The following solutions also are the “more affordable” variants in order to limit the scope of the review. Even so, each solution is prohibitively expensive and therefore out of the scope of the thesis objectives.

Sparton AHRS-8 The first industrial sensor to consider is the Sparton AHRS-8 [43] and is one of the cheapest at \$1,350 per unit. This is an AHRS designed to be interfaced over USB or Serial and uses Sparton’s proprietary AdaptNav™ sensor fusion algorithm. This gives a sub 1-degree accurate measurement of body attitude and heading at a range of temperatures and motions. However, this sensor lacks GPS/GNSS integration, wireless features, and independent data logging.

Xsens Mt-100 The second sensor to consider comes from the Movella/Xsense corporation and is their MTi-100 IMU [36]. This unit costs \$1,809 and provides an 9-DOF IMU with an RS232, RS422, RS485, UART, and USB output for integration on larger platforms. Like the Sparton, it runs a proprietary sensor fusion algorithm and does not have GPS/GNSS integration. But, it does have a software development kit (SDK) provided by the manufacturer to more easily integrate it and tweak settings. The unit boasts a sub 1-degree accurate

Figure 2.16: The Xsens Mti-100 IMU. Courtesy of Movella [36].



Figure 2.17: The VectorNav VN-200 IMU. Courtesy of VectorNav [49].



orientation estimate over a range of temperatures and motions.

VectorNav VN-200 From the VectorNav corporation comes the VN-200 AHRS [49]. The unit boasts a <0.2-degree accurate attitude estimation and integrates GPS/GNSS. However, it relies on a USB or Serial interface and uses closed source and proprietary firmware for sensor fusion. It also does not have wireless or independent data logging functionality and costs over \$3,000.

SwiftNav Duro The SwiftNav Duro is a centimeter-accurate GNSS real-time kinematic (RTK) solution that incorporates a high accuracy IMU for inertial navigation [45]. This unit is developed for robotics applications that need a bolt-on solution for autonomous navigation and positioning. The unit comes in a rugged IP67 enclosure and has multiple interfaces such as USB, Serial, and Ethernet for communications. However, it does not have wireless or independent data logging functionality. The firmware is also closed source and it costs \$5,795.

Figure 2.18: The SwiftNav Duro Inertial. Courtesy of SwiftNav [45].



Figure 2.19: The Aceinna INS401. Courtesy of Aceinna [1].



per unit.

Aceinna INS401 Rounding out the industrial solutions is the Aceinna INS401 which is a rugged centimeter-accurate inertial navigation system (INS) for automotive and industrial autonomous systems [1]. The unit is closed source, but provides a developer-friendly SDK and API for integration on a larger platform. Like its cousins, this unit also does not have wireless connectivity or independent data logging functions, but can communicate over several hard-wired methods.

x-io Technologies x-IMU3 Recently released in June 2023, the x-IMU3 has almost the exact same capabilities as the solution presented in this thesis [46]. This device was designed by the author of the Madgwick algorithm, and is the third iteration of his IMU data loggers.

Figure 2.20: The x-io Technologies x-IMU3. Courtesy of x-io Technologies [46].



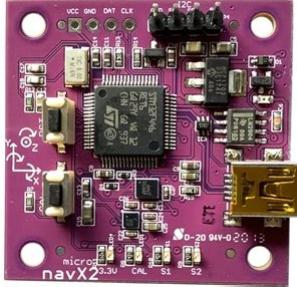
The x-IMU3 has a very low SWaP and comes in a IP67-rated enclosure with a USB and button interface. The USB interface can be used to directly link the device and offload files, or it can be used with external peripherals. Similarly, the x-IMU3 has extensive wireless support and can stream data back to a host application over WiFi (via TCP or UDP) or over BlueTooth.

The x-IMU3 and the thesis were developed independently, but the final products are near to each other in both cost and capability. In fact, the x-IMU3 was a major influence on the final firmware revision for the thesis and helped implement key features. The creator, Dr. Sebastian Madgwick, has also been helpful in critiquing the thesis design and offered key insights over multiple interviews [19]. We shall carefully consider this solution going forward and directly compare it to the thesis product in Chapter 5.

	Sparton AHRS-8	Xsens Mti-G-710	VectorNav VN-200	SwiftNav Duro	Aceinna INS401	x-IMU3
Cost	\$1,350	\$5,766	\$3,050	\$5,795	Unknown	\$410
Attitude Accuracy	<1°RMS	<1°RMS	<0.2°RMS	<1°RMS	<0.1°RMS	<1°RMS
GPS		✓	✓	✓	✓	
On-Board Storage						✓
Wireless						✓
Standalone						✓

Table 2.2: A comparison of the industrial solutions discussed above.

Figure 2.21: The Kauai Labs NavX2 Micro. Courtesy of Kauai Labs [25].



2.5.2 Hobbyist Solutions

Stepping down an order of magnitude in pricing and performance are the hobbyist sensors. These can be purchased by companies such as Adafruit and SparkFun and HobbyBro for tens, if not a couple hundred of dollars per unit, making it much more reasonable for the average consumer to use on their projects. The hobbyist solutions are much more in line with the objectives of the thesis and should therefore be more carefully considered.

Kauai Labs NavX2 Micro The NavX2 micro is derived from the popular NavX board that is routinely used in the First Robotics Competition for high school robotics [25]. The NavX2 micro shrinks the sensor down into a low SWaP and cost package that utilizes open source software. This makes it well-suited for its application in hobbyist robotics where even basic inertial navigation is a coveted feature. The unit costs \$89 and can be integrated onto a larger platform via a USB or Serial interface. However, it does not have wireless connectivity or independent data logging. For these reasons, the NavX2 micro will not suit the objectives of this thesis.

SparkFun OpenLog Artemis The OpenLog Artemis (OLA) is a hobbyist solution offered by SparkFun for \$55 [42]. It comes equipped with a 9-DOF IMU, battery integration, a real-time-clock, ADC, and an interface for easily integrating other sensors. The unit is pre-loaded with firmware that automatically begins logging when it is powered on and the software is open source and can be freely modified. It even has a BlueTooth interface that can be used to wireless monitor data. This solution meets the thesis objectives, but the code would have

Figure 2.22: The SparkFun OpenLog Artemis. Courtesy of SparkFun [42].



Figure 2.23: The Holybro Pixhawk 6X FCU. Courtesy of Holybro [23].



to be modified to perform sensor fusion for orientation estimation and an IP67-rated case would have to be designed for it.

Pixhawk FCU The Pixhawk Flight Controller Unit (FCU) is designed for hobbyist robotics applications such as drones (UAVs) and remotely operated vehicles [23]. It runs a sophisticated open source hardware and software stack that fuses a 9-DOF IMU with GPS data to get position and attitude with a reasonable degree of accuracy. The sensor fusion is performed with multiple Extended Kalman Filters and can be logged internally to a microSD card. By using a radio link over the serial bus, data can be wirelessly transmitted to a ground station for monitoring and logging.

This solution fits the objectives of the thesis, however it is overkill for the intended application and the MAVlink interface used to communicate with it is not intuitive. Because data logging is only meant for post-flight analysis if the vehicle crashes or malfunctions, there are significant limitations on what can be logged or how fast. For these reasons, the Pixhawk was not considered as a solution for the thesis.

Chapter 3

How I Made It Work

3.1 Stakeholders

As with any project, stakeholders are a crucial part of the design process. Stakeholders drive overarching requirements that the device must achieve in order to be accepted for operation. On an individual level, the committee overseeing this thesis are the major stakeholders as they have a vested interest in the success or failure of the project. However, some organizational-level stakeholders have also expressed interest in the project and provided some additional requirements. A summary of the stakeholders is provided in Table 3.1.

3.1.1 Hands-On Users

Thetis is intended to be used by college students that are at least sophomore-level and have a basic understanding of instrumentation and electronics. The device is simple to use, requiring the user to flick a switch to turn it on or off, and then press a physical or digital button to start or end logging. Thetis is also designed to accommodate advanced users who need to leverage different features of the device for their research or want to tinker with it. It can also be used by laboratory technicians and instrumentation engineers to aid with data-collection experiments for tracking three dimensional bodies

Table 3.1: A summary of stakeholders for the Thetis device

Stakeholder	Description
Committee members	Individual professors who have expressed interest in the project and have agreed to assist in its development. Specifically, Dr. Wood and Dr. Weaver would like to deploy Thetis on university projects; Dr. Gutierrez has provided many requirements on the performance of the instrumentation; and Dr. Silaghi is interested in its application to autonomous navigation.
Florida Institute of Technology	The university has several classes and projects where Thetis could be useful. The Instrumentation Design and Analysis class used Thetis as a demonstrator for designing PCBs and field experiments. Thetis was also designed with Surf Engineering Analysis in mind for students to have a new open source sensor to experiment with.
NSWC Carderock - Combatant Craft Division (CCD)	This three-letter agency of the government has expressed interest in using Thetis as a testing and evaluation tool for small unmanned crafts

3.2 Design Rationale

Thetis is envisioned as an open-source all-in-one inertial data logging solution for use in research projects. The device incorporates multiple sensors, GPS tracking, and a WiFi-capable microcontroller in order to enable as many features as can be envisioned by the end users. One of the driving considerations was the small footprint. Thetis Revision F is designed to fit into a small IP67-rated enclosure. This tiny form factor allows it to be inconspicuously mounted to any floating body like surfboards, scale models, or wave buoys without upsetting their inertial characteristics or impeding nominal operation.

3.2.1 Problem Description

I originally envisioned Thetis as the solution to a couple of problems I encountered during my undergraduate studies. For my fluid mechanics laboratory section, we conducted an experiment where we needed to track the motion of a model pontoon boat as we adjusted its center of gravity. Because the lab lacked any way to directly measure the body's inertial movement, inclinometers were placed on the hull and the students had to conduct a time

motion analysis from a video recording to plot the roll angle frame-by-frame. This was inefficient and made me start thinking about how this could be improved.

The second problem occurred in a class the following semester where we deployed surf boards and pressure gauges into the ocean to monitor a surfer's ride along a wave. We used two different instruments, the Lowell MAT-1 data logger [29], and an iPhone 6 inside a life proof case. The Lowell instrument was not designed for this use case. First, the sensor board is housed within a cylindrical body, making properly alignment to a Cartesian (square) coordinate system almost impossible. Second, the sensor only had an accelerometer and magnetometer on board. It reported roll, pitch, and yaw but the roll and pitch were derived from tilt and gravitational acceleration and the magnetometer determined yaw. These values assumed the sensor would be in the vertical position, and we used it in a horizontal position. Suffice to say, we were not using the Lowell properly and could not get reliable or useful information out of it for our analysis.

The iPhones were much better suited as they had a full 9-DOF IMU sensor suite and a great data logger application to get the raw sensor values and sensor fusion results. However, since this was a class comprised of inexperienced undergraduates, we didn't check that the seal on the phone cases were properly set before going into the water. Within a matter of months, both iPhones were lost because students removed the case and did not replace the water-tight gasket, costing a large sum of money.

Given my experience with these classes, it was clear to me that a low-cost inertial data logging solution could be valuable to the department as a teaching and laboratory instrument.

3.2.2 Mission Statement

Thetis aims to democratize the inertial measurement and tracking space for small scale experiments by implementing an open-source, feature-rich, all-in-one solution to monitoring the movements of bodies.

Table 3.2: A slimmed down traceability matrix for the stakeholder requirements. Verification and Validation status is not shown here. The priority of each requirement is the relative weight of requirement compared with all others.

ID	Description	Weight	Priority	Status
SR 01	The system shall be able to record acceleration, rotation rate, orientation, and position	1	14	TIP
SR 02	The system shall be able to fit into a small IP67-rated, or better, enclosure	1	14	D
SR 03	The system shall be able to be powered by battery for more than 4 hours continuously	1	14	A
SR 04	The system shall be cheaper than \$200 per unit	0.4	6	A
SR 05	The system shall use components that are readily available COTS	1	14	D
SR 06	The system designs shall be open source for modification by students	0.7	10	D
SR 07	The system shall allow users to change settings via interface and/or configuration file	0.7	10	D
SR 08	The system shall communicate extra-device using WiFi and USB	0.3	4	A
SR 09	The system shall have enough on-board storage for 4 hours of continuously logging at 64 Hz	1	14	TIP

3.2.3 Stakeholder Requirements

Interviews with the stakeholders occurred over several months and informed a set of requirements that they determined were necessary for the project's success. These overarching requirements were placed into a traceability matrix (Table 3.2) to track their priority and status throughout the project.

3.2.4 Risk Identification

The following tables explore some of the larger risks associated with each of the stakeholder requirements. Risks are categorized into several types: Technical, Cost, Schedule, Organizational, and Operational. Each risk has an associated consequence on the project’s development and a severity rating from “Low” to “High”. With every risk, we can apply a mitigation strategy to reduce the severity of the consequence.

Technical Risks These are risks that pose a technical challenge to the project such as sensor accuracies, programming difficulty, or component specifications. These risks are likely to impede the development of Thetis by showing up during testing and forcing workarounds after the fact, rather than before.

Cost Risks These risks pose a threat to Thetis’s development budget. They are more likely to manifest in the supply chain of Thetis, especially in the wake of the COVID-19 pandemic and the chaos of the chip shortage.

Schedule Risks By delaying the program’s development, Schedule Risks are the most notorious and costly. These risks manifest as delays in development or the supply chain and can significantly impact the expected delivery time of the device. They also are prominent across all phases of the design process from planning to testing and are only mitigated, but never eliminated during the design process.

Organization Risks The stakeholders represent various organizations that have an interest in Thetis. This risk type poses a problem for the specific stakeholder organization and their interests. This can be viewed from the political, economical, or temporal points of view as these are important considerations for the stakeholders.

Operational Risks These risks challenge end users of the device while they are testing. Most of them can be mitigated in the design process by working directly with the end users and ensuring their concerns are met. However, the majority of them will only manifest during

testing and must be mitigated after the fact. This could mean substantial cost increases and schedule delays depending on the severity of the risk.

SR 01 - The system shall be able to record acceleration, rotation rate, orientation, and position

Stakeholder:	Thesis Committee				
Rationale:	The system needs to be able to record the inertial characteristics of a floating body				
Fit Criterion:	This will be accomplished using a 9-DOF IMU and GPS receiver with accuracies not to exceed one standard deviation of a reference source				
Risk	Risk Issue	Risk Consequence	Initial Risk	Risk Mitigation	Risk After Mitigation
Technical Assessment	The IMU and/or GPS will report measurements that have a high margin of error and little consistency	Worthless data for analysis	Medium	Selection of sensors that have decent accuracy and low drift. Use a sensor calibration model to improve reported sensor accuracy	Low
Cost Assessment	Chip shortage as a result of the COVID-19 pandemic.	Unable to find appropriate components. Any found components are prohibitively expensive	Medium	Find components that are in stock and order in bulk. Rework design based on available components.	Medium
Schedule Assessment	Calibration is time consuming to implement, verify, and validate	Schedule overrun trying to tune the algorithms. Inaccuracies introduced through improper tuning.	Medium	Good programming practices to make tuning easier during testing. Good understanding of process and mathematics.	Low
Organizational Assessment	Lack of subject matter experts	Improper implementation and design of sensor and embedded systems.	Medium	Thorough background review and study of embedded system design Consult with external experts.	Low
Operational Assessment	Unreliable sensors.	Device fails and does not recover during testing; lost data	Medium	Reliability analysis and testing required. Consult and follow FMECA.	Low

SR 02 - The system shall be able to fit into a small IP67-rated, or better, enclosure

Stakeholder: Thesis Committee					
Rationale: Thetis's electronics need to be independently sealed from the elements, including being temporarily submerged					
Fit Criterion: The selected enclosure should have a UL-listing and have a verified IP rating that fits the requirement.					
Risk	Risk Issue	Risk Consequence	Initial Risk	Risk Mitigation	Risk After Mitigation
Technical Assessment	Unable to find a small enough IP-rated enclosure	Need to increase size that may violate size restrictions	Medium	Extensive searching of online retailers of catalogs	Low
Cost Assessment	Small IP-67 rated enclosures are a niche item that may have to be custom-ordered and therefore expensive.	Enclosures may be prohibitively expensive and drive up program costs	Low	Find reputable suppliers and order enclosures in bulk, if possible.	Low
Schedule Assessment	If the enclosures have to be custom ordered, there may be an extended lead time for delivery	Schedule overruns	Medium	Find reputable suppliers and order in bulk well ahead of time. Maintain a stock of several enclosures as spares.	Low
Organizational Assessment	N/A	N/A		N/A	
Operational Assessment	IP-rating is not valid. Enclosure is not fully sealed. Operation exceeds rated limits	Chamber floods, damaging equipment, causing a potential fire hazard and data loss.	High	Reputable supplier and traceable IP-rating. End user vigilance to ensure enclosure is fully closed and sealed properly.	Medium

SR 03 - The system shall be able to be powered by battery for more than 4 hours continuously

System Requirements Analysis					
Stakeholder:	Thesis Committee				
Rationale:	The system is expected to deploy for a minimum of 3 hours on a typical deployment. Need margin for battery chemistry and preparation time.				
Fit Criterion:	The battery shall have an appropriate capacity. Device will be tested with extended duration deployment				
Risk	Risk Issue	Risk Consequence	Initial Risk	Risk Mitigation	Risk After Mitigation
Technical Assessment	Device is not turned off before battery voltage is critically low	Battery is damaged and needs to be replaced; fire hazard	Medium	Build the power regulator such that the device auto turns off before battery is damaged Integrate battery monitoring and reporting	Low
Cost Assessment	Larger capacity batteries are more expensive. Damaged batteries need to be replaced	Cost overruns	Low	Order batteries in bulk and ensure spares are available	Low
Schedule Assessment	If there are no spares, there may be delays in testing schedule	Any critical testing may have to be delayed, resulting in program delays	Low	Order batteries in bulk and ensure spares are available	Low
Organizational Assessment	N/A	N/A		N/A	
Operational Assessment	Battery does not have enough capacity to support a 4 hour deployment	Data is lost	Medium	Conservative power analysis and choosing an oversized battery. Implement a battery monitoring circuit to force save and shutdown at low voltage levels.	Low

SR 04 - The system shall be cheaper than \$200 per unit

Stakeholder: Thesis Committee					
Rationale: Thetis is designed to be a cheap, expendable unit so that when damaged it is financially feasible to replace it. It needs to be cheaper than comparable devices.					
Fit Criterion: The Bill of Materials for Thetis shall not exceed the required threshold.					
Risk	Risk Issue	Risk Consequence	Initial Risk	Risk Mitigation	Risk After Mitigation
Technical Assessment	Cheaper components (sensors) that are not as precise or accurate.	Less accurate data that may not be useful for analysis	Medium	Use a sensor calibration model to improve reported sensor accuracy	Low
Cost Assessment	Chip shortage as a result of the COVID-19 pandemic. Small IP-67 rated enclosures are a niche item that may have to be custom-ordered	Parts significantly inflate the Bill of Materials cost	Meidum	Source from reputable suppliers and purchase in bulk	Medium
Schedule Assessment	N/A	N/A		N/A	
Organizational Assessment	Bill of Materials cost exceeds requirement limit. Development costs become too expensive	Invested stakeholders may pull support. Hardware development slows or stops	High	Purchase materials in bulk, when possible. Use proper development techniques and analysis to reduce the number of hardware iterations and cost	Medium
Operational Assessment	Cheaper, not as reliable parts.	Unexpected and unrecoverable failures during deployments	Medium	Reliability analysis and testing required. Consult and follow FMECA.	Low

SR 05 - The system shall use components that are readily available COTS

System Requirements Analysis: SR 05					
Stakeholder:	Thesis Committee				
Rationale:	Using commercial off the shelf parts will reduce development time and costs. COTS parts will also make repair and maintenance easier.				
Fit Criterion:	All parts used on Thetis must be available from commercial suppliers and catalogs.				
Risk	Risk Issue	Risk Consequence	Initial Risk	Risk Mitigation	Risk After Mitigation
Technical Assessment	N/A	N/A		N/A	
Cost Assessment	Chip shortage as a result of the COVID-19 pandemic. Small IP-67 rated enclosures are a niche item that may have to be custom-ordered	Parts significantly inflate the Bill of Materials cost	Medium	Source from reputable suppliers and purchase in bulk	Low
Schedule Assessment	Parts are not available. Small enclosure may have a long lead time	Development schedule should be delayed	Medium	Order parts in bulk and ensure spares are available	Low
Organizational Assessment	The small enclosure may no longer be available from a commercial supplier	Forces a new design iteration to fit a new enclosure. Forces end users to adapt to a new enclosure	Low	Work with supplier to ensure supply chain; order in bulk. 3D print enclosure, if COTS is no longer available	Low
Operational Assessment	N/A	N/A		N/A	

SR 06 - The system designs shall be open source for modification by students

Stakeholder: Thesis Committee					
Rationale: Thetis will need to be maintained while students use it for class projects. Therefore, they will need to have the ability to directly modify the hardware and software.					
Fit Criterion: All of the code and hardware will be published open source in GitHub repositories under an MIT license.					
Risk	Risk Issue	Risk Consequence	Initial Risk	Risk Mitigation	Risk After Mitigation
Technical Assessment	Designs are modified by students who unknowingly break functionality	Disabling or “bricking” Thetis, making it useless. Loss of development history and/or “good” designs	Low	GitHub tracks all commits and history. “Good” designs are published as release snapshots and cannot be lost. Only allow new features to be created through forks	Low
Cost Assessment	New design modifications require more investment	Program cost increases as students change the design	Low	Ensure that any design modifications go through a design review and adds substantial value to Thetis	Low
Schedule Assessment	New designs may require additional development time	New additions may not be ready in time for when students need them	Low	Ensure a ready supply of spare parts. Proper design practices to ensure students can easily add features	Low
Organizational Assessment	Thetis design may not be patentable and commercialized	Thetis cannot be patented since it is published open source	High	Thetis can still be commercialized by providing board assembly and support services	Low
Operational Assessment	Vulnerabilities are found and exploited by bad actors	User data can be compromised WiFi capability opens an attack vector to local network	Low	Avoid improper implementation of OTA software updates Use proper WiFi network security techniques.	Low

SR 07 - The system shall allow users to change settings via interface and/or configuration file.

System Requirements Analysis					
Stakeholder:	Thesis Committee				
Rationale:	Having settings the users can change will make it easier to configure on deployments and lower the knowledge threshold required to use Thetis in different applications.				
Fit Criterion:	An Application Programming Interface (API) will be interfaced that will modify a settings file that can also be accessed by the user.				
Risk	Risk Issue	Risk Consequence	Initial Risk	Risk Mitigation	Risk After Mitigation
Technical Assessment	Users can overwrite certain configurations and compromise sensor performance	Thetis will not provide accurate or useful data measurements	Medium	Implement read-only for critical settings and prevent users from overwriting them outside a “factory” mode.	Low
Cost Assessment	N/A	N/A	Medium	N/A	Medium
Schedule Assessment	Settings implementation and API will need to be custom	Development time will need to be spent on a custom API and implementation	Medium	Thorough background research to see if library solutions exist and can be adapted.	Low
Organizational Assessment	Students may not know how to edit or revert settings	Settings will not be able to be changed by future students	Low	Extensively document API and provide a README or tutorial for editing settings	Low
Operational Assessment	Configurations are lost or not saved between deployments	Users may lose custom settings and need to reload. Thetis may crash with corrupted configurations	Medium	Backup settings to data storage device. Have default settings config that loads on issue to prevent instability	Low

SR 08 - The system shall communicate extra-device using WiFi and USB

Stakeholder: Thesis Committee					
Rationale: A physical USB connection is crucial for programming and debugging Thetis. A WiFi connection would allow for wireless programming and data off-loading					
Fit Criterion: The microcontroller shall have a USB-compatible PHY and WiFi capability.					
Risk	Risk Issue	Risk Consequence	Initial Risk	Risk Mitigation	Risk After Mitigation
Technical Assessment	WiFi performance (speed/range) is too low and unacceptable	Data offloading is cumbersome. Unable to monitor device while deployed	Medium	Select microcontroller with known good WiFi-performance. Select an appropriate antenna for the situation	Low
Cost Assessment	WiFi is not useful and not fully utilized	Cost per microcontroller is needlessly increased; inflated bill of materials cost	Medium	Verify that WiFi functionality is desired by Stakeholders and end-users	Low
Schedule Assessment	Implementing WiFi functionality is difficult and increases complexity; feature creep.	Increased time to develop front end interface; schedule overruns	High	Agile management to limit feature creep	Medium
Organizational Assessment	Some stakeholders may have electromagnetic emissions restrictions	To comply, WiFi functionality will need to be able to be disabled	Low	Allow WiFi features to be disabled by uploading new firmware without those features. Allow users to toggle this function on/off in runtime using a configuration file.	Low
Operational Assessment	Operators may not have a device that can connect to Thetis over WiFi	Data cannot be offloaded easily in-situ. Wireless monitoring not possible	Low	Ensure that there is always a physical backup for offloading data. Physical lights or other indicators to display system status	Low

SR 09 - The system shall have enough on-board storage for 4 hours of continuously logging at 64 Hz

Stakeholder: Thesis Committee					
Rationale: This will be deployed for several hours and needs to continuously record data. Data rate should be as fast as possible to capture higher frequency oscillations.					
Fit Criterion: Storage devices shall be large enough to support a full-length deployment and record all data as fast as possible (>64 Hz)					
Risk	Risk Issue	Risk Consequence	Initial Risk	Risk Mitigation	Risk After Mitigation
Technical Assessment	Logger is not able to consistently achieve 64 Hz rate	Significant data loss on higher frequency oscillations	Medium	Utilize fast storage mediums. Implement proper sensor interrupt practices	Low
Cost Assessment	Small flash chips will not be sufficient to store the amount of data	More expensive data storage devices (micro SD cards) will have to be used; increase bill of materials cost	Medium	Purchase storage devices in bulk and at the minimum capacity.	Low
Schedule Assessment	Optimizing the logging functionality will take significant development time to be consistently as fast as possible	Schedule overruns	High	Proper programming practices to optimize code wherever possible; reduce refactors	Medium
Organizational Assessment	N/A	N/A		N/A	
Operational Assessment	Inertial movements and oscillations will occur at higher frequencies than can be reasonably measured (>16 Hz)	Significant decrease in data accuracy and correlation to real-world motions	High	Allow users to change sample rate in-situ using configuration settings. Validate that sensors can accurately record expected signals.	Medium

3.2.5 Quality Functional Deployment

In engineering, there are a variety of customers that need to have their needs met. These customers can be internal or external to the project and are described in Section 3.1. To ensure that Thetis is designed with the stakeholders in mind, a Quality Functional Deployment (QFD) was performed. A QFD weighs the stakeholder requirements (the “whats”) with the “hows”, or engineering methods, in a weighted design matrix. The QFD analysis is designed to inform engineers what are the highest priority methods, based on customer feedback. This gives stakeholders a direct influence on the design in a compact matrix format. The product created by a QFD analysis is the House of Quality (HOQ), which can be seen in Table 3.12

The left side of the matrix defines the customer desires with an associated weight or priority for each. This data is collected by canvassing the stakeholders and getting their feedback on their priorities.

The middle of the matrix is dedicated to the engineering methods. Here, the design engineers add the methods they think are required to fulfill the projects requirements. The numbers throughout the matrix are the strength of the relationship between the requirement and method. The scale is the Fibonacci sequence from 0 to 8¹ with higher numbers indicating a stronger relationship. For example, SR-03 has a middling relationship with most of the engineering methods and a very strong (8) relationship with the “Battery Capacity” method. However, it does not correlate at all (0) with the storage capacity method.

The “roof” or top of the matrix is the inter-relationship between the different engineering methods. The grading is slightly different with “-”, “0”, “+” indicating a detrimental, non-existent, or positive relation, respectively, with multiple of each denoting the magnitude. As an example, the cost of Thetis has varying degrees of negative association with each of the

¹0, 1, 2, 3, 5, 8

other engineering methods. The type of IMU, GPS, storage method, battery capacity, and enclosure size all add cost to the project and therefore are detrimental to this method.

The right side of the matrix is part of the competitive analysis where Thetis is compared with the other products in use by stakeholders. Each product is rated on a 1-5 scale of increasing quality for each stakeholder requirement. This rating is produced based on their capabilities and performance as determined by the stakeholders. From the matrix, we can see that Thetis has a strong overall rating with regard to all the requirements where the Lowell MAT-1 struggles with the cost and measurement requirements and the iPhone is strong overall except its price and access for students.

At the bottom of the matrix is the technical performance analysis. The first two rows rate each engineering method with an importance which is described by Equation 3.1 and 3.2. The importance of each method informs the design engineers what they should be focussing on during development and what is most important to the stakeholders. Beneath that is a technical comparison of Thetis and competing products across the engineering methods. This provides a quick breakdown of each product through the lense of the engineering methods, providing some insight to stakeholders of the benefit of Thetis over competitors.

$$I = \sum_{n=1}^N w_n \times R_n \quad (3.1)$$

where I is the engineering method's importance

N is the number of stakeholder requirements

w_n is the weight of a stakeholder requirement, n

R_n is the strength of the relationship between the requirement and method

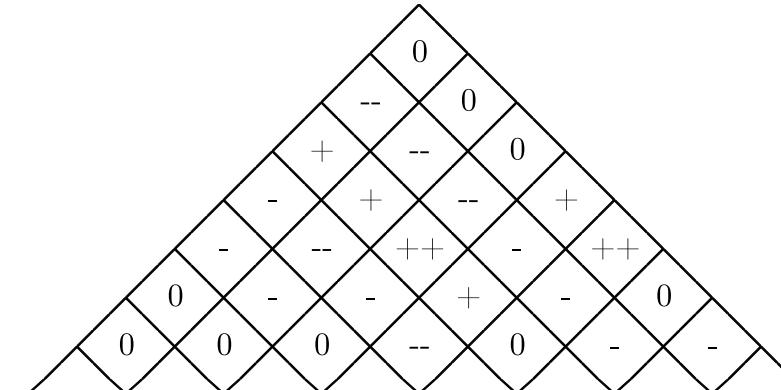
$$RI = \frac{I_m}{\sum I} \quad (3.2)$$

where RI is the engineering method's relative importance

I_m is the current method's importance

I is the vector of all the method's importance

Table 3.12: The House of Quality matrix for the Thetis instrumentation package.



		House of Quality Matrix											
		Stakeholder Requirements			Product Features								
		Weight	IMU	GPS	Storage Capacity	Battery Capacity	Size	Sample Rate	Cost	Connectivity	Thetis	Lowell	iPhone
SR 01	1	8	8	0	0	0	2	0	0	5	3	5	
SR 02	1	0	5	5	8	8	0	5	2	5	3	4	
SR 03	1	3	5	0	8	5	3	2	3	5	5	5	
SR 04	0.4	5	5	2	5	5	0	8	3	5	1	1	
SR 05	1	5	5	3	3	5	0	8	5	5	3	1	
SR 06	0.7	3	3	3	1	5	0	2	3	5	1	1	
SR 07	0.7	8	8	8	0	0	1	0	3	5	5	5	
SR 08	0.3	0	0	0	2	2	0	1	8	5	3	5	
SR 09	1	3	1	8	0	0	8	1	0	5	5	5	
Importance Weight		28.7	33.7	24.5	22.3	24.1	13.7	20.9	17.8				
Relative Importance		0.15	0.18	0.13	0.12	0.13	0.07	0.11	0.10				
Measurement Unit		DOF	Meters	GB	Wh	Cu. cm	Hertz	US Dollars	Ways				
Target Value		9	3	64	1.5	73.7	64	\$ 200.00	3				
Actual Value		9	3	64	1.5	73.7	100	\$ 150.00	3				
Lowell Instrument		6	-	64	1.5	122	64	\$ 900.00	1				
iPhone		9	3	128	6.7	63.8	100	\$ 649.00	2				

3.3 Concept of Operations

Thetis is envisioned to be an all-in-one data logging solution that can provide real-time calculations of body orientation and position at a low cost. In a normal operating case, Thetis will be placed near the center of gravity of a body and turned on to begin recording. The user can then connect Thetis to a larger wireless network or use its local WiFi access point to monitor the system and adjust configuration settings.

3.4 Capabilities

Based on the stakeholder requirements and concept of operations, a series of capabilities were derived that influenced the subsystem and component-level requirements. These capabilities were organized into three categories:

Threshold: The bare minimum features the system needs to be have to be considered “successful” or “delivered”.

Reach: Features that enhance the product for the stakeholders and end users but require a moderate amount of effort above a threshold feature.

Stretch: Features that would distinguish the project from competitors and add immense value for stakeholders and end users, but come at a steep development cost in terms of time and money.

The capabilities were organized into a traceability matrix (more information in Appendix H), which is shown in Tables 3.13 through 3.15.

3.5 Conceptual Design

The general design of the Thetis instrumentation package was built around the principles of capability, manufacturability, and affordability. The parts selected for the design had to offer a suite of capabilities that would enable a wide range of projects. While Thetis was designed

Threshold				
ID	Description	Weight	Priority	Status
101	The system must be housed in an IP67-rated enclosure	1	10	D
102	The system enclosure must fit within the volume of 8" x 5" x 1.25"	0.8	8	D
103	The system software and firmware will be fully open-source	0.9	9	D
104	The system shall record inertial measurements at a minimum frequency of 64 Hz	1	10	TIP
105	The system shall be able to store data locally for up to 4 hours continuously	0.8	8	TIP
106	The system shall be able to operate for up to 4 hours continuously	0.8	8	A
107	The system shall have simple human interface mechanism for status and logging	0.7	7	D
108	The system firmware shall be open-architecture	0.5	5	A
109	The system will be fully documented	1	10	A
110	The system shall use version control software to track changes	1	10	D
111	The operator shall be able to offload data from the system	1	10	D
112	The system shall be capable of being assembled by hand using basic soldering tools	0.8	8	D

Table 3.13: Thetis threshold capabilities organized in a traceability matrix without the verification and validation fields present.

primarily for monitoring the movement of floating bodies, it can be adapted for use in general robotics, embedded AI research, and other datalogging tasks. Thetis is also envisioned to be continued by other endeavoring students, so it has to be easily manufacturable with a single-sided design that can be assembled by hand using tweezers and a hot-air gun. All of this had to be designed with low-cost readily-available components in mind. Thetis was originally designed at the start of the global chip shortage in 2020, so parts were limited in

Reach				
ID	Description	Weight	Priority	Status
201	The system shall use a GPS with a minimum 1 Hz update rate for position tracking	0.7	6	D
202	The system will have a simple logging and status interface accessible via web terminal	0.3	2	A
203	The system shall be able to monitor and report battery state of charge	0.2	2	A
204	The system will have configurable settings that can be changed by the operator	0.4	3	D
205	The system can operate in a Wi-Fi access point or client mode	0.2	2	A
206	The sensor measurements shall incorporate a calibration model	0.5	4	TIP

Table 3.14: Thetis reach capabilities organized in a traceability matrix without the verification and validation fields present.

Stretch				
ID	Description	Weight	Priority	Status
301	The system file storage shall be accessible via web or USB interface	0.1	0.8	A
302	The system shall have a backup storage option in case primary storage fails	0.1	0.8	R
303	The system will use a microcontroller capable of machine learning using TinyML	0.05	0.4	UR
304	The system shall be able to log in a burst mode for up to 24 hours	0.05	0.4	UR

Table 3.15: Thetis stretch capabilities organized in a traceability matrix without the verification and validation fields present.

choice, especially the IMU. It is also assumed that Thetis will be lost at some point during testing. This is just an inevitability during field work, so keeping the cost down would also reduce the impact of such a loss on a project's budget.

3.5.1 System Requirements

To guide Thetis' design, a series of system requirements were generated that would need to be met in order to declare the design as “delivered”. These requirements were derived from the House of Quality (Table 3.12), the stakeholder requirements (Section 3.2.3), and the derived capabilities (Tables 3.13 through 3.15). Each requirement is categorized into subsections and given a numerical designation. The requirement is then given as a clear statement of fact with only one consideration given per requirement [38]. A comment may be associated with each requirement that provides additional context or relevant information. These requirements are arranged within a traceability matrix, as shown in the following table.

Section	Design Considerations/Requirement	Capability	Comments	Verified?	Verification Method	Associated Test Number	Verification Comments	Validated?	Status
1	System								
1.1	Operation								
1.1.1	The system shall record inertial measurement data at least 64 Hz	104	Expected motion characteristic to be about 16 Hz; to gather accurate sinusoidal image requires at least 4x signal rate	Y	Demonstration	AT/GT/MT	2023-02-11: The current version of the software samples at the maximum possible rate (90-100 Hz). A future update will enable more uniform update rates	N	TIP
1.1.2	If GPS is present, the system shall record position, velocity, and course data at least 1 Hz, when fix is valid	201		Y	Demonstration	GPST		N	TIP
1.1.3	The system shall record all data onto onboard storage of at least 1 GB	105		Y	Inspection			Y	D
1.1.4	Data contained within the onboard storage shall be able to be retrieved either physically, over the air, or through a serial connection	111		Y	Inspection	Any	2023-02-11: As of writing, data has been retrieved physically (by removing the uSD card) and OTA. Serial not tested yet.	N	TIP
1.1.5	The system shall fit within a water-tight enclosure	101		Y	Inspection			Y	D
1.1.6	The system shall be capable of being mounted to a surface with bolts, screws, or zipties			Y	Inspection			Y	D
1.1.7	The system shall be operable by a single person			Y	Demonstration			Y	D
1.1.8	The system shall be able to load user configurations from a file in the onboard storage unit when booted	204		Y	Test	Any		Y	D
1.1.9	The user shall be able to overwrite the user configurations through the serial console or over the air	204		Y	Test	Any		Y	TIP
1.1.10	The user shall be able to turn on and off the system using a physical switch	107	Power supply for the regulator should automatically be chosen when the switch is flipped. USB power has priority over the battery.	Y	Test	Any		Y	D
1.1.11	The system shall be able to power down peripherals (sensors, indicators, radios, etc.) when entering a sleep mode	304	This can be done either by setting each chip individually to their sleep or low power states, or by turning off a dedicated peripheral voltage regulator	N	Test		We can measure the power draw by supplying 5V through the test pads and monitoring current draw in the different modes	N	R
1.1.12	When plugged into a USB data cable the system shall enter a diagnostic mode		Need a sense line on the V_USB bus to detect USB is present	Y	Test	Any		Y	D
1.1.13	When plugged into a USB data cable, the system shall not start the main firmware loop until a diagnostic serial terminal is opened			Y	Test	Any		Y	D
1.1.14	The system shall have an RGB LED that can be used to provide operational feedback to the user without a diagnostic console	107	Different color codes for system states - error codes for failure modes	Y	Test	Any		Y	D
1.2	Sensors								
1.2.1	The system shall utilize an IMU with at least 6 DOF	104		Y	Inspection	AT/GT/MT		Y	D
1.2.2	If the IMU has only a gyroscope and accelerometer, the system shall integrate another sensor to determine heading	104	In GPS-enabled environments, the GPS sensor will suffice. Otherwise, a 3DOF magnetometer will need to be added	N	Inspection	MT		N	R
1.2.3	Accelerometer shall be capable of measuring accelerations up to $\pm 24g$	104		Y	Inspection			N	A

1.2.4	Accelerometer shall have a sensor resolution of at least 12 bits at $\pm 8g$ sample range	104		Y	Inspection			N	A
1.2.5	Gyroscope shall be capable of measuring rotation rates of up to 2000 deg/sec	104		Y	Inspection			N	A
1.2.6	Gyroscope shall have a sensor resolution of at least 12 bits at ± 500 deg/sec sample range	104		Y	Inspection			N	A
1.2.7	Magnetometer shall be capable of measuring magnetic fields up to $\pm 8G$	104		Y	Inspection			N	A
1.2.8	Magnetometer shall have a sensor resolution of at least 12 bits at $\pm 2G$ sample range	104		Y	Inspection			N	A
1.2.9	When in realistic conditions, the GPS shall report position data with an accuracy of at least $\pm 3m$	201		Y	Analysis			N	A
1.2.10	The GPS shall be capable of reporting NMEA-encoded data at least every 1 Hz	201		Y	Demonstration	GPST		N	A
1.2.11	The polling rate of the sensors shall be configurable by the user through the configuration file and must conform to the sensor's options	204		Y	Demonstration	AT/GT/MT		N	A
1.2.12	The sample rate of the sensors shall be configurable by the users through the configuration file	204	1, 2, 4, 8, 16, 32, 64 Hz options	Y	Demonstration	AT/GT/MT		N	A
1.3 Units									
1.3.1	Unless otherwise specified, internal processing units shall be metric (SI)			Y	Inspection			Y	D
1.3.2	Unless otherwise specified, measurements shall be reported in metric (SI) units			Y	Inspection			Y	D
1.3.3	Unless otherwise specified, internal timing shall be done with microsecond-precision		Arduino: micros()	Y	Inspection			Y	D
1.3.4	Unless otherwise specified, internal timestamping shall be done with POSIX epoch		Seconds since January 1, 1970 @ 00:00 UTC	Y	Inspection			Y	D
1.3.5	Unless otherwise specified, reported timestamps shall be in ISO8601 format with millisecond precision		YYYY-MM-DDTHH:mm:SS.sss	Y	Inspection			Y	D
1.3.7	Unless otherwise specified, reported timestamps shall be in the Universal Time Coordinated			Y	Inspection			Y	D
1.3.8	The user shall be able to override the timestamp format using the configuration file	204		N	Demonstration			N	UR
1.3.9	The user shall be able to override the recorded timezone using the configuration file	204		N	Demonstration			N	UR
1.3.10	Whenever possible the internal RTC shall be synchronized with external time sources like NTP and/or GPS	201		Y				N	TIP
1.3.11	In the absence of a synchronization source like NTP or GPS, the device shall report time in terms of time since power on			Y	Demonstration			Y	D
2 Mechanical									
2.1 Physical dimensions									
2.1.1	The system shall fit within a 8" x 5" x 1.25" space	102		Y	Inspection			Y	D
2.1.2	The system shall not weight more than 500 grams	102		Y	Inspection			Y	D
2.2 Enclosure									
2.2.1	The enclosure shall be rated to withstand at least submersion in 1 meter of water for up to 4 hours	101		Y	Test			Y	A
2.2.2	The enclosure shall not allow any dust to enter it	101		Y	Test			Y	A
2.2.3	The enclosure shall be sealed using a replaceable gasket or o-ring	101		Y	Inspection			Y	D
2.2.4	The enclosure shall not exceed the physical dimensions specified in Requirement 2.1.1	101		Y	Inspection			Y	D
2.2.5	The enclosure shall be able to be bolted, screwed, or zip tied to a surface with at least 2 points of contact			Y	Inspection			Y	D
2.2.6	The enclosure shall have external markings indicating the system's measurement axes and sensor location			Y	Inspection			N	A

2.2.7	The enclosure shall be made of a non-RF blocking material, unless an external antenna is available	205		Y	Inspection			N	A
2.2.8	The enclosure shall be made of a material resistant to continuous submersion in salt water (>25 ppt NaCl)	101		Y	Test			N	A
2.2.9	The enclosure shall be made of a material that can withstand constant exposure to sunlight (UV radiation)	101		Y	Test			N	A
2.2.10	The enclosure shall be capable of withstanding multiple drops without compromising its integrity	101		Y	Test			N	A
2.2.11	The enclosure shall have multiple points on which to mount the instrumentation board	101		Y	Inspection			Y	D
3 Electrical									
3.1 Power									
3.1.1	The system shall operate off a 1S (3.7V nom.) lithium polymer battery			Y	Inspection			Y	D
3.1.2	The system shall use appropriate onboard voltage busses, as necessary			Y	Inspection			Y	D
3.1.3	The system shall not exceed the current draw of the battery			Y	Test	PWT		N	A
3.1.4	The system shall be optionally powered from a USB or other external source			Y	Demonstration			Y	D
3.1.5	In accordance with Requirement 3.1.4, the system will not allow current to flow unregulated from the external source to the battery			Y	Analysis			N	A
3.1.6	The system shall not have multiple power sources being used at once			Y	Analysis	PWT		N	A
3.1.7	In accordance with Requirement 3.1.6, the system shall draw power from the external voltage source, before the battery			Y	Analysis	PWT		N	A
3.1.8	The system shall use low quiescent-current regulators, where feasible			Y	Analysis	PWT		N	A
3.1.9	The system shall provide a battery backup voltage to the GPS module, if supported			Y	Inspection			N	A
3.1.10	The system shall provide a battery backup voltage to the RTC, if supported			N	Inspection			N	R
3.1.11	The system shall be able to recharge the battery when plugged into USB power			Y	Demonstration			Y	D
3.2 Mechanical Connections									
3.2.1	Where possible, the system shall be assembled using lead-free solder that passes ASTM standards	112		Y	Inspection			N	A
3.2.2	Components shall be soldered to the PCB following IPC J-STD-001 standards for electrical soldering	112	More information: https://www.protoexpress.com/blog/IPC-J-STD-001-standard-soldering-requirements/	Y	Inspection			N	A
3.2.3	When possible, components shall be placed on a single side of the PCB	112		Y				Y	D
3.2.4	Any PCB designs will be made in accordance with the IPC-2221B standard	112	More information: https://www.protoexpress.com/blog/IPC-2221B-circuit-board-design/	Y	Inspection			N	A
3.2.5	Any board-to-board or board-to-cable connections shall use keyed receptacles that prevent connector reversal	112		Y	Inspection			N	A
3.2.6	Any board-to-board or board-to-cable connections shall use components that are rated for automotive use	112		Y	Inspection			N	A

3.2.7	Any board-to-antenna connectors shall use locking, friction-fit connectors with appropriate strain relief	112	This is meant to ensure that the antenna connectors do not dislodge during use	Y	Inspection			N	A
-------	---	-----	--	---	------------	--	--	---	---

3.5.2 Failure Mode, Effect, and Criticality Analysis

Since Thetis is intended to be used by end users who may not have a solid background in programming or electrical skills, it is important to identify potential flaws in the design and attempt to mitigate them. Each potential flaw is given a probability of occurrence and a weight from 0 to 10 inclusive for its severity. By multiplying these two values together, a Risk Priority Number (RPN) is created that highlights the severity of a potential failure mode. The RPNs are categorized into three groupings:

LOW These are RPNs with a value below 2.5 and represent little risk to the system. These are denoted with green highlights in the FMECA chart.

MEDIUM These are RPN's between 2.5 and 7.5 that are represented with yellow highlights. Typically, these failure modes represent a noticeable threat to the system and the mitigations should be carefully implemented.

HIGH These are the highest risk RPNs between 7.5 and 10 and are represented with red highlights. They are the most severe and should be mitigated at all cost during and after the product development cycle.

The probabilities are generally conjectured from past experience, the failure ratings of parts, and other factors that are carefully mapped out and documented over time. However, since Thetis is using hobby-grade materials and failures are not generally well-documented for the parts, the probabilities represented herein are my best guess based on experience. The same applies for the criticality score for each potential failure.

Reference Number	Related Requirement	Failure Mode	Type of Failure	Cause of Failure	Effect of Failure	Before			Mitigation of Failure	After		
						Likelihood	Criticality	RPN		Likelihood	Criticality	RPN
1a	1.1.3	uSD card not present	Operator Error	uSD card is not inserted into the receptacle	System will be unable to log data to a uSD card for storage and off-loading	0.50	9	4.50	Go into an error state when uSD is not detected. Prompt user to override by storing data on backup flash storage - notify them that doing so will dramatically limit their deployment time	0.50	7	3.50
1b	1.1.3	uSD card is broken	Hardware Failure	The uSD card is damaged and inoperable	System will be unable to log data to a uSD card for storage and off-loading.	0.10	9	0.90	Go into an error state when uSD is not detected. Prompt user to override by storing data on backup flash storage - notify them that doing so will dramatically limit their deployment time.	0.10	9	0.90
					Depending on the physical damage to the card, an electrical short may develop, leading to excess heating and power draw	0.10	9	0.90	Protect the SD card with a strong receptacle. Have fuses that can limit the current drawn in the event of a short circuit	0.05	9	0.45
2	1.1.10	The power switch is non-operable	Hardware Failure	The switch is melted and locked into a certain setting. Debris or corrosion jammed the switch into its current settings	The system will be unable to turn on	0.01	10	0.10	Ensure that components are protected during use by strong case.	0.01	10	0.10
					The system will be unable to turn off	0.01	1	0.01	Ensure that components are able to withstand intended soldering conditions (especially temperature)	0.01	1	0.01
3	3.1.11	The battery does not charge when plugged into USB power	Hardware Failure	Internal failure of the IC. Faulty connections	The system will be unable to charge the battery by itself	0.10	1	0.10	Ensure that the battery is easily removable and an external charger is available for it	0.10	1	0.10
					Physical damage to the charging IC	0.01	1	0.01	Have fuses on the board that can limit the current drawn in the event of a short circuit. Protect the IC from physical damage with a case	0.01	1	0.01
4a	1.1.2	The GPS does not receive a valid satellite fix	Radio Failure	The radio antenna does not have a clear LOS to the sky	The system will be unable to locate itself in geographic coordinates	0.90	5	4.50	Ensure that the GPS radio antenna has a clear line of sight to the sky. Ensure that antenna is clear of concrete or metal obstructions.	0.01	5	0.05
					The system will not be able to coordinate the system time to UTC reported by satellites	0.90	2	1.80	Allow the user to overwrite the current RTC setting via configuration settings.	0.90	1	0.90
4b	1.1.2	The GPS does not send or receive data to the microcontroller	Software Failure	The GPS firmware has crashed due to a buffer overflow error	The system will not be able locate itself in geographic coordinates	0.40	5	2.00	Integrate a software reset function for the GPS radio Adjust the GPS report rate and baud rate to be more stable	0.40	2	0.80
					The system will not be able to coordinate the system time to UTC reported by satellites	0.40	2	0.80	Allow the user to overwrite the current RTC setting via configuration settings.	0.40	1	0.40

5a	3.2.7	The microcontroller does not broadcast a WiFi Access Point station	Radio Failure	The radio antenna is not properly connected to the surface mount connector	The system will not be able to connect to outside devices and users will not be able to configure the device remotely	0.60	5	3.00	Ensure that the antenna connectors are securely mounted to any board connectors. Minimize cycle times of plugging/unplugging the connectors to maximize usable lifespan	0.20	5	1.00
	2.2.7			The radio antenna is mounted inside a RF-blocking material		0.10	5	0.50	Ensure that the enclosure is non-RF-blocking, or the antenna is mounted externally	0.10	5	0.50
			Hardware Failure	The WiFi PHY hardware inside the microcontroller fails to initialize		0.05	5	0.25	Ensure that the microcontroller is not damaged and does not exceed its maximum operating limits for power, temperature, humidity, etc.	0.05	5	0.25
			Software Failure	The WiFi PHY firmware inside the microcontroller crashes		0.05	5	0.25	Ensure that the microcontroller code does not crash due to run-time errors and the web server cannot be disrupted easily. Implement a software check that resets the system after a hanging failure.	0.01	5	0.05
				The WiFi AP is not initialized with the correct settings		0.30	5	1.50	Ensure that the configuration file is correct. Allow users to change settings in-situ	0.10	5	0.50
5b	3.2.7	The microcontroller does not connect to a WiFi network	Radio Failure	The radio antenna is not properly connected to the surface mount connector	The system will not be able to connect to outside devices and users will not be able to configure the device remotely	0.60	5	3.00	Ensure that the antenna connectors are securely mounted to any board connectors. Minimize cycle times of plugging/unplugging the connectors to maximize usable lifespan	0.20	5	1.00
	2.2.7			The radio antenna is mounted inside a RF-blocking material		0.10	5	0.50	Ensure that the enclosure is non-RF-blocking, or the antenna is mounted externally	0.10	5	0.50
			Hardware Failure	The WiFi PHY hardware inside the microcontroller fails to initialize		0.05	5	0.25	Ensure that the microcontroller is not damaged and does not exceed its maximum operating limits for power, temperature, humidity, etc.	0.05	5	0.25
			Software Failure	The WiFi PHY firmware inside the microcontroller crashes		0.05	5	0.25	Ensure that the microcontroller code does not crash due to run-time errors and the web server cannot be disrupted easily. Implement a software check that resets the system after a hanging failure.	0.01	5	0.05

				The WiFi connection is not initialized with the correct settings	0.30	5	1.50	Ensure that the configuration file is correct. Allow users to change settings in-situ	0.10	5	0.50			
6	3.2.2	The I2C devices are not initialized properly	Hardware Failure	A solder joint is not connected properly to one or multiple devices	0.05	10	0.50	Ensure that the device is properly assembled and tested before deployment	0.05	10	0.50			
				One or more devices has physically broken	0.01	10	0.10	Ensure that the device is properly protected by a drop-resistant case Ensure that the device is stored in accordance with manufacturer recommendations	0.01	10	0.10			
			Software Failure	The I2C bus is not properly initialized on the device	0.30	10	3.00	Ensure that the SDA and SCL pins are properly set in the board configuration file. Ensure that "Wire.begin((int) SDA, (int) SCL);" is called before I2C sensor initialization begins	0.01	10	0.10			
				One or more devices have reached a soft failure state	0.05	10	0.50	Have the microcontroller be able to soft reset each device Notify the user of failure modes and advise them to hard restart the device	0.05	3	0.15			
	7	1.1.12	The device is not put into a diagnostic mode when a USB cable is inserted	Hardware Failure	One or more of the resistors in the sense divider are not the correct value	0.01	3	0.03	Ensure that the board is assembled with the appropriate parts and that the connections are secure	0.01	3	0.03		
	8a	1.1.14	The NeoPixel LED does not come on during system start	Hardware Failure	The MOSFET enabling current flow to the component may not be open	0.20	1	0.20	Check that the NEOPixel_EN line is LOW (0V)	0.20	1	0.20		
		3.2.2			A solder joint is not connected properly to the NeoPixel or related components	0.10	1	0.10	Ensure that the device is properly assembled and tested before deployment	0.05	1	0.05		
					One or multiple LEDs within the NeoPixel are burned out	0.30	1	0.30	Ensure that the current-limiting resistor to the NeoPixel is greater than to equal to the value specified in the schematic	0.05	1	0.05		
	8b	1.1.14	The NeoPixel LED does not display the proper colors	Hardware Failure	One or multiple LEDs within the NeoPixel are burned out	0.30	1	0.30	Ensure that the current-limiting resistor to the NeoPixel is greater than to equal to the value specified in the schematic	0.05	1	0.05		
	The color values provided to the NeoPixel color() function are not in GRB format	0.05	1		0.05	Ensure that the colors specified in the firmware are in the proper color format	0.01	1	0.01					

9	1.1.3	The diagnostic file logger does not initialize properly	Operator Error	uSD card is not inserted into the receptacle	System will be unable to log data to a uSD card for storage and off-loading	0.50	9	4.50	Go into an error state when uSD is not detected. Prompt user to override by storing data on backup flash storage - notify them that doing so will dramatically limit their deployment time	0.50	7	3.50
			Software Failure	The CS line is not properly selected in the firmware		0.01	9	0.09	Ensure that the SD_CS variable in the firmware is properly configured to the schematic.	0.01	7	0.07
10	1.1.8	Secondary flash storage (SPIFFS) is not initialized	Hardware Failure	The SPI bus is having electrical issues, or CS line is not set correctly. The microcontroller may be physically damaged	System will be unable to load settings from the on-board configuration file.	0.10	9	0.90	Warn user of a fault with secondary flash storage; enter failsafe state	0.10	9	0.90
					System will be unable to load webpage information for serving the dashboard and UI	0.10	1	0.10		0.10	1	0.10
			Software Failure	The SPIFFS memory registers may not be initialized	System will be unable to load settings from the on-board configuration file.	0.10	9	0.90	Warn user of a fault with secondary flash storage; enter failsafe state	0.10	9	0.90
					System will be unable to load webpage information for serving the dashboard and UI	0.10	1	0.10		0.10	1	0.10
12a	1.1.8	The configuration file cannot be loaded from secondary storage	Hardware Failure	The SPI bus is having electrical issues, or CS line is not set correctly. The microcontroller may be physically damaged	System will be unable to load settings from the on-board configuration file.	0.10	9	0.90	Warn user of a fault with secondary flash storage; enter failsafe state	0.10	5	0.50
						0.10	9	0.90		0.10	5	0.50
			Software Failure	The configuration file may not be loaded into the SPIFFS memory registers	The user configuration settings will be unable to be loaded	0.60	9	5.40	Ensure that the filesystem image is flashed to the device before deployment	0.50	9	4.50
						0.10	9	0.90		0.01	9	0.09
13	1.3.10	The microcontroller's internal RTC is not able to synchronize properly	Radio Failure	The GPS does not have a valid fix to the satellite constellation	The on-board report time will not be synchronized to UTC, rather it will be in terms of time since power-on	0.80	2	1.60	Attempt to synchronize the RTC every minute Advise the user of the synchronization status by using the diagnostic logs	0.80	1	0.80
			Software Failure	The RTC synchronizes to the GPS time, but incorrectly counts the milliseconds between updates	The on-board report time will not be correct. For instance, the time may be 2023-02-23T09:00:00.120000 where the clock is misreporting the 2 minutes that have passed.	0.90	9	8.10	Regulararly update the RTC milliseconds in the main loop execution	0.01	9	0.09

14	1.1.3	The data logger does not initialize	Operator Error	uSD card is not inserted into the receptacle	System will be unable to log data to a uSD card for storage and off-loading	0.50	9	4.50	Warn user of a fault with secondary flash storage; enter failsafe state	0.50	7	3.50
			Hardware Failure	The uSD card is damaged and inoperable	System will be unable to log data to a uSD card for storage and off-loading.	0.10	9	0.90		0.10	9	0.90
			Software Failure	The CS line is not properly selected in the firmware	Depending on the physical damage to the card, an electrical short may develop, leading to excess heating and power draw	0.10	9	0.90	Protect the SD card with a strong receptacle. Have fuses that can limit the current drawn in the event of a short circuit	0.05	9	0.45
			Software Failure	The maximum number of data logs have been created	System will be unable to log data to a uSD card for storage and off-loading	0.01	9	0.09	Ensure that the SD_CS variable in the firmware is properly configured to the schematic. Warn user of a fault with secondary flash storage; enter failsafe state	0.10	7	0.70
15	2.2	The enclosure is not securely closed	Hardware Failure	The enclosure floods when exposed to water	System will become inundated with water and cease working. Board will be destroyed by galvanic corrosion	0.33	10	3.30	Set the maximum number of log files absurdly high Well before reaching the maximum limit, warn the user about the potential failure and prompt them to clear the data	0.01	9	0.09
									Ensure all screws for enclosure are present and properly torqued Do not over torque screws Inspect enclosure to ensure no cracking evident	0.10	10	1.00

3.5.3 Functional Block Diagram

Factoring in the design principles of capability, manufacturability, and affordability, along with the stakeholder requirements listed in Section 3.2.3, a block diagram was constructed that integrated the crucial parts and drew out their interactions. This block diagram shows the data sharing relationships in blue and the power relationships in warm colors. Labels on the specific lines indicate the specific communication protocol or voltage level being used by the line. Explanations for these protocols and acronyms can be found in Appendix D.

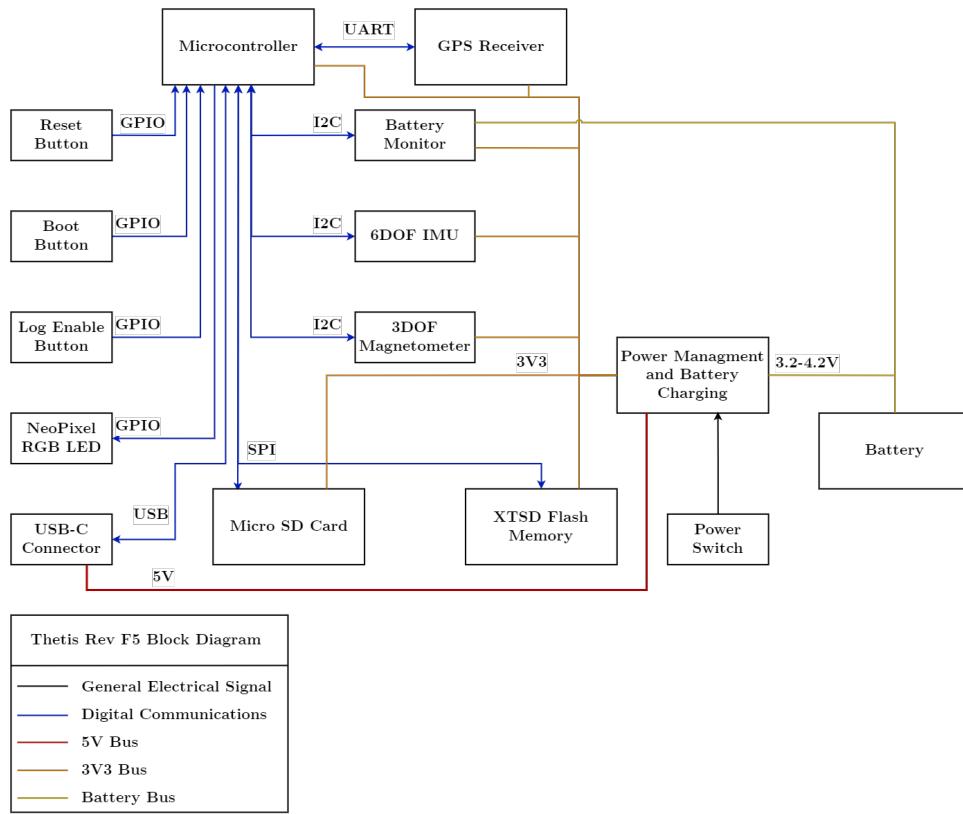


Figure 3.1: A block diagram representation of the Thetis Revision F5 instrumentation board.

3.5.4 Functional Flow Diagram

Based off the block diagram and interviews with potential end users, a notional flow diagram for the firmware was created. The code was designed to be modular and allow for new features

to be quickly added or removed with separate libraries.

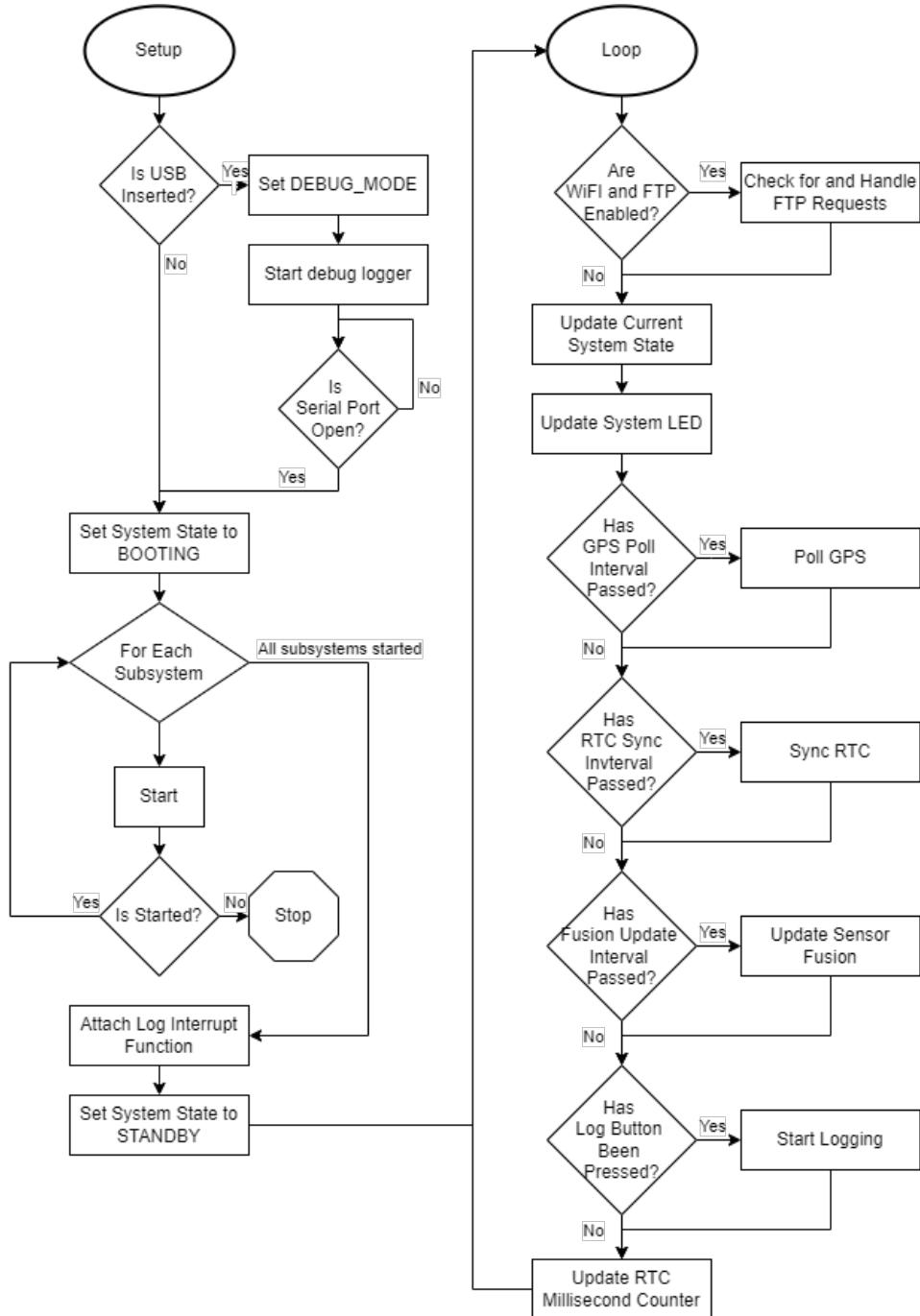


Figure 3.2: A flow diagram representation of the base firmware of the Thetis instrumentation board.

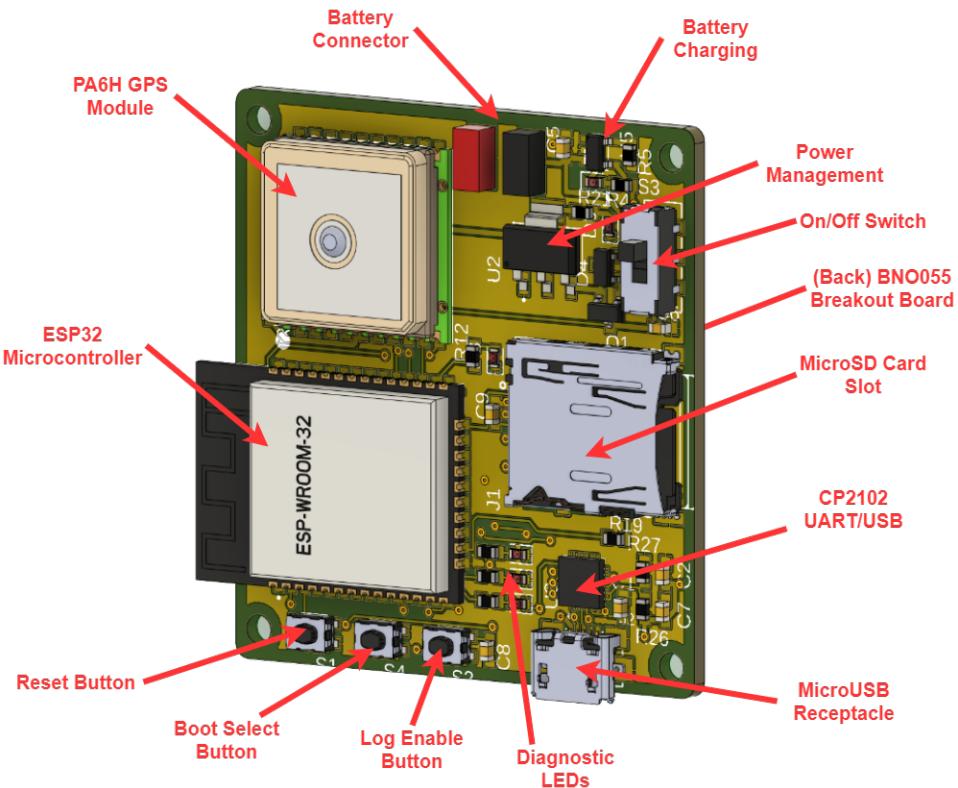
3.6 Brief Revision History

After carefully considering all the design criteria from the stakeholder requirements and system requirements, Thetis began several design revisions.

3.6.1 Revision F1

The first design revision, Revision F1, was a functional proof of concept. It incorporated many of the base design features of later Thetis revisions, but relied on a breakout board for the IMU mounted to the bottom of the board and had the battery placed on top. This forced the battery to be smaller than the desired capacity and overfilled the enclosure, causing excessive compression and stress when the enclosure was tightened and secured.

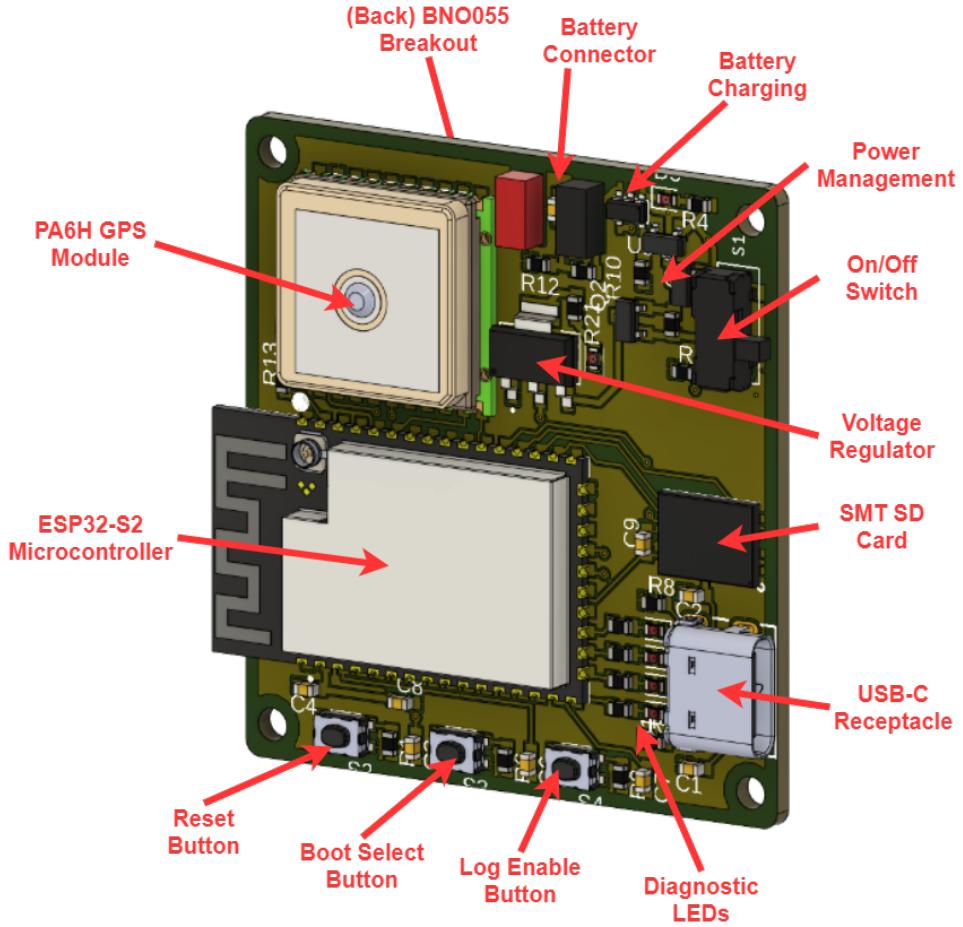
Figure 3.3: Thetis Revision F1 PCB render with call outs for important components.



3.6.2 Revision F2

The next revision, F2, changed to a new microcontroller version that simplified the circuit while providing the same capability of the previous revision. This version also switched from a micro SD card receptacle to a soldered SMD flash memory chip. This chip maintained the same capacity requirements from the stakeholders, but took up a fraction of the space on the board. However, since this chip was directly soldered to the board, the only way to offload data was through the microcontroller either through an FTP server, or through the USB interface. This was not ideal - especially in the early testing phases, as those features are more complex to implement programmatically and test data cannot be easily offloaded if there is an issue with the microcontroller. There were also reliability issues which manifest in force on later revisions. This revision also kept the breakout board for the IMU on the back, giving it the same space constraint problem as its predecessor.

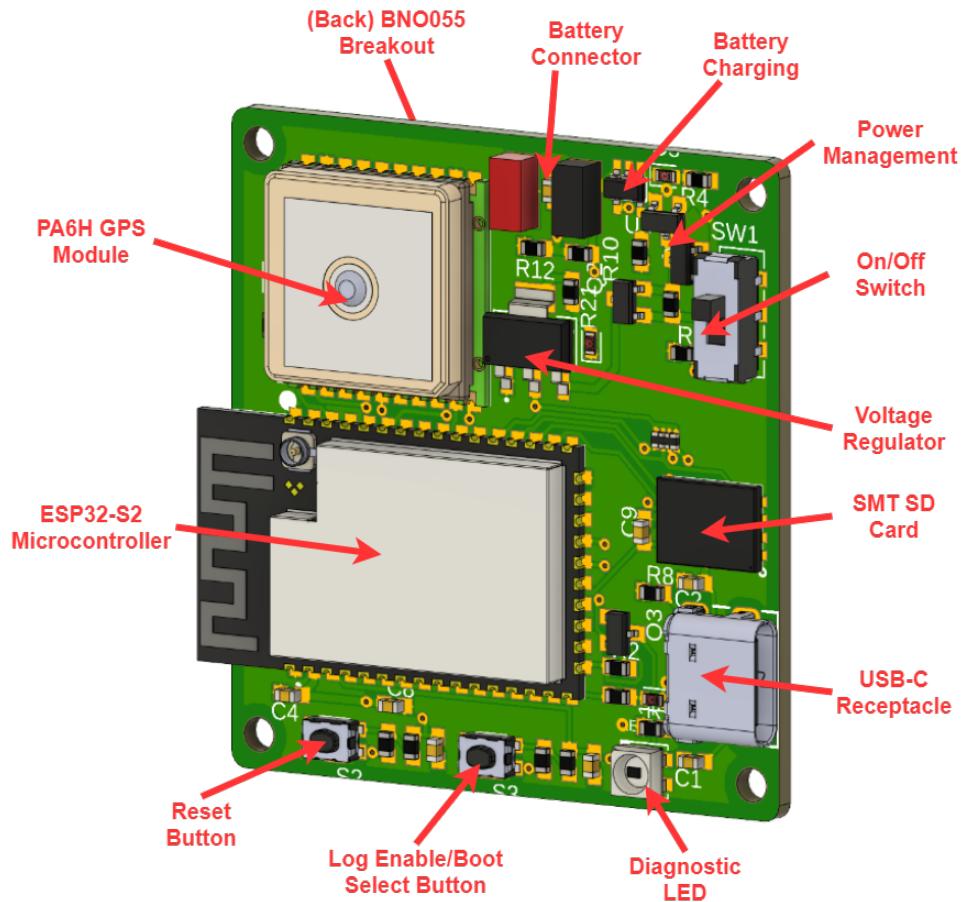
Figure 3.4: Thetis Revision F2 PCB render with call outs for important components.



3.6.3 Revision F3

For Revision F3, some minor changes were made to the diagnostic LEDs and user inputs. Several discrete LEDs for logging, error notification, and low battery, were replaced by a single addressable RGB LED that could flash different colors and patterns to convey the system state or errors. The dedicated log enable button was also removed in favor of using the boot select button for both purposes. However, this method was entirely flawed as the log enable button relies on an interrupt input to trigger the system to begin recording. On the ESP32-S2 microcontroller, an interrupt tied to GPIO 0 (the boot select pin) would cause the system to crash and force a reset. Therefore, the entire design had to be scrapped due to this design flaw.

Figure 3.5: Thetis Revision F3 PCB render with call outs for important components.



3.6.4 Revision F4

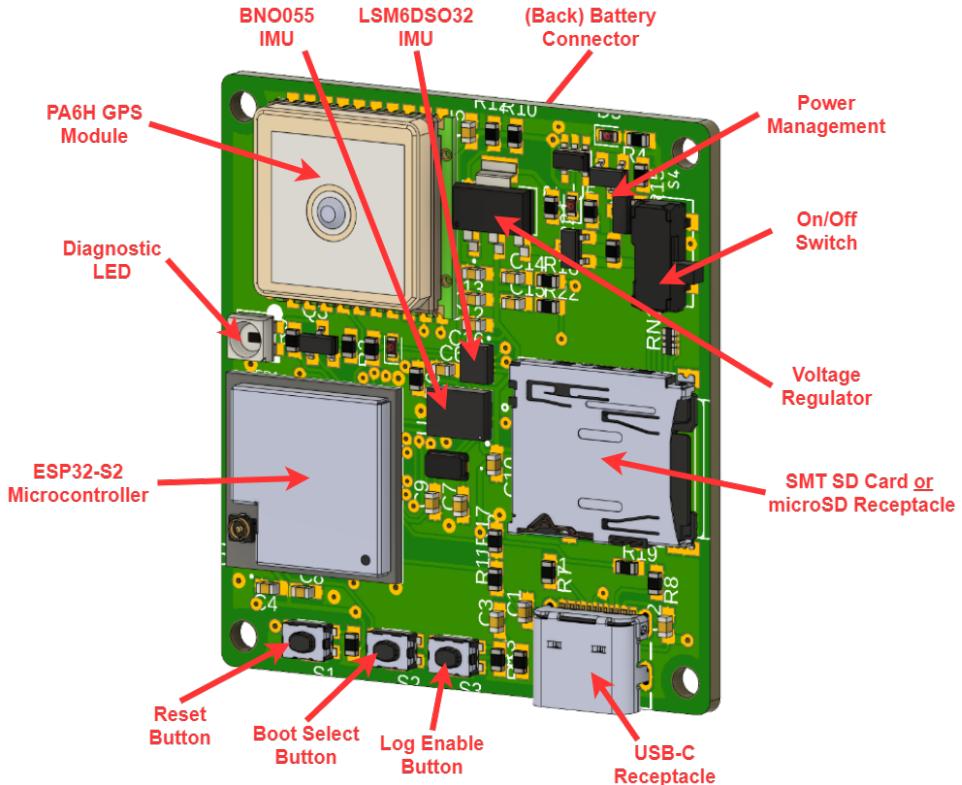
Revision F4 was the most radical design change of the series. It marked a migration to all SMD components (no breakout board for the IMU) and incorporated all of the design lessons from the previous revision. However, the IMU that was used in previous revisions, the Bosch BNO055² was amongst the many IMUs affected by the Great Chip Shortage and was therefore unavailable. So, this design had to adapt by incorporating a new, previously untested IMU, the STdevices LSM6DSO32³. This new IMU does not have a magnetometer and at the time

²<https://www.digikey.com/en/products/detail/bosch-sensortec/BN0055/6136301>

³<https://www.digikey.com/en/products/detail/stmicroelectronics/LSM6DS032TR/11694177>

of its design, they were hard to find in stock at any distributors. So, in anticipation of the Chip Shortage ending, the BNO055 was incorporated to the design, but not populated, giving this board the unique opportunity of having either or both IMUs present - a useful feature for more accurate sensor fusion, but a terrible prospect for software integration. Additionally, this design reintroduced the micro SD card receptacle, but this time, the SMD flash chip was laid out beneath it. This provided the assembler to determine which storage system they wanted: easier to use micro SD, or more resilient SMD flash. Programmatically, these devices work the same, so the software cost was minimal while providing some flexibility. However, this design was also fundamentally flawed as the newly reintroduced log enable button was not integrated into the design properly and the chip select pin for the storage device was connected to an incorrect controller pin. The latter issue forced the microcontroller to crash and reset every time it tried to write to the external flash memory, necessitating a new board revision.

Figure 3.6: Thetis Revision F4 PCB render with call outs for important components.



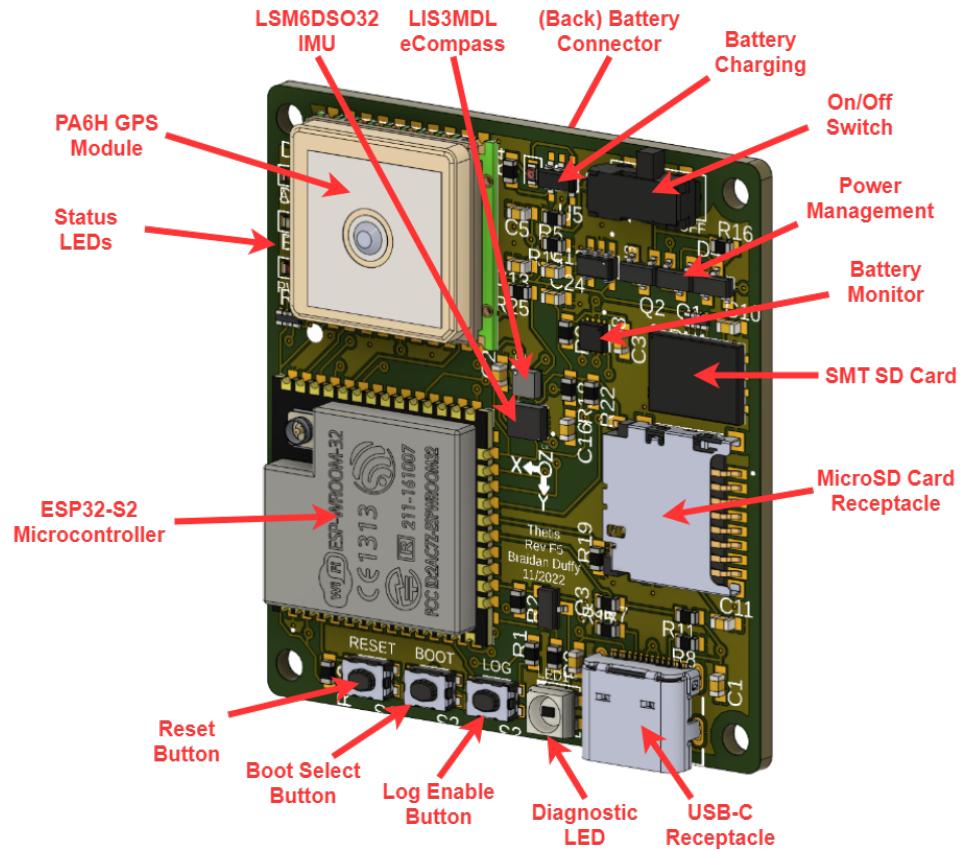
3.7 Final Design

Revision F5 is the latest version that realizes all the hardware features Thetis is meant to incorporate while being easy to assemble and use. The BNO055 was fully removed from the system in favor of adding a dedicated magnetometer, the STdevices LIS3MDL⁴. By reorganizing the layout, the SMD flash chip and micro SD card receptacle were placed side-by-side enabling both primary and secondary storage services, enabling backups and improving reliability. Then, to improve the battery monitoring capabilities, a battery gauge IC was added. The MAX17048⁵ is a monitoring IC that runs a mathematical model of the battery's discharge curve given its amp-hour capacity and current operating voltage. This data is reported to the microcontroller as the current battery percentage, or life remaining - making this a crucial part of monitoring performance and operations of Thetis when deployed. Component types and placements were also streamlined to improve manufacturability and allow the potential for the device to be more easily assembled by a pick and place machine. This also allows students who have never assembled PCBs before to more easily take up the task and build these boards themselves.

⁴<https://www.digikey.com/en/products/detail/stmicroelectronics/LIS3MDLTR/4309733>

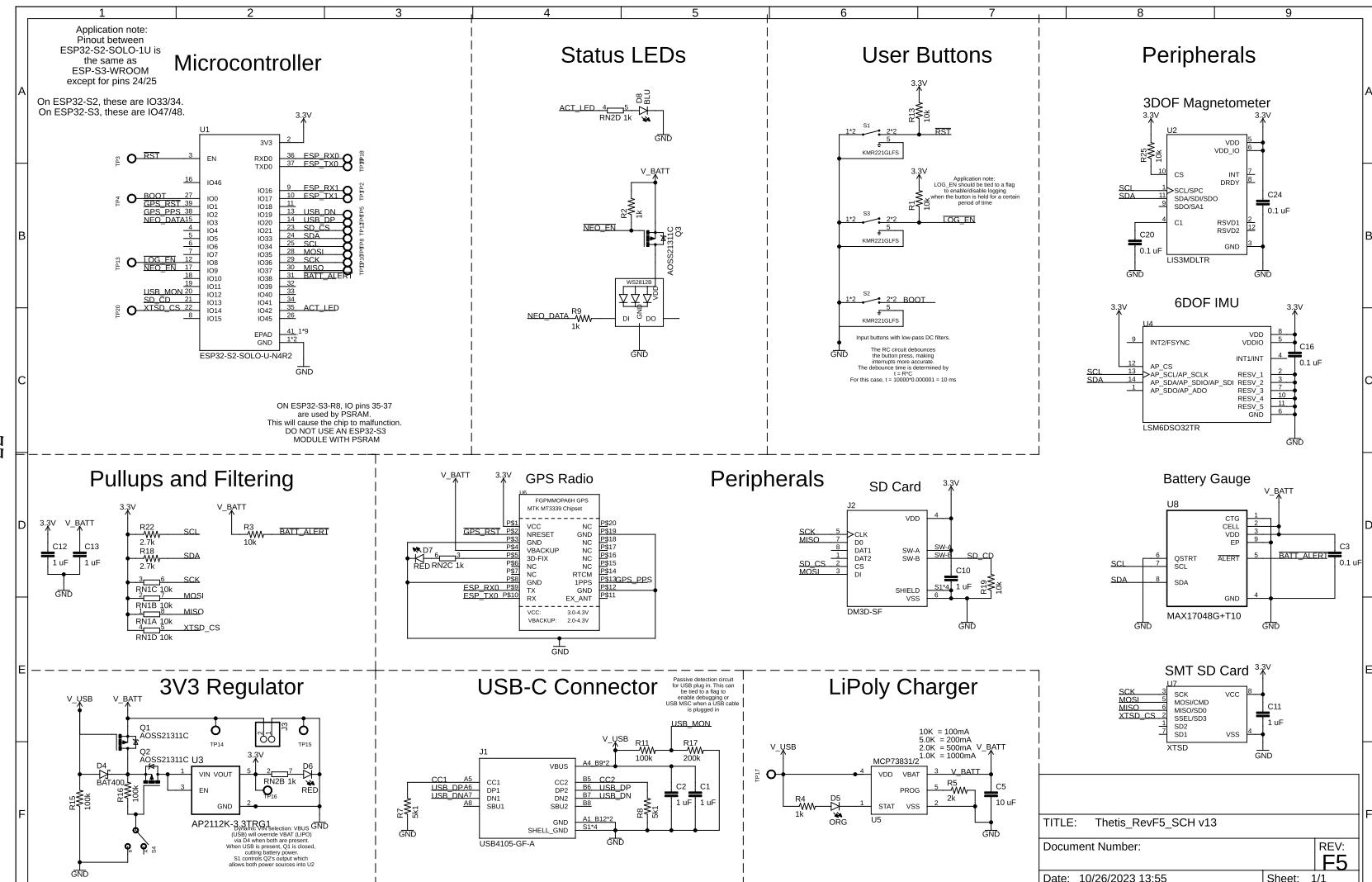
⁵<https://www.digikey.com/en/products/detail/analog-devices-inc-maxim-integrated/MAX17048G-T10/3758921>

Figure 3.7: Thetis Revision F5 PCB render with call outs for important components.



The next sections detail the assembly process and provide the schematics for Revision F5. The schematics for the previous revisions can be found in Appendix J.

3.7.1 Schematic



3.7.2 Assembly Technique

Thetis was designed with surface mount soldering as the primary assembly technique. This technique differs from traditional soldering as the components are so small that a soldering iron is more difficult to use to mount components to the board. Many of the ICs used on the final Thetis revision do not have exposed legs or pads that can be touched with an iron, so solder paste and reflow are required. In Figure 3.8, a comparison between through hole (“traditional”) and surface mount components is shown. The surface mount components are 0603 resistors, meaning they are 60 thousandths by 30 thousandths of an inch in size - about a quarter the size of a large grain of rice.

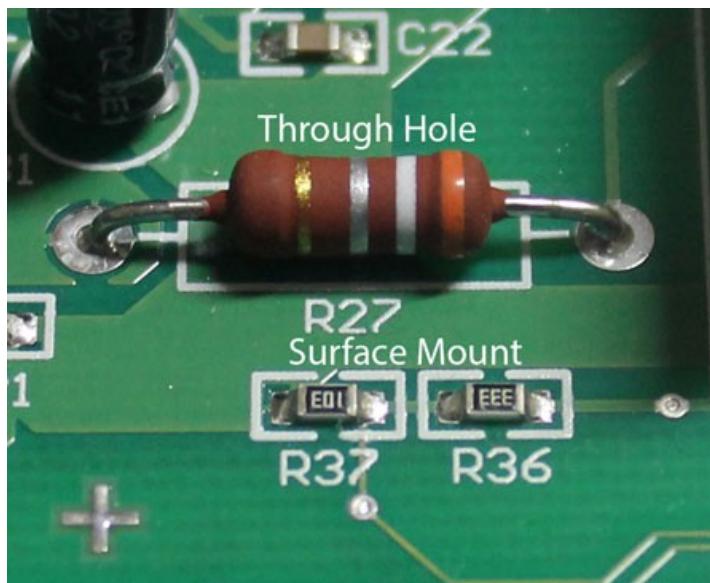


Figure 3.8: A comparison between through hole (THT) (top) and surface mount (SMT) (bottom). Retrieved from Yaman Electronics

Solder Deposition First, the bare PCB is placed beneath a stencil and secured with tape. It is important that all of the pads on the PCB are aligned with the cutouts in the stencil as the solder paste must be evenly and thoroughly distributed on the board. When the PCB is secured, a line of paste is placed on one side and smoothly dragged along the cutouts with even pressure applied throughout. When the stencil is removed, there should be a healthy deposit of solder paste on each pad as shown in of Figure 3.9c

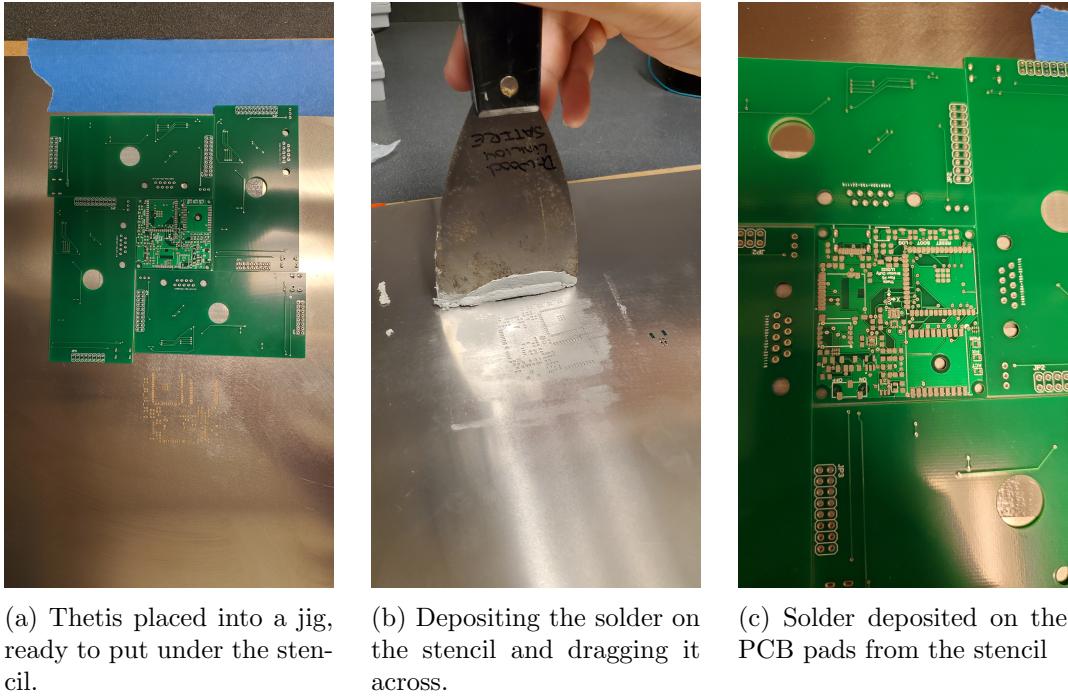
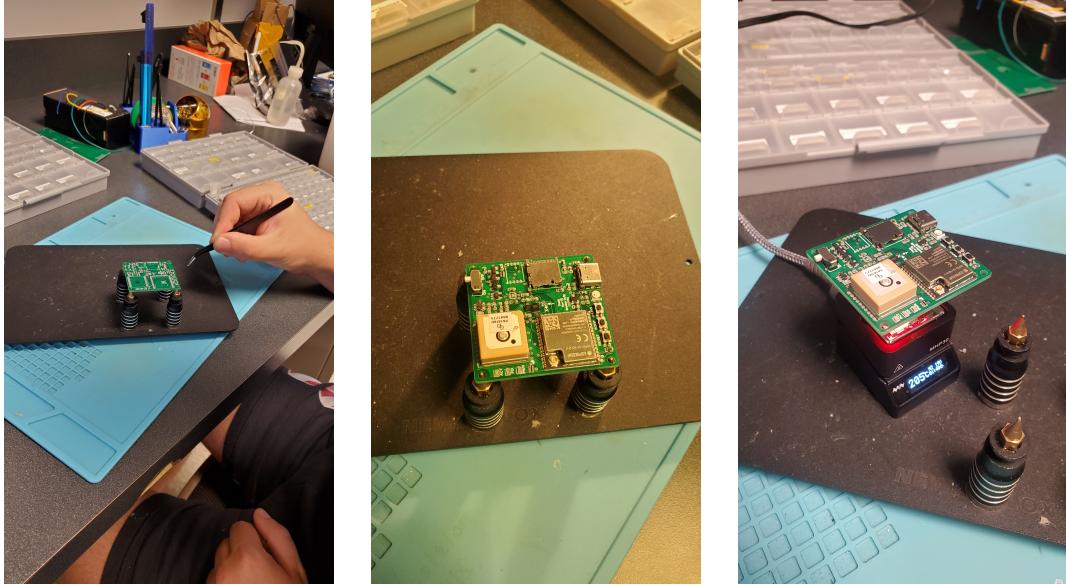


Figure 3.9: A series showing the process of putting solder paste onto a PCB from a stencil using Revision F5.

Component Placement From here, components can be placed onto the PCB carefully using tweezers. Some components measure in at tenths of an inch across and require placement tolerances of hundredths of an inch. Thankfully, in some areas, the surface tension of the solder when melting will forgive slightly misplaced components and drag them into the appropriate spots. However, it is important not to smear the solder as it may inadvertently bridge pads together and create shorts and undesired behavior. Throughout assembling these boards, this particular issue was the most common during the assembly process and occurred regularly on the USB-C receptacle, and IMU ICs where the pins are close together.

Reflow Soldering When all of the components are placed, the PCB can be placed onto a hotplate for reflow. The hotplate heats the PCB and solder up to the melting point (approximately 250°C) where the flux in the paste evaporates and the metallic solder is left behind. Solder paste typically has a reflow profile provided by the manufacturer to ensure optimal mechanical strength and solder characteristics, but this is ignore for simplicity. For

a more precise reflow, an infrared or convection oven can be used where the optimal reflow profile is programmed into the machine and the board and components are more evenly and thoroughly heated. The entire process from putting the board beneath the stencil to assembled and ready for inspection can take anywhere from 3 hours for a novice assembler, to 45 minutes for a more experienced one.



(a) Components being placed from inventory boxes onto the boards manually using tweezers
 (b) All in-stock components placed on the PCB.
 (c) PCB on a mini hotplate for solder reflow.

Figure 3.10: A series showing the process of placing components onto a Revision F5 PCB from an inventory and using a hotplate for solder reflow.

Inspection Following a successful reflow, the board must be inspected for defects. The most common are solder bridges between pads that create undesired shorts and can damage parts. The inspection will also check that parts are in the correct locations and orientations and that no shorts exist between power rails and ground. A microscope and good multimeter are crucial to this process.

If a soldering defect is discovered, the board can be placed onto a small heating plate to melt the solder. Then, the part can be removed and the pads on both the component and board are cleaned up using a soldering iron to ensure the proper amount of solder is present.

Once the pads are cleaned, either hot air or a hot plate can be used to heat the afflicted area and remelt the solder so the part can be replaced on the board. The board is inspected again and if the issue is cleared, the board is ready for testing.



Figure 3.11: The Thetis Revision F5 PCB that was assembled is placed beneath a microscope for parts inspection.

Testing When the board passes a visual inspection and there are no power shorts present, the board is plugged into a host computer via the USB port. The microcontroller is placed into the bootloader mode by holding the boot button and pressing the reset button. This should cause the microcontroller to show up as a USB device on the computer. From here, the latest Thetis firmware can be flashed to the new board. By default, when plugged into USB, the board enters a diagnostic testing mode that prints out the boot and initialization status. By opening a serial console, the assembler can see this process in real time. If there are any errors while initializing, the firmware will report which device is having a problem. The assembler will then need to do a deeper inspection to find the cause of the error and rectify it.



Figure 3.12: The Thetis Revision F5 PCB that was inspected and cleaned up is connected to a host computer where the firmware is loaded onto it.

Once the new board boots without issue, the board is ready for deployment and can be used by operators and intended end users.

3.8 Prototype Problems

Revision F5 was the most extensively tested revision of the Thetis instrumentation package. It has gone on several deployments to beaches inside surfboards and down to 20-feet of water on an ROV. Unfortunately, there was a serious flaw that plagued Revision F5 - it would not reliably or consistently record data.

From testing with Revision F3, I initially thought that the SPI communication bus was faulty. The XTSD storage chip recommends having pullup resistors on the bus to improve signal integrity. I checked all the resistors placed on those lines and they followed the recommended specifications. I moved onto to other theories, not knowing how close I originally was to the source of the problem.

For months I ran down everything I could think of within the software and hardware. I hardwired the chip select pin to always be active; I increased the memory buffer within the SD card library to see if it was a memory overflow problem; I tried different SD card libraries and even tried dumping the data to the serial monitor and logging that way. No matter what I tried, the system worked fine, reported no errors, never detected a single fault, but would not log to the microSD card.

After six months, I finally looked back at the design changes between Revisions F4 and F5. Though I did not experiment with Revision F4 extensively, the few tests I had done with it were flawless, so what was different? Revision F5 introduced a single change to the SPI bus where the XTSD flash storage chip was in parallel to the microSD card, rather than one or the other. When I removed the XTSD flash chip, logging to the microSD card worked flawlessly again!

I surmise that I was not initializing the two SD storage devices properly and there was cross-talk or corruption in the SPI bus communications that prevented data from being written to the microSD card. This corruption was not consistent, so I could not consistently recreate the error which hindered the debugging process. Additionally, this type of error is not well documented because two SD cards are not typically used in parallel, especially in any documented projects that I could find. This simple mistake held Thetis' development back by over six months and could have been avoided if I had checked the difference between the two revisions first.

The lesson learned from this exercise: if something works and you change it and it stops working, check those changes and eliminate each one as the problem until you find what went wrong. It works in software and hardware and can save months of work and heartache.

Chapter 4

Calibration

Now that Thetis is designed, we have to calibrate the instruments on board. We will follow the procedures used in Madgwick [32] and the mathematical methods discussed in Chapter 2. Before calibration began, it was necessary to design a couple of frames and machines to assist with the process.

Since the three sensing axes follow Cartesian standard and are mutually orthogonal, a calibration cube is needed to align a single axis to the measurement apparatus. The other two axes will then be planar, thus mitigating their influence on the data. The cube is also designed to hold an x-IMU3 [46] next to Thetis, allowing a direct comparison to be made while collecting data - this is discussed further within this section.

To test the accelerometer and orientation data, a 72-tooth gear and socket was cut out of spare wood using a laser cutter. With 72 teeth, the gear has a rotational resolution of 5-degrees per tooth with high precision as the laser cut did not leave any room for play between the gear and socket.

Finally, to test the gyroscope, a machine was designed and created that could rotate the calibration cube at a constant specified rate. It is built around a Raspberry Pi running

Robot Operating System 2 [30] using a stepper motor driver HAT¹. The HAT is connected to a stepper motor which is coupled to a plate mounted on a lazy susan bearing². Commands to the Pi allow it to start the motor, begin rotating, and then stop rotating. Some other features are implemented, but not used for calibration.

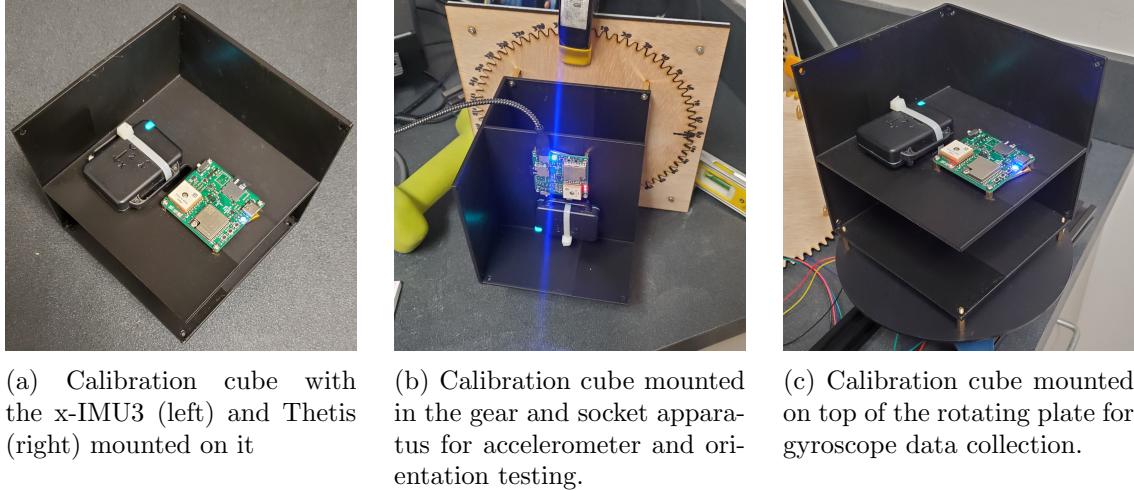


Figure 4.1: The different tools designed and used for calibration Thetis

4.1 Methodologies

The calibration process is important to improve the accuracy of the sensor before post processing and analysis are performed. It is crucial that the calibration parameter settings on the device are at their default values before beginning. Each of the starting misalignment (M_0), soft iron (W_0^{-1}), and sensitivity (S_0) matrices should be 3×3 identity matrices and the starting bias (\mathbf{b}_0) and hard iron vectors (\mathbf{v}_0) should be zero vectors.

¹<https://www.waveshare.com/stepper-motor-hat.htm>

²<https://www.tindie.com/products/fluxgarage/turtable-for-stepper-motor-kit/>

$$M_0 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$W_0^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$S_0 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{b}_0 = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}$$

$$\mathbf{v}_0 = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}$$

4.1.1 Magnetometer

To calibrate the magnetometer, it needs to be rotated about all three axes in a “figure-8” motion. This can be best accomplished by rotating about a point while making the figure-8 pattern. This ensures that the measurements in the x-, y-, and z-axis are evenly distributed throughout the magnetic field. Without any distortions, this will create a sphere with a radius of the magnitude of the magnetic field. However, as discussed in Chapter 2 and evidenced in Section 4.2, this is not always the case. To improve accuracy, the entire apparatus should be far away from any magnetic influences such as electrical fields and permanent magnets.

4.1.2 Gyroscope

The gyroscope is calibrated using the calibration machine discussed more in Appendix ???. To begin, the machine must be booted and the measurement devices turned on. Then, two terminal windows can be opened on the machine’s Raspberry Pi through SSH or a headed setup. On one terminal, navigate to the `calibrator_ws/` directory and execute the following

command:

```
source install/setup.bash && ros2 launch calibrator Bringup plate.launch.py
```

This will start the ROS2 environment and allow the user to run the motor using the command line interface (CLI). Ensure that the calibration machine is flat and level with the rotating plate horizontal as shown in Figure 4.1c. When ready, set up the test by specifying the motor direction and speed in the second terminal window. The former can be set using the command:

```
ros2 service call /set_motor_dir calibrator_interfaces/SetBool "{data: DIR}"
```

Where **DIR** is the desired direction - **true** will set the motor direction to clockwise, **false** will set the motor direction to counter-clockwise. The motor speed can be set with the command:

```
ros2 service call /set_motor_speed calibrator_interfaces/SetFloat64 "{data: SPEED}"
```

Where **SPEED** is the desired test speed in degrees per second. Note that it is not accurate and needs to be checked using an external tachometer. For reference, calibration was done using three settings: 200, 300, and 400 which corresponded with the tachometer measurements of 305 deg/sec, 410 deg/sec, and 500 deg/sec, respectively. To start the motor spinning, execute the command:

```
ros2 service call /start_motor std_srvs/Trigger
```

After a moment, the plate will begin spinning at the specified speed and direction. Once you have collected the data, you can stop the motor using the command:

```
ros2 service call /stop_motor std_srvs/Trigger
```

After a few moments, the motor will come to a stop and the data can be offloaded. Note that this command resets the motor speed to a default value, so you will need to set the speed

using the commands again. It will be beneficial to start/stop recording at the start and end of every test so that a single log file represents a single test.

4.1.3 Accelerometer

The accelerometer must be calibrated on the gear and socket apparatus while it is vertical. The apparatus should be leveled such that the instruments on the calibration cube are perfectly vertical with respect to gravity and the other axes are planar to the Earth's surface. Choose an axis and place the positive direction downwards (-1g). The arrows on the coordinate reference markers on each device point towards the positive direction. Then, align the indicator on the gear with 0-degree marker on the socket, as shown in Figure 4.1b.

Rotate the calibration cube and gear in 45-degree increments, stopping for 30-seconds at each interval to collect an good average of data. It will be beneficial to start/stop recording at the start and end of each increment so that a single log file represents a single orientation.

4.1.4 Orientation

Roll and pitch orientation data can be collected simultaneously with the accelerometer calibration data as the devices are rotated about the x- and y-axis, respectively. This will give roll and pitch in 45-degree increments that can be analyzed for errors before and after calibration.

To get the yaw readings, we can re-use the gear and socket apparatus, this time placing it flat on a surface and placing the z-axis of the devices vertical. Make sure that the entire apparatus is far away from any magnetic influences and near the location where the magnetometer was calibrated. Start the gear at the 0-degree marker on the socket, then orient the entire apparatus to magnetic north as determined by an external compass as shown in Figure 4.2. Rotate the calibration cube and gear in 45-degree increments for 360-degrees, pausing for 30-seconds at each interval to get an average reading. It will be beneficial to start/stop recording at the start and end of each increment so that a single log file represents a single orientation.



Figure 4.2: Calibration cube aligned with an external compass.

4.2 Results

The calibration data was collected and ran through a calibration script in [15]. The raw log files were converted to CSVs using the x-IMU3 software conversion tool³. The resultant Magnetometer, Inertial, and Quaternion files were imported to the script using the `pandas` library. Then, they were initially processed into dictionaries the contained the relevant data for each device for each test for each axis.

The raw data from each test dataset was cleaned by removing outliers and then windowed to an approximately 15-second span in the center of the data. Windowing helped reduce processing time and automatically removed the heads and tails from the Thetis gyroscope plots, making analysis easier. This cleaned and windowed data were placed back into the dictionary for each dataset and then plotted into the raw data figures shown in Appendix K.

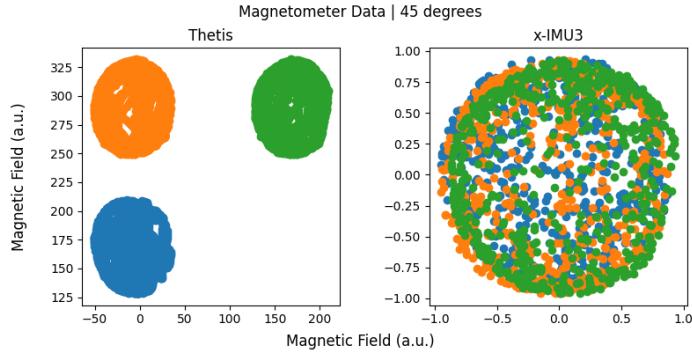
After plotting the raw inertial data, the calibration parameters were calculated for the accelerometer and gyroscope. The process used the mathematical techniques described in Chapter 2 and several external toolboxes. For determining the misalignment sensitivity matrices, the `scipy.optimize.minimize` toolbox was used with the Sequential Least Squares Programming (SLSQP) method. For this method, the objective function defined in Equation 2.12 was used as the scalar function.

³https://github.com/xioTechnologies/x-IMU3-Software/blob/main/Examples/Python/file_converter.py

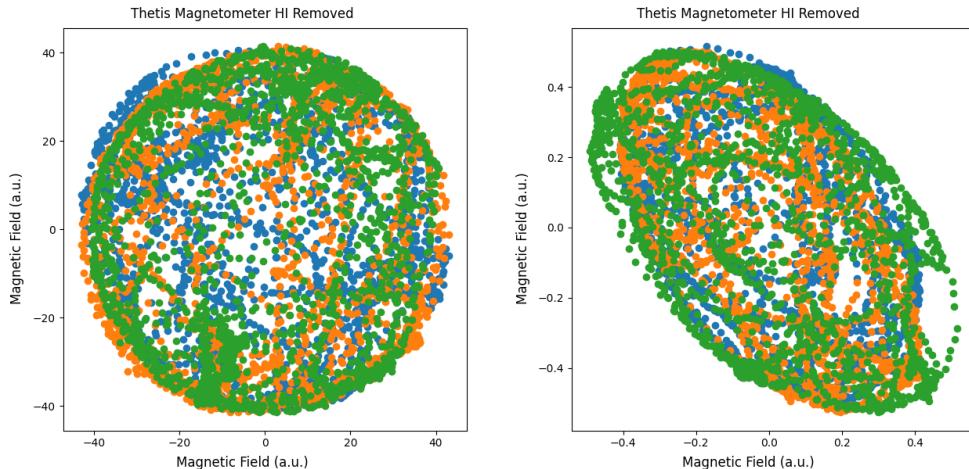
After producing the inertial calibration parameters, the script calculates the root-mean-square-error between for the devices with respect to ground truth and each other. These plots are shown in the following section. After determining the inertial RMSE, the magnetic parameters are calculated using the previously discussed mathematical methods and several plots are generated.

4.2.1 Magnetometer

This section shows the magnetometer calibration data as it is processed by the calibration script. Note that the x-IMU3 does not have a hard or soft iron plot as it was already calibrated to remove these distortions.



(a) Raw magnetometer calibration data from Thetis (left) and the x-IMU3 (right)



(b) Thetis magnetometer calibration data with hard iron distortion removed.

(c) Magnetometer calibration data with soft and hard iron distortion removed.

Figure 4.3: Magnetometer calibration process

The calibration data yielded the following parameters:

$$\mathbf{v} = \begin{bmatrix} -8.0752 \\ 168.7446 \\ 290.5438 \end{bmatrix}$$

$$W^{-1} = \begin{bmatrix} 0.0072 & -0.0040 & -0.0049 \\ -0.0040 & 0.0100 & -0.0048 \\ -0.0049 & -0.0048 & 0.0106 \end{bmatrix}$$

4.2.2 Gyroscope

The calibration data yielded the following parameters:

$$\mathbf{b}_g = \begin{bmatrix} -0.0003 \\ 0.0002 \\ 0.0002 \end{bmatrix}$$

$$S_g = \begin{bmatrix} 1.0110 & 0 & 0 \\ 0 & 0.9672 & 0 \\ 0 & 0 & 1.0041 \end{bmatrix}$$

$$M_g = \begin{bmatrix} 1 & 0.0037 & 0.0399 \\ 0.0100 & 1 & -0.0075 \\ -0.0259 & 0.0105 & 1 \end{bmatrix}$$

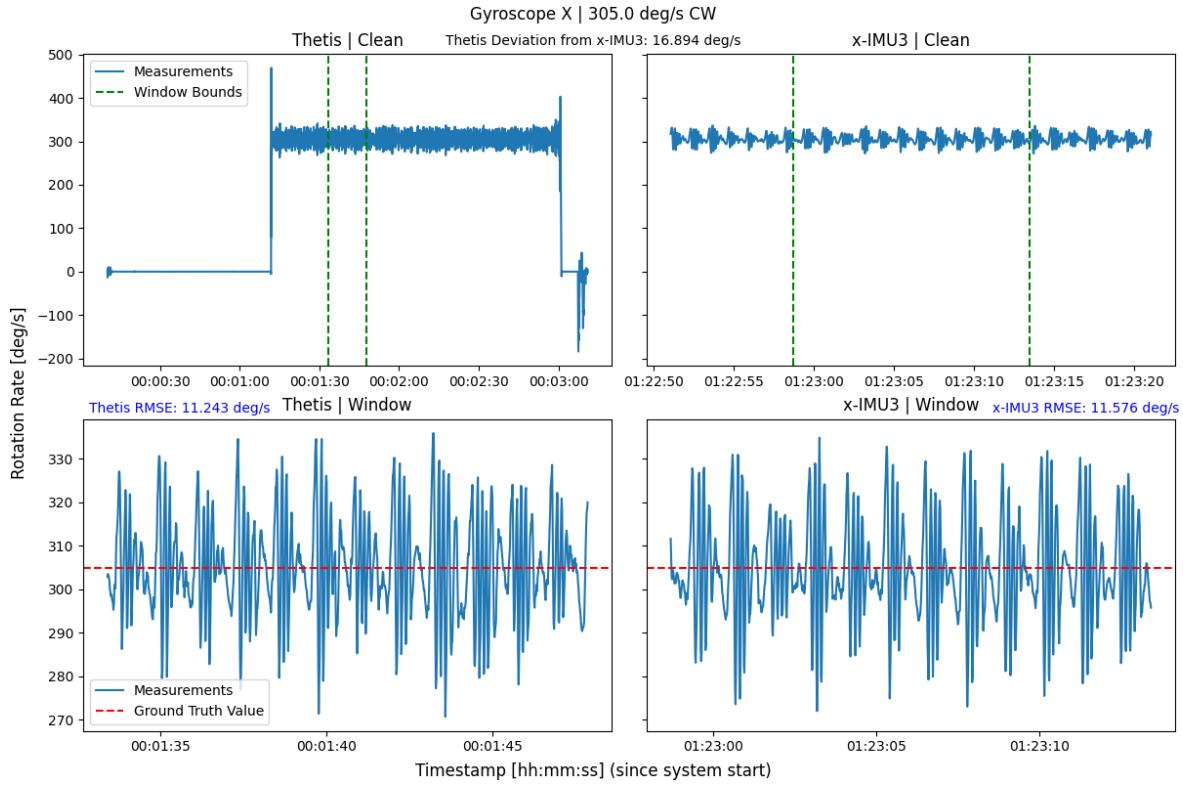


Figure 4.4: Raw and windowed gyroscope data from Thetis (left) and the x-IMU3 (right).

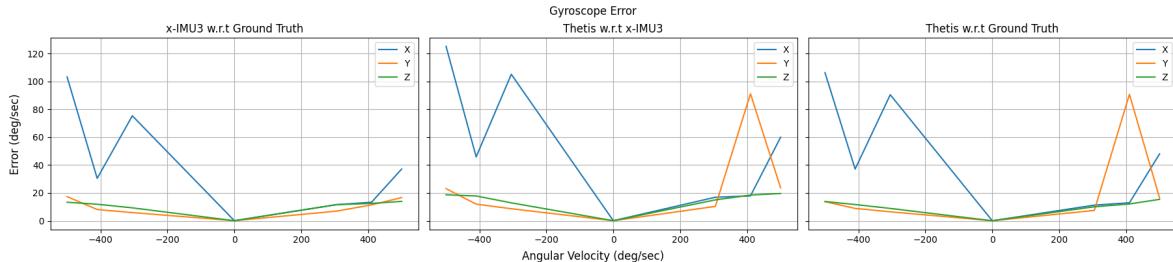


Figure 4.5: Root-Mean-Square-Error during calibration across all axes and test speeds. From left to right, Thetis with respect to the ground truth, Thetis with respect to the x-IMU3, and the x-IMU3 with respect to the ground truth.

4.2.3 Accelerometer

The calibration data yielded the following data:

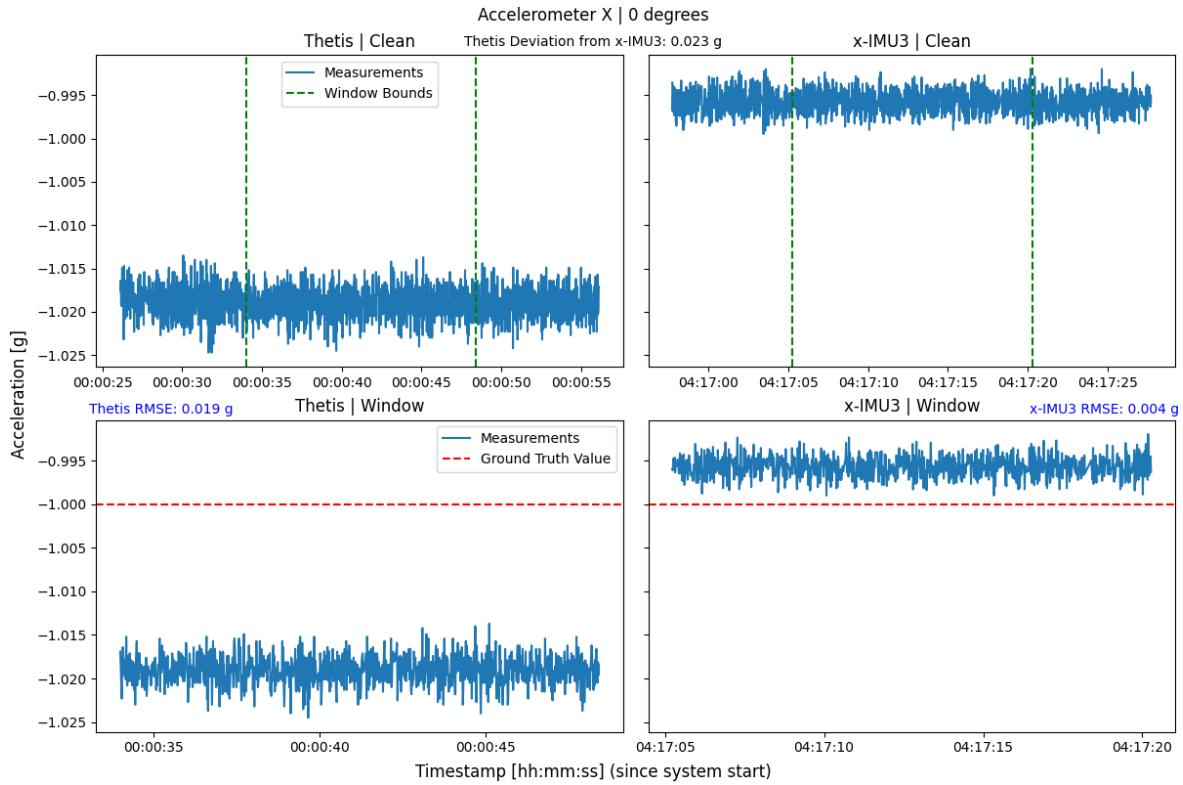


Figure 4.6: Raw and windowed accelerometer data from Thetis (left) and the x-IMU3 (right).

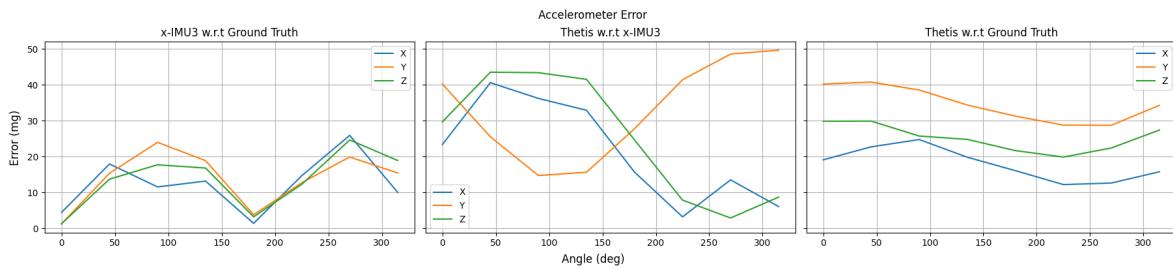


Figure 4.7: Root-Mean-Square-Error during calibration across all axes and test orientations. From left to right, Thetis with respect to the ground truth, Thetis with respect to the x-IMU3, and the x-IMU3 with respect to the ground truth.

$$\mathbf{b}_a = \begin{bmatrix} -0.0243 \\ 0.0342 \\ 0.0112 \end{bmatrix}$$

$$S_a = \begin{bmatrix} 0.0171 & 0 & 0 \\ 0 & 0.6617 & 0 \\ 0 & 0 & 1.0022 \end{bmatrix}$$

$$M_a = \begin{bmatrix} 1 & 0.0033 & 0.0401 \\ -0.3294 & 1 & -0.0067 \\ -58.2018 & 0.0135 & 1 \end{bmatrix}$$

4.3 Discussion

From the calibration results, we can identify several potential errors in the employed analytical methods.

Magnetometer First, regarding Figure 4.3c, the shape of the data points changes from spherical to ellipsoidal with a higher eccentricity than expected. The measurements also decrease in intensity by two orders of magnitude. This indicates that 1) the Li ellipsoid fitting algorithm may not be applied using the correct radius for the target ellipsoid, and 2) that the target ellipsoid has too high of eccentricity to be fitted into a sphere. Identifying and implementing a fix to this issue is currently out of the scope of this thesis. Therefore, it is recommended that only the hard iron offset, \mathbf{v} , be used for calibrating magnetic measurements for now.

With the hard iron offset applied to the data, the following RMSE values and per cent errors are reported in Table 4.1. Here, we can see a larger deviation from Thetis with respect to the ground truth and the x-IMU3, but the couple of percentage points difference between Thetis and the x-IMU3 could be caused by sensor performance and the x-IMU3's application of soft iron distortion compensation.

Comparison	RMSE (uT)	Percent Error
Thetis w.r.t Ground Truth ⁴	3.49	7.33%
Thetis w.r.t x-IMU3 ⁵	4.92	9.34%
x-IMU3 w.r.t Ground Truth ⁵	2.48	4.69%

Table 4.1: Comparison errors between the magnetometers onboard Thetis and the x-IMU3

Gyroscope In the gyroscope calibration data, we can see a lot of error introduced via noise into the measurements. The calibration machine does not spin at a constant rate and seems to have an induced oscillation and variance as the plate rotates. Additionally, the calibration cube’s moment of inertia is not centered directly on the rotation axes because the object is not weighted symmetrically. These errors lead to the large RMSE values shown in Figure 4.5 between Thetis and the x-IMU3. The large noise magnitudes for some of the x- and y-axis tests also introduced exceedingly large errors for the calibration data upwards of 100 deg/sec (approximately 20%).

The errors are consistent across the Thetis and x-IMU3 datasets suggesting that they are systematic errors with the calibration machine, and not related to any sensor itself. When the machine works properly (i.e. minimal noise in the data set), the error in Thetis and x-IMU3 with respect to each other and the ground truth varies between 5% and 10% which is within reason, given that Thetis is uncalibrated and not accounting for any measurement errors itself. If the machine was more stable and able to give more useful data, the calibration parameters could be calculated better and applied to reduce Thetis’s measurement error. However, because of the large discrepancies in the calibration data and error, it is not recommended to use the calculated calibration parameters for the gyroscope.

⁴As measured during calibration, the magnitude of the magnetic field was 45 uT

⁵From the x-IMU3 data sheet, the x-IMU3 readings were converted from a.u. to uT using 50 uT/a.u. as stated in the calibration certificate

Accelerometer The accelerometer calibration data looks good at first glance except for the average error being several tenths of a g. Even on the x-IMU3, the error remains higher than provided in the calibration certificate, implying that there are errors in the test apparatus. Most likely, the gear and socket were not perfectly flat and aligned with the gravitational field. Therefore, the other axes sensed the gravitational field more and caused the readings on the sensing axis to skew. The error persisted throughout all angles of testing on all axes as shown in Figure 4.7, further reinforcing the theory that it was introduced by the testing apparatus rather than the instruments themselves.

A very concerning problem are the misalignment sensitivity matrices calculated from the calibration data. The sensitivity matrix, S_a , should have values around 1 in the diagonal. The x-IMU3 calibration certificate shows that the sensitivity values for that device are within 1% of unity. The calculated values in $S_{a,11}$ and $S_{a,22}$ are significantly less than 1, meaning that the measurements will not be near their true value. Additionally, the misalignment matrix should have values that are near zero surrounding the diagonal 1's. The value of $M_{a,31}$ is -58.2018 which strongly indicates this calibration was not successful.

The most likely reason these discrepancies have occurred is because of the method from which the sensitivity and misalignment matrices are calculated. They are calculated using a non-linear optimization algorithm that tries to minimize the error found in an objective function (Equation 2.12). As the minimization occurs, it can fall into local minima that fulfill the boundary conditions laid out for this methodology, but will result in invalid calibration parameters. Further research needs to be performed on the best optimization techniques to find these calibration parameters. This research is out of the scope of this thesis due to the technical involvement. Based on these errors, it is not recommended to use the accelerometer calibration parameters calculated here.

Conclusion Due to errors in the inertial data collection and analytical methods, Thetis cannot have its accelerometer or gyroscope calibrated at this time. However, the magnetometer can be calibrated using only the hard iron offset. The calibration data taken between Thetis and the x-IMU3 suggests that the devices perform closely with one another and that

given more time and expertise with calibration, Thetis can become reasonably equivalent to the x-IMU3 in terms of sensor performance under the methodologies described above. The data collection process also demonstrated that Thetis can collect measurements reliably via its on board micro SD card storage and that can be offloaded and processed. The data was collected at 64 Hz sample rate which also validated some system requirements - this will be further explained in the next chapter.

Chapter 5

Verification and Validation

Now that we have an instrument designed and tested, how can we determine if it meets requirements or not? The system can be verified and then validated using a variety of techniques. First, to verify that a requirement is present, the component or subsystem can be examined independently. If it passes examination, we can update the traceability matrix to reflect that the requirement is now verified. Once the feature is integrated with the entire system, then we can determine if it is validated or not. Instead of examining the feature independently, it will be done so with the entire system to ensure that it functions properly. These examinations can fall under four main types:

Inspection: The nondestructive examination of a product or system using one or more of the five senses (visual, auditory, olfactory, tactile, taste). It may also include simple physical manipulation and measurements.

Demonstration: The manipulation of the product or system as it is intended to be used to verify that the results are as planned or expected.

Test: The verification of a product or system using a controlled and predefined series of inputs, data, or stimuli to ensure that the product or system will produce a very specific and predefined output as specified by the requirements.

Analysis: The verification of a product or system using models, calculations and testing equipment. Analysis allows someone to make predictive statements about the typical performance of a product or system based on the confirmed test results of a sample set or by combining the outcome of individual tests to conclude something new about the product or system. It is often used to predict the breaking point or failure of a product or system by using nondestructive tests to extrapolate the failure point.

For the following chapter, we shall examine each of Thetis' design capabilities from Tables 3.13 through 3.15 and verify their presence on the latest design. Then, we shall do the same for the stakeholder requirements.

5.1 Threshold Capabilities

101 - The system must be housed in an IP67-rated enclosure This capability was initially verified by locating a suitable enclosure for Thetis and confirming with the manufacturer through the product data sheet that the system was properly rated. Then, it was validated by placing the enclosure onto a Remotely Operated Vehicle (ROV) while it performed an underwater mission (Figure 5.1). Paper towels were placed within the enclosure so that if any water breached the seals, it would be immediately apparent. This represented a worst-case scenario and the enclosure held its seal down to a depth of 20-feet, meaning that it exceeded the capability specification.

To represent a more realistic use case, Thetis was placed into a cutout on a surfboard and deployed into the ocean to catch some waves. Out of nine separate deployments, the case seal failed three different times:

The first time, a screw was not present in one corner, thereby not clamping that section of the sealing material;

the second time, a screw was over-torqued and caused the area around the screw to crack, bypassing the seal;

Threshold				
ID	Description	Verified?	Validated?	Status
101	The system must be housed in an IP67-rated enclosure	Y	Y	D
102	The system enclosure must fit within the volume of 8" x 5" x 1.25"	Y	Y	D
103	The system software and firmware will be fully open-source	Y	Y	D
104	The system shall record inertial measurements at a minimum frequency of 64 Hz	Y	Y	D
105	The system shall be able to store data locally for up to 4 hours continuously	Y	Y	D
106	The system shall be able to operate for up to 4 hours continuously	Y	Y	D
107	The system shall have simple human interface mechanism for status and logging	Y	Y	D
108	The system firmware shall be an open-architecture	N	N	A
109	The system will be fully documented	Y	N	TIP
110	The system shall use version control software to track changes	Y	Y	D
111	The operator shall be able to offload data from the system	Y	Y	D
112	The system shall be capable of being assembled by hand using basic soldering tools	Y	Y	D

Table 5.1: Verification and validation of threshold capabilities

the third time, the same case as before was used accidentally.

All three failures were due to operator error proving that careful procedures need to be implemented for actual deployments. However, when the case was properly sealed by operators, the seals held and the electronics within were not damaged, even when completely submerged and subjected to dynamic forces as the surfboard rolled and slammed into waves.

Figure 5.1: Thetis (bottom) attached to a Blue ROV2 frame for enclosure testing.



Figure 5.2: Thetis (right) and the x-IMU3 (left) secured in a cutout on a modified surfboard during a deployment.



102 - The system enclosure must fit within a volume of 8" x 5" x 1.25" Like Capability 101, this was initially verified by inspection during the search for this enclosure and confirming the enclosure dimensions with the manufacturer. Then, it was validated by placing it securely into the cutouts in the surfboard made for the iPhone 6S - one of the devices Thetis is intended to replace, as shown in Figure 5.2.

103 - The system software and firmware will be fully open-source This capability is both verified and validated by inspection as the code is readily accessible on GitHub.

The firmware is broken into several sub repositories: Thetis-Firmware [14], ThetisLib [16], xioAPI-Arduino [18], Timer-Events-Arduino [17], and Fusion-Arduino [11], all of which are under the MIT license and available. Appendix L also includes a guide for setting up the development environment and forking the source code. Thetis has several tangential software packages that are also open source such as the scripts repository [15] used for data processing and analysis, the x-IMU3 GUI [53] used to visualize data and log from a host computer, and the code for the calibration machine [13].

104 - The system shall record inertial measurements at a minimum frequency of 64 Hz This capability requires a demonstration in order to be verified and validated. We can initially inspect the firmware to ensure that the inertial measurements are taken every 15.6 milliseconds, but it is not guaranteed that the system can consistently take measurements at that speed. Therefore, the best way to demonstrate this capability was during the calibration procedure detailed in Section 4.1. Thetis was set to record at 64 Hz and when the data was offloaded, it was confirmed to be taken at the appropriate interval. Therefore, this capability has been verified and validated within the system.

105 - The system shall be able to data locally for up to 4 hours continuously We can analyze the size of logging messages and micro SD card to determine if this capability is verified. In the latest version of the firmware, five messages are published to the data storage device: position, inertial, magnetic, quaternion, and euler angle. Combined, these messages take 198 bytes of space and occur, on average, 64 times per second for 12,672 bytes per second. There are 14,400 seconds in 4 hours, so multiplying these values together, we get 182.5 megabytes of storage required for 4 hours of use. The microSD cards used throughout testing are at least 4 gigabytes which gives an estimated 88 hours of continuous logging (Equation 5.1). This requirement was validated by running Thetis for four hours continuously and verifying that the log file was successfully created and maintained for that period.

$$t_{\text{samples}} = \frac{N_{\text{storage}}[\text{Bytes}]}{198[\text{Bytes}] \times 64[\text{s}^{-1}] \times 3600 \left[\frac{\text{s}}{\text{h}} \right]} = \frac{4[\text{GB}]}{198[\text{Bytes}] \times 64[\text{s}^{-1}] \times 3600 \left[\frac{\text{s}}{\text{h}} \right]} = 88[\text{h}] \quad (5.1)$$

106 - The system shall be able to operate for up to 4 hours continuously For this capability, we can verify it by analysis, starting with a couple of assumptions:

1. Battery capacity is 420 mAh with 3.7V nominal voltage,
2. current consumption without WiFi enabled is approximately 50 mA, and
3. current consumption with WiFi enabled is approximately 120 mA while transmitting.

The latter two assumptions are based on a zeroth-order estimate by summing together the estimated current consumption of the various components from their data sheets. We can then make a zeroth-order estimate of battery run time using Equation 5.2. This yields an estimated continuous battery life of 9.4 hours without WiFi and 3.9 hours with WiFi. It is important to note that the WiFi estimate assumes continuous transmission - the worst case scenario. In reality, this may not be accurate so the battery life may be longer. If it is below the four-hour threshold, then certain mitigations can be implemented like a burst-mode transmission of data every couple of seconds or minutes.

$$t_{\text{battery}} = \frac{V_{\text{battery}} \times I_{\text{battery}}}{V_{\text{supply}} \times I_{\text{mode}}} = \frac{3.7[\text{V}] \times 420[\text{mAh}]}{3.3[\text{V}] \times I_{\text{mode}}[\text{mA}]} \quad (5.2)$$

This capability was validated by running Thetis for four hours continuously from full battery power and ensuring that the battery voltage at the end of the test was within the safe operating limits. Specific power consumption tests were not performed due to the technical complexity of precisely measuring current draw and were not in the scope of this thesis.

107 - The system shall have a simple human interface mechanism for status and logging This capability can be verified and validated using inspection techniques. First, to enable logging, an operator only needs to hold the “log” button for a half second and the

same to stop logging. To convey status, the on-board RGB LED changes color and pattern. By referencing the current RGB LED color and pattern with the diagnostic LED table [12], then the operator can know what the system is doing. These features were used extensively throughout testing.

108 - The firmware shall be open architecture This capability is challenging to fully define and implement, hence its relatively low priority in the threshold category. To verify this capability has been met, we should consider the difficulty of adding a new feature or component to the firmware. The firmware uses an object-oriented approach with a star topology. This means that a new feature can be added by putting it into the appropriate class and tying it to other classes/functions through the main `Thetis` object. Similarly, we can add a new component by creating a new class in the library and then implementing it in the main class.

Validating this capability will require more research that is out the scope of this thesis to ensure a proper software architecture is implemented and followed.

109 - The system will be fully documented This capability can be validated and verified through testing. Multiple groups of students and stakeholders will be asked to perform simple tasks using Thetis such as replacing a component, assembling the board, adding a firmware feature, or performing calibration. If the participants are able to perform the task using the available documentation, then this capability has been verified and validated. Otherwise, the documentation needs to created or modified accordingly.

110 - The system shall use version control software to track changes As discussed in Section 2.4.1, GitHub is a centralized VCS solution that enables version history and tracking. Since all of the software and firmware for Thetis is on GitHub, this capability has been thoroughly verified and validated. Similarly, all of the hardware was designed in Fusion 360 which implements its own VCS solution and then backed up to GitHub.

111 - The operator shall be able to offload data from the system Since the operator needs to be able to pull data off the system after an experiment, this capability is verified and validated via demonstration. During the calibration process, data was routinely written to the onboard microSD card and then the operator could pull the card out, load it into a host computer, and transfer the data into an analysis script. Additionally, data was able to be recorded in real-time through the x-IMU3 GUI application on a host computer through a USB connection to Thetis. Both of these methods verified that the capability was implemented and it was validated by reliably used throughout multiple tests.

112 - The system shall be capable of being assembled using basic soldering tools This capability was clearly demonstrated in Section 3.7.2 where the board is shown to be assembled using solder paste, tweezers, a soldering iron, and a hot plate. We can also verify this capability by inspecting the various component used throughout the design. The smallest component is an “0603” which is 60 thousandths of an inch long by 30 thousandths wide. While certainly small, they can be manipulated by a steady hand and placed on the board with reasonable accuracy. For these reasons, this capability has been verified and validated.

5.2 Reach Capabilities

201 - The system shall use a GPS with a minimum 1 Hz update rate for position tracking This capability can be verified by inspecting the GPS used on Thetis. The GPS receiver uses the MTK3339 chipset¹ which is capable of sending position messages every 1 to 10 Hz, according to its manufacturer data sheet. This feature was validated by placing Thetis at a monument² with a known GPS coordinates for one hour. Readings were taken at 1 Hz and recorded to the data log file. Figure 5.3 shows the distribution of measurement

¹<https://www.adafruit.com/product/746>

²https://ngs.noaa.gov/cgi-bin/ds_mark.prl?PidBox=AK4011

Reach				
ID	Description	Verified?	Validated?	Status
201	The system shall use a GPS with a minimum 1 Hz update rate for position tracking	Y	Y	D
202	The system will have a simple logging and status interface accessible via web terminal	N	N	A
203	The system shall be able to monitor and report battery state of charge	Y	N	A
204	The system will have configurable settings that can be changed by the operator	Y	Y	D
205	The system can operate in a Wi-Fi access point or client mode	Y	N	A
206	The sensor measurements shall incorporate a calibration model	Y	N	TIP

Table 5.2: Verification and validation of reach capabilities

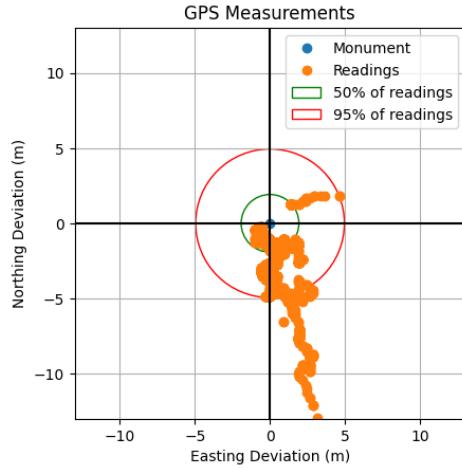
errors over the test in Northings and Eastings and the calibration script and data can be found in [15].

The measurement plot shows that 95% of the GPS measurements fall within 5-meters of the known GPS position. This is slightly worse performance than typically expected (about 3-meters [22]) but can be attributed to the small chip antenna on top of the module and the unoptimized PCB layout. On Revision F5, the GPS module has a broken ground plane beneath it. This results in RF noise and attenuation within the module that will decrease its accuracy. This issue is addressed in Revision F6 (Section 6.2). The drifting readings to the south east are also interesting and should be further investigated.

202 - The system will have a simple human interface accessible via web browser

In earlier versions of the firmware, there was a basic webpage that allowed operators to start/stop logging and check the device status. However, in the latest firmware version, this capability was removed to simplify testing. The hardware is still capable of providing this interface but it remains to be implemented. For more information, see Section 6.3.1.

Figure 5.3: GPS measurement deviations as measured from Thetis Revision F5 in middle of the day, clear conditions. Expected error should be less than 3-meters



203 - The system shall be able to monitor and report battery state of charge

This capability has been verified through inspection by integrating the MAX17048 onto the board design. This chip calculates state of charge and reports it to the microcontroller. Then the battery state is reported as a battery message through the device API. However, this capability has not been validated as the battery monitor chip has not been calibrated for the selected batteries nor analyzed for its accuracy. Further testing is required as detailed in Sections 6.3.2 and 6.4.2.

204 - The system will have configurable settings that can be changed by the operator

This capability was another one that was verified and validated throughout the calibration process by testing. The firmware implements and extends the xioAPI which provides an extensive list of settings that can be changed via JSON commands [12]. The calibration process required tweaks to some settings which were routinely done through the USB connection to a host computer. Additionally, during development of this feature, every setting was tested and validated to work.

205 - The system can operate in a Wi-Fi access point or client mode This capability can be verified by inspecting the manufacturer data sheet for the microcontroller used

on Thetis. The ESP32 series of chips used on the board are Wi-Fi enabled microcontrollers that can broadcast their own networks (access point mode) or connect to a local one (client mode). On Thetis, switching between these modes or disabling Wi-Fi altogether can be done through configuration commands. Performance of the Wi-Fi connection and other features have not been validated in the latest firmware and should be tested following the discussion in Sections 6.3.1 and 6.4.1.

206 - The sensor measurements shall incorporate a calibration model The capability is implemented by using the Madgwick sensor fusion algorithm described in Chapter 2 and the `Fusion` library made for that algorithm [11]. The `Fusion` library incorporates the calibration model described in Equations 2.5 and 2.14 by default and the specific calibration parameters are loaded as configuration settings. Due to the failure of the calibration process, this capability was not validated.

5.3 Stretch Capabilities

Stretch				
ID	Description	Verified?	Validated?	Status
301	The system file storage shall be accessible via web or USB interface	N	N	A
302	The system shall have a backup storage option in case primary storage fails	N	N	R
303	The system will use a microcontroller capable of machine learning using TinyML	N	N	UR
304	The system shall be able to log in a burst mode for up to 24 hours	N	N	UR

Table 5.3: Verification and validation of stretch capabilities

301 - The system file storage shall be accessible via web or USB interface From previous experiments with the ESP32-S2 and -S3 microcontrollers, it is possible to host a

File Transfer Protocol (FTP) server³ or have the device appear as a USB flash drive⁴ when connected to a host computer. However, in the latest firmware version, these features are not implemented. So, this capability is feasible, but not verified or validated in the current system.

302 - The system shall have a backup storage option in case primary storage fails

This capability was initially envisioned as a mitigation for any microSD card failure mode as described in the FMECA (Section 3.5.2). Especially in cases where the microSD card could become physically damaged during a test, the secondary option would still allow some data to be recovered. However, as explained in 3.8, this capability introduced system-breaking errors and the anticipated failure mode was never encountered throughout initial testing. Therefore, this capability was rejected in favor of a simplified and more reliable software and hardware design in Revision F6 (Section 6.2).

303 - The system will use a microcontroller capable of machine learning using TinyML

This capability was added to address some of the future efforts proposed in the next chapter. An experiment was run⁵ that proved the feasibility, but due to the technical effort and time constraints, this capability was not seriously considered for implementation.

304 - The system shall be able to log in a burst mode for up to 24 hours

This is another capability that was envisioned for a future use, but never seriously considered for implementation. The idea is to extend the battery life to capture inertial signals over a longer time span such as required for the WEAVE experiment proposed in Appendix I. However, this capability could be better implemented by a new hardware revision that can accept

³<https://www.mischianti.org/2020/02/08/ftp-server-on-esp8266-and-esp32>

⁴<https://github.com/Legohead259/TheTisLib/blob/ba94c8f8eba002fef2d88b5f21aa544ef2b5b2b1/Examples/usbmsc/test.ino>

⁵<https://github.com/Legohead259/Project-TheTis-TinyML-Example>

ID	Description	Verified?	Validated?	Status
SR 01	The system shall be able to record acceleration, rotation rate, orientation, and position	Y	Y	TIP
SR 02	The system shall be able to fit into a small IP67-rated, or better, enclosure	Y	Y	D
SR 03	The system shall be able to be powered by battery for more than 4 hours continuously	Y	Y	D
SR 04	The system shall be cheaper than \$200 per unit	Y	N	A
SR 05	The system shall use components that are readily available COTS	Y	Y	D
SR 06	The system designs shall be open source for modification by students	Y	Y	D
SR 07	The system shall allow users to change settings via interface and/or configuration file	Y	Y	D
SR 08	The system shall communicate extra-device using WiFi and USB	Y	Y	A
SR 09	The system shall have enough on-board storage for 4 hours of continuously logging at 64 Hz	Y	Y	D

Table 5.4: Verification and validation of stakeholder requirements

external power, instead of relying on a smaller internal battery. Theoretically, this is possible by integrating various component's sleep functionalities and other techniques.

5.4 Stakeholder Requirements

SR 01 - The system shall be able to record acceleration, rotation rate, orientation, and position The requirement was verified and validated throughout the calibration procedure described in the previous chapter. Thetis recorded and output the inertial measurements (rotation rate and acceleration), magnetometer readings, position measurements from the GPS, and orientation as both a quaternion and euler angles. This was recorded through

the on-board microSD card logger and through the x-IMU3 GUI application running on a host computer and connected via USB.

SR 02 - The system shall be able to fit into a small IP67-rated, or better, enclosure

As explained with Capabilities 101 and 102 in Section 5.1, this requirement was verified and validated by testing the enclosure at depth on an ROV and in a more realistic deployment on a surfboard. The enclosure held a water-tight seal when properly engaged and undamaged.

SR 03 - The system shall be able to be powered by battery for more than 4 hours continuously This requirement was verified using the analysis discussed in Section 5.1 with Capability 106. It was further validated by running the system for more than 4 hours and checking that the end battery voltage was at an acceptable level.

SR 04 - The system shall be cheaper than \$200 per unit This requirement was verified by inspecting the Bill of Materials for Thetis and checking that the summation was less than \$200. This requirement is difficult to validate however, because additional hardware revisions may be necessary to improve performance or add additional features. This estimate also does not include the development time or overall expense, which would increase the unit price.

SR 05 - The system shall use components that are readily available COTS This requirement was verified and validated by sourcing components from online distributors like DigiKey. All of the components used on Thetis can be purchased online and then assembled on the custom PCB. This requirement was tricky to implement during the chip shortage of 2020, but now that more components are more readily available, further hardware revisions should not have any problems continuing to use COTS parts.

SR 06 - The system shall be open source for modification by students This requirement is verified by the extensive use of GitHub repositories that are public-facing and accessible by anyone with an internet connection. Students can access the repositories,

download the source code and files, and make modifications as needed. To validate the requirement, multiple groups of students and stakeholders were gathered to attempt various tasks such as repair a chip, calibrate the board, add a new software feature, etc. As they were performing these tasks, they were given documentation and in turn provided feedback on the quality of the documentation and any clarifications they needed.

SR 07 - The system shall allow users to change settings via interface and/or configuration files The requirement was verified by implementing and extending the xioAPI in the Thetis firmware. This API provides a suite of settings to control how the board performs and its capabilities. The configurations can be changed via a web or USB interface to a host computer or by uploading a custom configuration file to the onboard flash storage. To validate the requirement, each setting was thoroughly tested to ensure it could be changed and the interface was used during the calibration process to tweak a couple of settings.

SR 08 - The system shall communicate extra-device using Wi-Fi and USB Since Thetis uses a microcontroller that is capable of USB and Wi-Fi connections, this requirement is verified by inspecting the manufacturer data sheet. However, the requirement remains unvalidated because the performance when using the wireless interface is severely limited. More testing is required as detailed in Sections 6.3.1 and 6.4.1

SR 09 - The system shall have enough on board storage for 4 hours of continuous logging at 64 Hz This requirement was verified and validated throughout the calibration process. Firstly, the calibration data was captured at 64 Hz proving the second part of the requirement. Then, as described in Section 5.1, the microSD card storage size was analyzed to ensure that it was large enough and Thetis was run for 4 hours, logging the entire time.

Chapter 6

Future Efforts and Potential Applications

As you may have noticed, this thesis was an intensive, multi-disciplinary effort that required in-depth knowledge of electronics, board design, software engineering, and systems engineering. Because of the breadth and depth required by this thesis, some areas were not covered due to technical or temporal constraints. A large portion of the software was only finalized in the months leading up to finishing this thesis and a hardware fault cost another couple of months of development work. Therefore, there is a lot left unfinished that is intended for future students to pick up as their own research projects. Some of these efforts are detailed in this chapter.

6.1 Calibration

As discussed at the end of Chapter 4, the calibration procedure failed for the inertial sensors. The specific reasons could not be determined before the thesis needed to be completed and were out of the technical scope anyways. Therefore, it will be required in the future to examine these procedures and determine the source of the flaws.

In order to accomplish this, the objective function and constraints should be examined

first. It is likely that the objective function was not used properly with the optimization procedure, producing results that fell into a local minima that satisfied the problem, but did not satisfy the calibration application. Therefore, the optimization function should be modified to include additional constraints or boundaries to limit the range of “acceptable”. This may improve the accuracy of the calculated misalignment and sensitivity matrices.

Additionally, these procedures should be more thoroughly tested and evaluated across a range of boards and types of sensors. The calibration script [15] created for this thesis is not sufficient for this task, so the concepts employed by it should be expanded as required. The measurements should be compared between the boards before and after calibration to determine the efficacy of the proposed procedures.

6.2 Hardware Revision F6

Based on testing and interviews with Dr. Madgwick [19], another hardware revision, Revision F6, was created to address some of the concerns with Revision F5. First, the secondary flash storage chip, the XTSD, was removed entirely since it was redundant and caused the hardware issues that crippled development of Revision F5 for months.

The space created by removing this chip allowed the MARG array to move to a section of the board that could be mechanically isolated from the rest of the board. This was necessary because MEMS sensors are affected by strain and in the configuration on Revision F5, the sensors were located at a point of maximum strain when the board was screwed into the enclosure. This would affect precision and could cause measurements to slightly vary depending on the torque of the screws used in the assembly.

Additionally, the “data ready” interrupts from the sensors were attached to the microcontroller which should improve performance by switching the measurement method to an interrupt-based one versus polling. By switching to an interrupt-based measurement method the readings can be taken at a much higher sample rate and be more accurate to the recorded timestamp. This method is also less computationally intensive on the microcontroller.

Then, the microcontroller was changed to one that had an antenna attached directly to

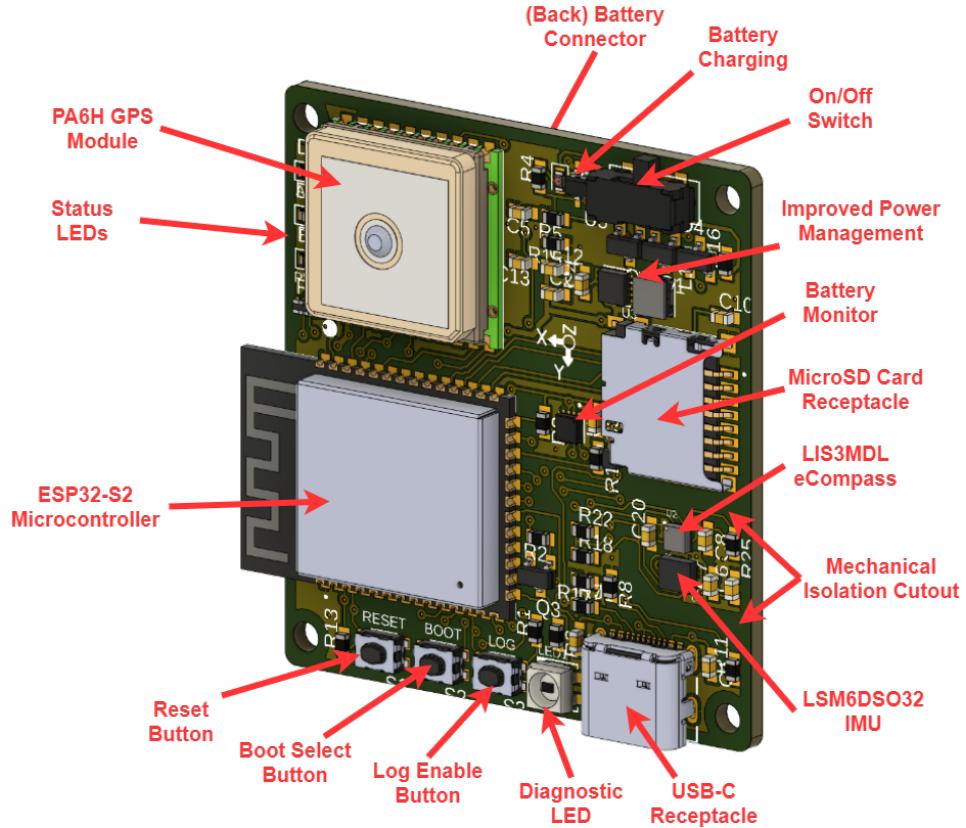
its PCB, removing the need for an external antenna. On the ESP32-S2 microcontroller, the chip would refuse to boot when an external antenna was attached. This issue was never fully investigated, but the onboard antenna should negate this problem and simplify the overall assembly.

Another change affects the GPS module. On Revision F5, the ground plane beneath the module was not contiguous and small relative to the antenna size. This introduced RF noise and attenuation, reducing overall accuracy. On the new revision, the ground plane is unbroken and stitched together with vias linking the top and bottom planes. This should improve performance slightly over Revision F5.

Finally, the power supply was substituted for a more efficient DC-DC converter which should improve efficiency and extend battery life.

This revision has been designed in ECAD, but was not ordered or built due to time constraints. Therefore, in the future, a student or group of students can build and verify and validate this design using the same procedures laid out for Revision F5.

Figure 6.1: Thetis Revision F6 PCB render with call outs for important components.



6.3 Software Improvements

There are several things that are feasible with the hardware available that were not able to be implemented before the thesis needed to be completed. Most of these features are not explicitly required by the system requirements, but they could improve the quality and usability of Thetis. These improvements can be tracked in the Thetis project on GitHub¹.

6.3.1 Wireless

Thetis currently struggles with sending data wirelessly to a host computer. The only protocol supported is the User Datagram Protocol (UDP) which is the simplest internet protocol (IP)

¹<https://github.com/users/Legohead259/projects/1>

to implement. Packets of data are broadcast with a header denoting the target IP address and port with no regard to signal strength or if the target receives the packet. This is the most efficient method as there is no handshaking or acknowledgement protocol to saturate the limited bandwidth and compute cycles.

However, as currently implemented, the process takes far to long to format a packet and send it to the target, resulting in a 90% drop in overall sensor sample rate. I believe the implementation of the `UDPServer` class is to blame, but more thorough research needs to be performed. For Thetis to be more effective for the operator, the UDP service should be fixed so that it performs adequately when wirelessly streaming data to a host computer.

Additionally, Thetis cannot handle when the x-IMU3 GUI first connects to it over UDP. The GUI application broadcasts over 70 requests for settings data simultaneously when first connecting or reading/writing settings which overloads the UDP client buffer on Thetis. This means that GUI cannot establish proper communication with the device and cannot change settings wirelessly. Again, to improve its effectiveness and ease of use, this should be fixed.

For deployments, it would also be preferred if the logging and status could be monitored remotely. This can be done by incorporating a basic web server into Thetis that shows an HTML page with a button for starting/stopping logging and a status box that mimics the on board LED. Some system information such as time and sensor health could also be presented to the operator for easier diagnostics. An example can be found in older versions of the firmware².

6.3.2 Settings/API

The settings used on Thetis follow the xioAPI specification which is a set of key/value pairs. Some of these settings need to be accessible on the device, but should not be overwritten (i.e. factory-settable only). Currently, Thetis's firmware does not protect these settings and does not support a factory program mode. This means that at any time, crucial settings

²<https://github.com/Legohead259/ThetisLib/blob/d5982b721212952777a11934ff837b62f7591226/src/radios/wifi.cpp>

like the calibration parameters, serial number, etc can be overwritten by the user or a host application. This commonly occurs with the x-IMU3 GUI application which writes blank strings, zero arrays, and zero matrices for some factory settings. When Thetis is connected to the GUI and settings are written to it, a lot of features are temporarily broken until the default values can be restored. This needs to be fixed to improve the interaction with the operators.

Some settings are also not fully implemented when they should be or they are only implemented at start up. For example, settings related to sensors such as calibration parameters and sample rate are configured when the system first boots. If these settings are changed during operation, the system must be reset for them to take effect. This can be streamlined by having callback functions for updating system settings when a new configuration is received.

Several message also need to be implemented, like the battery message. This will allow the log file to record the battery state over time and may aid with power verification and validation.

6.3.3 Data Storage

Currently, the data log recorded to the microSD card is a generic “logXXX.bin” where “XXX” is a three-digit number that is one larger than the previous log file. While simplistic to implement, this method makes tracking down specific log files difficult as the operator must track the number of times the system is restarted or reset. The settings already support custom log file names and prefixes and suffixes, so these should be implemented fully to make logging easier. Also, the file name extension should be changed to “.log” or something similar.

The firmware should also be updated so that the microSD is accessible via different methods. First, Thetis should show up as a USB flash drive when connected to a host computer. Files can be transferred over the USB connection without removing the microSD card. Second, the filesystem should be accessible over a FTP service where operators can wirelessly log in and copy/remove files. This will be very useful for deployed applications where Thetis cannot be easily opened or physically accessed.

6.3.4 Sensors

Many of the sensors implemented in the firmware are based upon the Adafruit libraries for those components. While this simplifies the integration, it bloats the memory and storage requirements. This also creates a dependency on third-party software which could be problematic in some situations. Therefore, it would be nicer to have all of the sensors packaged individually in libraries that are self-contained and conform to the I2CDevLib³ standards. This would improve the code performance and reduce memory requirements.

Additionally, the `Fusion` library has implemented some new features for accelerometer and magnetometer rejection and algorithm status flags that would be useful for the entire sensor fusion process. Therefore, the firmware should be migrated to this new version and have these features implemented.

If the hardware is migrated to Revision F6, the sensor polling method should also be swapped out for an interrupt-based approach. Instead of asking the sensors every period what their data is, the method must change to asking as soon as a pulse is received on the “data ready” input pins. This will make computation more efficient and allow for high sampling speeds. Also, the divisor settings can be implemented here to automatically average N number of samples before reporting it.

6.4 More Verification and Validation

Identifying and fixing a major design flaw with Thetis Revision F5 occupied a large amount of time that could have been used to further verify and validate some features. Unfortunately, that means there are still some things to be analyzed before the design can be considered “100%” delivered.

³<https://www.i2cdevlib.com/>

6.4.1 Wireless

The wireless features of Thetis need to be verified by proving that data and settings can be reliably streamed from the device to the host computer over a UDP connection. This should be done in both access point and client modes to cover the breadth of use cases. Of course, before these tests can be done, the problem with the UDP server and client should be addressed.

It is also important to characterize the strength, speed, and bandwidth of the wireless connection to understand the limits of the feature. To test the signal strength, Thetis should be enclosed in its container and then set to wireless access point mode. A smartphone with the appropriate application can then detect the Thetis access point and report its strength. The phone can then be moved several distances away from the access point and plot to get a rough strength over distance characteristic.

Similarly, Thetis can be placed into the wireless client mode and connected to a local access point. Assuming the message is implemented in the firmware, Thetis can report the wireless signal strength and log it. Thetis can be moved to several distances away from the access point and the log data can be plotted to roughly characterize the connection strength with respect to distance. The connection speed and bandwidth can be tested by running files of a known size through the FTP service and recording the amount of time to transfer the data. Ideally, there will be decent strength out to about 10-meters and the file transfers will occur within a minute or two.

6.4.2 Power

The verification and validation for power described in this thesis was just “does it last four hours or not?” It is important to fully characterize the power draw so that the operator can know the limits of the device. Additionally, measuring the power draw during certain operation modes can identify points for improvement and the hardware or software can be modified to increase battery life. The ICs within Thetis can also be put into a sleep mode (assuming that is supported in the firmware) and the power consumption can be characterized again. With the normal operating mode and sleep mode consumptions known, it will be

Figure 6.2: The rear of a three-axis gimbal used on UAVs for action cameras. Courtesy of RC Product India.



possible to set up a burst measurement mode to increase deployment life, if desired.

6.5 Calibration Machine

Calibrating Thetis is currently a manual process that is prone to errors from the procedure and the testing apparatus. For example, in the gear apparatus for calibrating the accelerometer and orientation, the calibration cube could accidentally be misplaced by a single tooth, causing a deviation of 5-degrees. This may not be immediately caught, introducing errors during processing. To automate the process and reduce potential errors, an expansion upon the current calibration machine is proposed which incorporates more features to improve usability.

6.5.1 Requirements

In order to effectively automate the process and reduce the chances of human error, the machine should be able to rotate the IMU about all three-axis on its own and in a controlled manner. Additionally, each of these rotations should be tracked with an absolute position sensor to understand the IMU's position and rotational velocity to a high degree of accuracy. The most compact way to accomplish this may be to borrow the design from a three-axis camera gimbal using high torque brushless DC motors, as shown in Figure 6.2.

Controls for the motor need to happen in near real-time and the main computer should

also be able to send commands and data around the system quickly and efficiently. This can be accomplished with a Data Distribution Service (DDS) topology that incorporates publish/subscribe and client/server interactions. The most popular framework for this type of network is ROS2, which is used on the current calibration machine and should be migratable to a new platform. The new calibration machine should also be open architecture such that new IMUs with different communication methods or new calibration sensors can be quickly integrated.

Finally, the main controller for the new machine should be powerful enough and have enough storage access to build a database with the calibration and validation results. Ideally, this controller would also have access to the Internet so that the information could be published to a cloud database for more accessibility. It would also be prudent to have the controller in a headed configuration with a simple touchscreen user interface to start/stop the procedure and monitor the system status.

6.5.2 Concept of Operations

For the new calibration machine, the idea is a one-touch solution that is fully automated, requiring minimal human intervention. At the start of the test, the operator can press the “Start” button and the machine will home each axis to a zero point and begin collecting the accelerometer and gyroscope datasets. Each axis will be individually rotated in 45-degree increments for a complete revolution. Then, they will be rotated at a few target speeds. Simultaneously, the IMU will be streaming data to the host computer for collection and processing. After the accelerometer and gyroscope datasets are collected, the IMU will be simultaneously rotated about all three axes to collect the magnetometer calibration dataset.

When these steps are completed, a master script will compute the calibration parameters and upload them to the IMU’s sensor fusion algorithm. Then, the testing procedure will restart with finer increments to increase the number of validation data points collected. After the sensor validation datasets are collected, the machine will rotate the IMU into various poses as part of the orientation validation dataset. Finally, the master script will

process the validation datasets into a single calibration certificate product that the operator can view.

While the entire process is occurring, crucial data will be shown on the screen to the operator along with an “ESTOP” button. If the machine detects an error, or the human operator wants to stop the test, there will be a set of commands to safely shutdown the system. Additionally, the machine will have several settings that can be tweaked by the operator prior to a calibration run - like the gimbal zero positions.

6.6 Floating Body Testing

The original intent of Thetis was to supplement class resources in the Ocean Engineering department. Naturally, this means that Thetis should be used in Ocean Engineering applications by students. Instructors may also find Thetis useful for their research as it is a small and cheap platform, perfect for a proof of concept experiment that could lead to larger revelations.

6.6.1 Surfboard

The first proposed application is in the Surf Engineering Analysis or Ocean Engineering Data Analysis course. Here, students will put Thetis into a surfboard and paddle out into the ocean surf to catch some waves. Thetis’s sensor suite will provide students data on the board’s orientation, acceleration, velocity, and position in the local and global coordinate frames which they can use as part of their class project. Thetis improves upon the sensors used for that course by having a larger sensor suite and a better sensor fusion algorithm to process the data. It would also be interesting to put the x-IMU3 into the same board and environment and see how they compare.

6.6.2 Model Barge

Another potential application is with the fluids laboratory or a naval architecture laboratory section. Currently, one lab experiment performed by students is the analysis of a vessel’s

movement based on weight placement and metacentric height. The model barge is equipped with inclinometers for the pitch and roll that students must analyze a video frame by frame to create a plot of the roll and pitch. If the video is blurry, or the vessel yaws during the test, this analysis can be difficult and time-consuming.

To simplify the process, Thetis can be placed on the barge and set to record the vessel's orientation at a much high frequency than a frame-by-frame analysis and do so more accurately. The data can also be streamed wirelessly to a host computer that would allow the instructor to more dynamically and visually demonstrate the effect of metacentric height on vessel stability. This can also be done to larger scale vessels in a larger tank or in a class room.

6.6.3 WEAVE

The last major application considered is the proposed Wave Estimating Algorithm for Vessels Experiment, or WEAVE. Based on the floating model barge experiments and previous work done in [31], WEAVE is a proposed study into an alternative method for wave characteristic measurement. Thetis can be placed on top or within a wave buoy and deployed on the ocean. While there, it can record the inertial characteristics of the buoy which can be used to determine the wave parameters over time. The parameters can be calculated in both the time and frequency domains which would be compared to the ground truth from the wave buoy. For more information, a whitepaper manuscript is presented in Appendix I.

Chapter 7

Conclusion

A low-cost, low-profile inertial data logger was designed, tested, and validated throughout this thesis. I set out to design the board as a solution that students could use in their projects in and out of the classroom and as a replacement for some of the current equipment. In putting the board together, I did my best to learn and follow system engineering best practices for a small embedded device like this and thoroughly validate the design before delivering it. There were some problems, a couple of hiccups, and one or two or three major disasters along the way, but ultimately Thetis works. It is an all-in-one datalogging solution, that is beneath \$200 in cost, is open source and publicly available, and is capable of recording inertial characteristics in 9 degrees of freedom with a GPS radio providing positional fix.

By reading through this thesis we have considered the two research questions posed in the introduction. We have learned how inertial measurement units work, and we have learned how we can design our own instrumentation board to capture and report those readings. With the capabilities found in Thetis, we can now move on to larger experiments incorporating it and continue to innovate in the research field. All the while remembering to teach the next generation everything we know and providing them a platform to exceed us.

References

- [1] Aceinna. *Aceinna INS401 Product Page*. <https://www.aceinna.com/inertial-systems/INS401>. Accessed: 2023.
- [2] Sal Afzal. *I2C Primer: What is I2C? (Part 1)*. <https://www.analog.com/en/technical-articles/i2c-primer-what-is-i2c-part-1.html>.
- [3] Elie Allouis et al. “A facility for the verification and validation of robotics and autonomy for planetary exploration”. In: (May 2013).
- [4] Alex Becker. *Kalman Filter: From the Ground Up*, 2nd ed. Alex Becker, 2023.
- [5] Jim Blom. *Analog vs. Digital*. <https://learn.sparkfun.com/tutorials/analog-vs-digital/all>.
- [6] Datawell BV. *Datawell Waverider Manual: DWR4*. 2023.
- [7] Peter Corke. *Robotics, Vision and Control: Fundamental Algorithms in MATLAB*, 2nd ed. Springer Tracts in Advanced Robotics, 2011.
- [8] John Craig. *Introduction to Robotics: Mechanics and Control*, 4th ed. Pearson, 2022.
- [9] Robert G. Dean and Robert A. Dalrymple. *Water Wave Mechanics for Engineers and Scientists*. Ed. by Philip L-F Liu. World Scientific, 1991.
- [10] Piyu Dhaker. *Introduction to SPI Interface*. <https://www.analog.com/en/analog-dialogue/articles/introduction-to-spi-interface.html>.
- [11] Braidan Duffy. *Fusion-Arduino*. Version 1.0.0. Oct. 2023. URL: <https://github.com/Legohead259/Fusion-Arduino>.

- [12] Braidan Duffy. *Thetis User Manual*. URL: <https://github.com/Legohead259/Thetis-User-Manual>.
- [13] Braidan Duffy. *Thetis-Calibration*. Version 1.0.0. Oct. 2023. URL: <https://github.com/Legohead259/Thetis-Calibration>.
- [14] Braidan Duffy. *Thetis-Firmware*. Version 2.0.0-beta1. June 2023. URL: <https://github.com/Legohead259/Thetis-Firmware>.
- [15] Braidan Duffy. *Thetis-Scripts*. Version 1.0.0. Oct. 2023. URL: <https://github.com/Legohead259/Thetis-Scripts>.
- [16] Braidan Duffy. *ThetisLib*. Version 2.0.0-beta1. June 2023. URL: <https://github.com/Legohead259/ThetisLib>.
- [17] Braidan Duffy. *Timer-Events-Arduino*. Version 1.0.0. Oct. 2023. URL: <https://github.com/Legohead259/Timer-Events-Arduino>.
- [18] Braidan Duffy. *xioAPI-Arduino*. Version 1.0.0. Oct. 2023. URL: <https://github.com/Legohead259/xioAPI-Arduino>.
- [19] Braidan Duffy and Sebastian Madgwick. *Discussions with Dr. Sebastian Madgwick*. Interview. 2023.
- [20] Marshall D. Earle. *Nondirection and Directional Wave Data Analysis Procedures*. Tech. rep. NDBC Technical Document 03-01. National Oceanic and Atmospheric Administration, 2003.
- [21] Sir William Rowan Hamilton. “The London, Edinburgh and Dublin Philosophical Magazine and Journal of Science (3rd Series)”. In: *On Quaternions; or on a new System of Imaginaries in Algebra*. Ed. by David R. Wilkins. 1844-1850.
- [22] Bernard Hofmann-Wellenhof, Herbert Lichtenegger, and James Collins. *Global Positioning System: Theory and Practice*. Springer, 2001.
- [23] HolyBro. *Holybro Pixhawk 6X Product Page*. <https://holybro.com/products/pixhawk-6x>. Accessed: 2023.

- [24] R. E. Kalman. “Transactions of the ASME - Journal of Basic Engineering, 82 (Series D)”. In: *A New Approach to Linear Filtering and Prediction Problems*. 1960, pp. 35–45.
- [25] KauaiLabs. *Kauai Labs NavX2 Micro Product Page*. <https://pdocs.kauailabs.com/navx-micro/>. Accessed: 2023.
- [26] Jerome B. Kuipers. *Quaternions and Rotation Sequences: A Primer with Applications to Orbits, Aerospace and Virtual Reality*. Princeton University Press, 2022.
- [27] Qingde Li and J.G. Griffiths. “Least squares ellipsoid specific fitting”. In: *Geometric Modeling and Processing, 2004. Proceedings*. 2004, pp. 335–340. DOI: [10.1109/GMAP.2004.1290055](https://doi.org/10.1109/GMAP.2004.1290055).
- [28] M.S. Longuet-Higgins, D.E. Cartwright, and N.D. Smith. “Observation of the directional spectrum of sea waves using the motions of a floating buoy”. In: *Ocean Wave Spectra* (1963), pp. 111–136.
- [29] LLC Lowell Instruments. *Lowell Instruments MAT-1 Product Page*. <https://lowellinstruments.com/products/mat-1-data-logger/>. Accessed: 2023.
- [30] Steven Macenski et al. “Robot Operating System 2: Design, architecture, and uses in the wild”. In: *Science Robotics* 7.66 (2022), eabm6074. DOI: [10.1126/scirobotics.abm6074](https://doi.org/10.1126/scirobotics.abm6074). URL: <https://www.science.org/doi/abs/10.1126/scirobotics.abm6074>.
- [31] Sebastian Madgwick. *Gait-Tracking*. Version 1.0.0. Sept. 2023. URL: <https://github.com/xioTechnologies/Gait-Tracking>.
- [32] Sebastian O.H. Madgwick. “AHRS algorithms and calibration solutions to facilitate new applications using low-cost MEMS”. PhD thesis. University of Bristol, 2014.
- [33] 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. DOI: [10.1109/TAC.2008.923738](https://doi.org/10.1109/TAC.2008.923738).
- [34] Sylvia Martinez and Gary Stager. *The Maker Movement: A Learning Revolution*. <https://iste.org/blog/The-maker-movement-A-learning-revolution>. Accessed: 2023. 2021.

- [35] MEMS Exchange. *What is MEMS Technology*. <https://www.mems-exchange.org/MEMS/what-is.html>. Accessed: 2023. 2023.
- [36] Movella. *xsens MTi-100 Product Page*. <https://www.movella.com/products/sensor-modules/xsens-mti-100-imu>. Accessed: 2023.
- [37] NASA. *NASA Systems Engineering Handbook*. Ed. by Steven R. Hirshorn. National Aeronautic and Space Administration, 2019.
- [38] NASA. “NASA Systems Engineering Handbook”. In: ed. by Steven R. Hirshorn. National Aeronautic and Space Administration, 2019. Chap. Appendix C: How to Write a Good Requirement, pp. 197–199.
- [39] Talat Ozyagcilar. *Calibrating an eCompass in the Presence of Hard- and Soft-Iron Interference*. Tech. rep. AN4246. Freescale Semiconductor, 2015.
- [40] Eric Peña and Mary Grace Legaspi. *UART: A Hardware Communication Protocol Understanding Universal Asynchronous Receiver/Transmitter*. <https://www.analog.com/en/analog-digital/articles/uart-a-hardware-communication-protocol.html>.
- [41] Michael R. Riley et al. “Ride Severity Index: A Simplified Approach for Comparing Peak Acceleration Responses of High-Speed Craft.” In: *Journal of Ship Production & Design* 29.1 (2013), pp. 25–35. ISSN: 21582866. URL: <https://portal.lib.fit.edu/login?url=https://search.ebscohost.com/login.aspx?direct=true&db=a9h&AN=85589608&site=eds-live>.
- [42] SparkFun. *SparkFun OpenLog Artemis Product Page*. <https://www.sparkfun.com/products/16832>. Accessed: 2023.
- [43] Sparton. *AHRS-8 Product Page*. <https://www.spartonnavex.com/ahrs-8/>. Accessed: 2023.
- [44] Jeff Sutherland and J.J. Sutherland. *SCRUM: The Art of Doing Twice the Work in Half the Time*. Currency, 2014.

- [45] SwiftNav. *SwiftNav Duro Inertial Product Page*. <https://www.swiftnav.com/duro-inertial>. Accessed: 2023.
- [46] x-io Technologies. *x-io Technologies x-IMU3 Product Page*. <https://x-io.co.uk/x-imu3/>. Accessed: 2023.
- [47] H. E. Thomason. *A general description of the st124-m inertial platform system*. Tech. rep. NASA Marshall Space Flight Center, Huntsville, AL, United States, 1965.
- [48] Mika Tuupola. *How to Calibrate a Magnetometer?* <https://www.appelsiini.net/2018/calibrate-magnetometer/>. 2018.
- [49] VectorNav. *VectorNav VN200 Product Page*. <https://www.vectornav.com/products/detail/vn-200>. Accessed: 2023.
- [50] VectorNav. *What is an inertial measurement unit*. <https://www.vectornav.com/resources/inertial-navigation-articles/what-is-an-inertial-measurement-unit-imu>. Accessed: 2023. 2023.
- [51] Elecia White. *Making Embedded Systems*. Ed. by Andy Oram and Mike Hendrickson. O'Reilly Media, Inc., 2011.
- [52] Jin Wu et al. “Fast Complementary Filter for Attitude Estimation Using Low-Cost MARG Sensors”. In: *IEEE Sensors Journal* 16.18 (2016), pp. 6997–7007. DOI: 10.1109/JSEN.2016.2589660.
- [53] xio-Technologies. *x-IMU3-Software*. Version 1.3.1. Oct. 2023. URL: <https://github.com/xioTechnologies/x-IMU3-Software>.

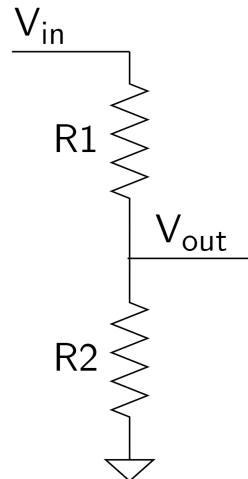
Appendix A

Voltage Divider

The most basic way to measure resistance is with a voltage divider. A voltage divider consists of two resistors in series that go between an input voltage and ground, as shown in the schematic below. The resistive sensor is placed at the R_2 position and we want to read the potential between the top of R_2 and ground, V_{out} to determine the sensor's resistance. This voltage can be determined by the equation:

$$V_{out} = V_{in} \frac{R_2}{R_1 + R_2} \quad (\text{A.1})$$

Figure A.1: A typical voltage divider circuit



A.1 Example: Battery Monitor

Let's say a device uses a 3.3V-logic microcontroller. It can only accept signals up to 3.3V before risking damage to its circuits. If you want your device to be portable, you can use a lithium polymer (LiPo) battery, but these must be carefully monitored to prevent over discharge. A single-cell LiPo battery has a peak voltage of 4.2V which is higher than what the microcontroller can handle. So, we must step down the voltage such that the peak battery voltage matches the peak input voltage of the microcontroller.

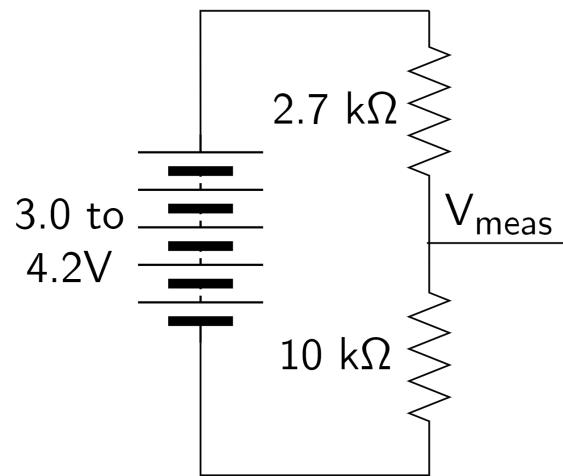
This is a perfect application for a voltage divider, but we must first determine what two resistors will be used. To linearize (simplify) the problem, we will arbitrarily set $R_2 = 10\text{k}\Omega$. Then, we can rearrange the problem to solve for R_1

$$\begin{aligned} V_{out} &= V_{in} \frac{R_2}{R_1 + R_2} \\ \frac{R_2}{R_1 + R_2} &= \frac{V_{out}}{V_{in}} \\ R_1 + R_2 &= \frac{V_{in} \cdot R_2}{V_{out}} \\ R_1 &= \frac{V_{in} \cdot R_2}{V_{out}} - R_2 \\ &= \frac{4.2 \cdot 10 \times 10^3}{3.3} - 10 \times 10^3 \end{aligned}$$

$$R_1 = 2727\Omega$$

To get a resistor that is exactly 2727Ω is very difficult, but we can approximate this value using the much more common $2.7\text{k}\Omega$ resistor without introducing much error into our measurements.

Figure A.2: A simple battery monitor circuit for 3.3V logic level microcontrollers.

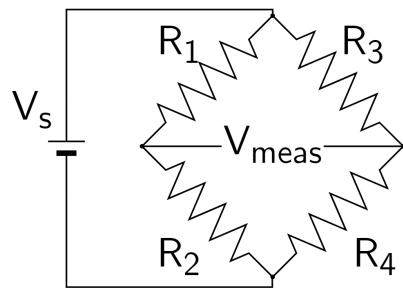


Appendix B

Wheatstone Bridge

The Wheatstone Bridge is a configuration of four resistors in two voltage dividers, as shown in Figure B.1, that is sensitive to minute changes in resistance. Resistors R_1 and R_2 form the first divider and R_3 and R_4 form the second divider. The bridge has at least three different solutions, depending on the configuration and desired application. For applications where R_1 , R_2 , and R_3 are known to a high precision, then R_4 can be determined to an equally high precision by adjusting R_3 until the voltage potential between points B and D is near 0, i.e. the bridge is “balanced”. However, for most embedded applications, R_4 cannot be determined by balancing the bridge, so either the general solution or linearization must be used. For simplicity, only the linearization solution will be examined here.

Figure B.1: A generalized wheatstone bridge

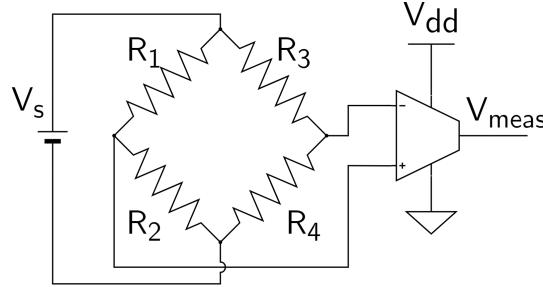


For the linear solution, the circuit adds an operational amplifier between points D and B and assumes that $R_1 = R_2 = R_3 = R_0$ and $R_4 = R_0 + \Delta R$ and the op-amp is an ideal

component ($R_{\text{amp}} = 0$). Because of this, the voltage potential at Point B will have a constant value of:

$$V_B = \frac{R_0 V_s}{R_0 + R_0} = \frac{V_s}{2}$$

Figure B.2: A linearized wheatstone bridge



Likewise, the op-amp will force the voltage at Point D to have the same voltage as B such that, $V_D = V_B = \frac{V_s}{2}$. This forces a constant current of $\frac{V_s}{2R_0}$ through R_3 and into the sensor. By Ohm's law, the voltage across the sensor will therefore be:

$$\begin{aligned} V_4 &= \frac{V_s}{2R_0} \cdot R_0(1 + \Delta R) \\ &= \frac{V_s}{2} + \frac{V_s}{2}\Delta R \end{aligned}$$

By applying Kirchoff's voltage law, the potential between the amplifier output and ground, V_{out} , is:

$$\begin{aligned} V_{\text{meas}} &= -V_4 + V_D \\ &= -\left(\frac{V_s}{2} + \frac{V_s}{2}\Delta R\right) + V_D \\ V_{\text{meas}} &= -\frac{V_s}{2}\Delta R \end{aligned}$$

The measured output voltage now linearly changes with the resistive sensor regardless

of if the sensor changes linearly itself. If the manufacturer of the sensor provides a table or equation that relates the sensor resistance to a real-world value, we can find the sensor resistance via:

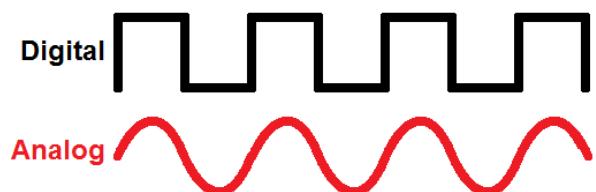
$$\begin{aligned} R_4 &= R_0 + \Delta R \\ &= R_0 - \frac{2V_{out}}{V_s} \end{aligned}$$

Appendix C

Analog Measurement

In electronics, there are two different types of signals: digital and analog. Digital signals are either a “high” voltage, or a “low” voltage, representing either a logical 0 or logical 1. These signals are great for threshold measurements such as “is there a cart present at the start of the ride, yes or no?”, but cannot express a range of values. Conversely, analog signals are better suited to expressing real-world values that naturally vary from a minimum to a maximum and can be any value in between. Thus, analog signals are prevalent for devices that measure “real” things such as speed, acceleration, rotation rates, barometric pressures, etc.

Figure C.1: A digital square wave (top) versus an analog sinusoidal wave (bottom). Courtesy of SparkFun [5]



Most analog sensors are resistive types where their electrical resistance changes to scale with a real-world measurement range. Ohm’s Law defines the relation between voltage, (V) resistance (R), and current (I) as $V = IR$. Therefore, if we can measure the voltage drop

across a resistive sensor and correlate it with a manufacturer-provided equation or table, we can quantify a real-world phenomenon. The resistance is best measured using a Wheatstone Bridge and an amplifier which can generate a voltage that is directly proportional to the change in resistance, ΔR

C.1 Analog to Digital Conversion

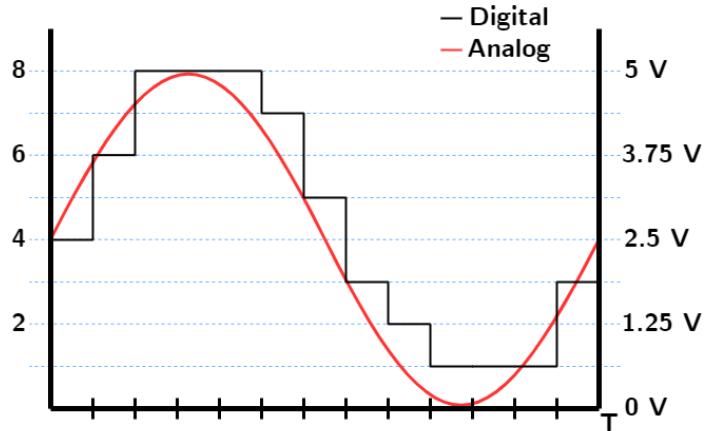
Now that we can get a sensor reading as an analog (changing) voltage, we need to find a way to translate it into a useful form. Decades ago, the analog voltage from a sensor was plotted onto a chart with respect to time and an analyst could run the conversion point by point. In a real-world experiment, it would be very difficult to deploy a large plotter to capture field data. We also want to work smarter and not harder, so we want a computer to perform the analysis for us. What can we do? Since we live in the digital age, we can convert the analog signal to a digital one and then we can store it on a digital media device such as an SD card then load it directly into a computer program like Excel to analyze it.

This process is done with an analog to digital converter circuit or ADC. The ADC samples an analog waveform at a given frequency and places the voltage level into bins for each measurement. There are 2^N bins in an ADC depending on its resolution, N . For each measurement, the ADC will also encode the voltage reading as a binary number and save it to a register.

This encoded binary number is expressed as “counts” which can be read by a microcontroller or other digital system. The number of counts recorded can be converted back to a voltage value at any time using the equation below. However, this conversion will lose some of the original precision of the analog signal, depending on the resolution of the ADC. A higher resolution will mean a more precise reading.

$$V = V_s \frac{\text{counts}}{2^N}$$

Figure C.2: A 3-bit ADC waveform converting an analog sinusoidal voltage wave (red) of period, T , to a digital representation (black). Each ADC bin is shown by a blue dashed line.



Aside: Registers

Digital values are stored in memory as 1's and 0's. Each bit of memory can be stored in a contiguous piece called a “register”. A digital system can access registers of memory to grab values and perform calculations on them. For example, an 8-bit ADC will store the encoding of an input signal in an 8-bit register. A microcontroller can then read this value from the ADC and convert it to a real-world value, or save it to non-volatile memory for logging.

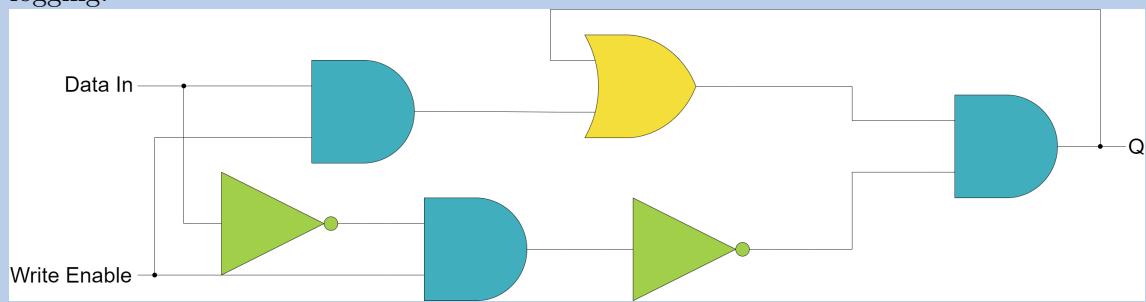


Figure C.3: A gated latch circuit used to store 1 bit of memory.

Appendix D

Digital Communication Methods

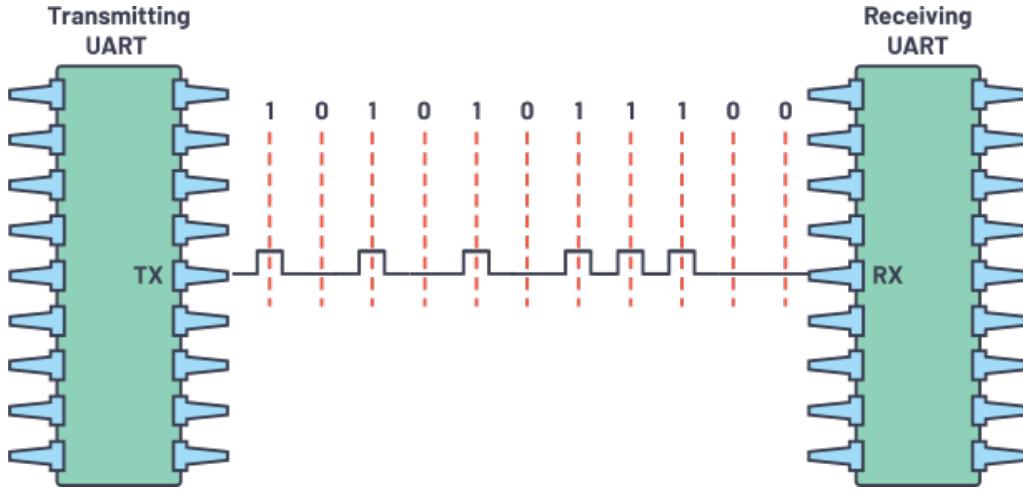
Now that we have converted the analog sensor reading into the digital realm, we have to move the data around to use it. There are many communication methods that utilize a variety of protocols, but for simplicity we will be discussing three of the most common in small digital circuits. Each of these methods utilize “Transistor-Transistor-Logic” or TTL voltages, meaning the signals use 0V for a logical 0, and any other voltage above a threshold is considered a logical 1. Each method is also known as a serial communication method as each bit in the message is communicated in series.

D.1 Universal Asynchronous Receive and Transmit

The Universal Asynchronous Receive and Transmit (UART) bus is a full-duplex digital communication method where the receiver and transmitter of information can transmit asynchronously. This is one of the simplest communication interfaces as a controller only needs to pull the data bus to the “low” state at certain intervals while transmitting. There is no addressing or synchronization required, so the transmitter can just “shout” the bits down the line and does not care if the receiver gets them or not. Since its asynchronous, both the receiver and transmitter must agree on the baudrate, or bits-per-second, beforehand and will set their own internal clocks to the appropriate speed. By timing the arrival of “low” or

“high” signals, the controllers can interpret data in the transmission packet. If the baudrates are not synchronized properly, the interval between the bits will be different and data will become corrupted at either end. Further reading can be done at Analog Dialogue [40].

Figure D.1: The UART protocol diagram with a small waveform example. Courtesy of Eric Pena and Mary Grace Legaspi [40].



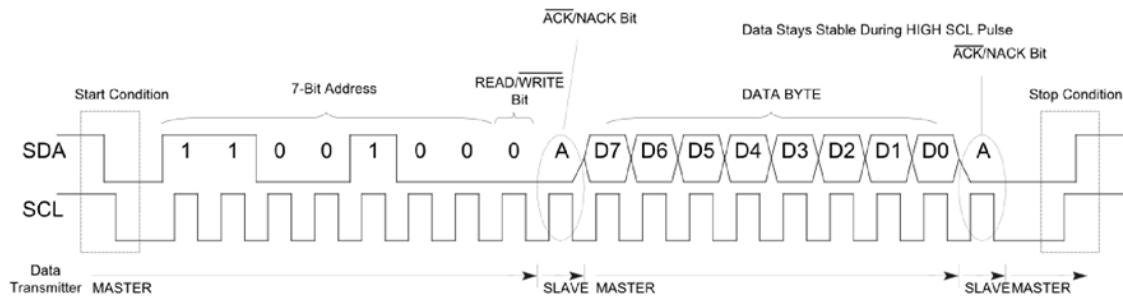
D.2 Inter-Integrated Circuit

The I2C, or Inter-Integrated Circuit, communication bus is most commonly used for slower speed transmissions between integrated circuits on a circuit board. It is a synchronous, multi-nodal, half-duplex topology which means that multiple controllers and responders can exist on the same bus and communicate, but not simultaneously. I2C uses two pins for communication, one for data (SDA) and one for clock (SCK). Unlike UART, I2C devices do not need to agree on a clock speed before communication begins, the clock line speed is adjusted by the controller and all devices automatically synchronize to it. Additionally, I2C devices have a one-byte address that distinguish them and allow for a controller to request data from a specific responder. This limits the number of unique devices on an I2C bus to 255 and can create some problems if multiple I2C devices of the same address are present.

For the latter problem, I2C multiplexing solutions exist to mitigate the issue ³.

To begin a message, the controller will initialize a start condition on the line then broadcast the desired device address. This lets all devices on the bus know that the controller wants to talk with a specific device. The controller will then follow-up with a one-byte data frame. Depending on the responder's programming, it will respond with another data frame that the controller will receive. A typical example of this is a controller querying a device for the value of a register in its memory. Further reading can be done at Analog Devices [2].

Figure D.2: The I2C protocol diagram with a successful write byte transmission. Courtesy of Sal Afzal [2].



D.3 Serial Peripheral Interface

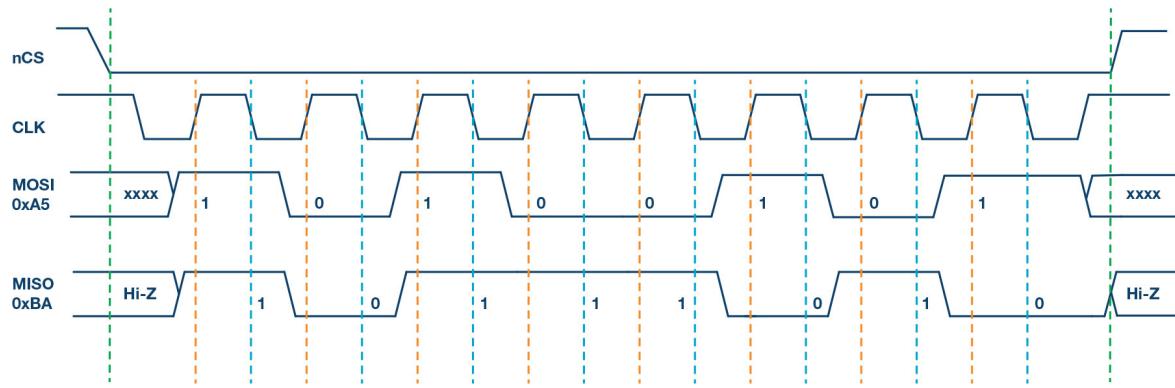
The Serial Peripheral Bus (SPI) is the slightly modified version of the UART protocol. SPI introduces clock synchronization so that a main and subnode do not have to agree on a communication rate beforehand and also allows multiple subnodes to be present on the same bus. By toggling an enable pin, the main node can tell a subnode to ignore or respond to a request on the data line. SPI uses four lines to communicate: Main Out Sub In (MOSI), Main In, Sub Out (MISO), Clock (SCK), and Chip Select (CS). A typical multi-subnode topology is shown in the figure below.

To begin a message, the main node will pull the CS pin for the desired subnode low and

³<https://www.adafruit.com/product/2717>

send out a message on the MOSI line. When the subnode receives the message, it can respond depending on its programming. In the case of a data storage unit like RAM or an SD card, the subnode will just write data to a specified register from the main node's message. If the subnode is a sensor, it may respond with the value from one or more registers in its memory. More reading can be done at Analog Dialogue [10].

Figure D.3: The SPI protocol diagram with a successful write byte correspondence. Courtesy of Piyu Dhaker [10].



Appendix E

Gyroscope Coriolis Effect Proof

The position of the center of mass in the gyroscope body frame is given by:

$$B_c = \begin{bmatrix} x \\ y \end{bmatrix} \quad (\text{E.1})$$

The inertial velocity of the mass is given by the positional derivative and the tangential velocity due to rotation:

$$\dot{B}_c = \begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} + B_\omega \times B_r = \begin{bmatrix} \dot{x} - \omega y \\ \dot{y} + \omega x \end{bmatrix} \quad (\text{E.2})$$

The inertial acceleration is the next derivative given by:

$$\ddot{B}_c = \begin{bmatrix} \ddot{x} \\ \ddot{y} \end{bmatrix} = \begin{bmatrix} \dot{x} - \omega y \\ \dot{y} + \omega x \end{bmatrix} + B_\omega \times B_{\dot{r}} = \begin{bmatrix} \ddot{x} - 2\omega\dot{y} - \omega^2 x \\ \ddot{y} + 2\omega\dot{x} - \omega^2 y \end{bmatrix} \quad (\text{E.3})$$

The first element in E.3 represents the acceleration experienced by the x-axis (driven axis). This axis is actively controlled by the gyroscope driving circuit. The second element is that of the sensing axis (y-axis). Recall Newton's Second Law of Motion, $F = ma$, which for the y-axis yields:

$$F_y = mB_{c,y} = m(\ddot{y} + 2\omega\dot{x} - \omega^2 y) \quad (\text{E.4})$$

If the mass starts at the resting position, $y = \dot{y} = \ddot{y} = 0$. Therefore, we get:

$$F_y = 2m\omega\dot{x} \quad (\text{E.5})$$

Since the mass is displaced along the x-axis at a high frequency and short distance, \dot{x} is significant and the Coriolis effect generates a high amplitude, proportional displacement in the y-axis. The tuning fork configuration essentially doubles the displacement and makes capacitive detection easier. Additionally, when undergoing linear acceleration, the masses move equally, minimizing sensitivity to shock, vibration, and tilt.

Appendix F

Rotation Matrices

A rotation matrix is a mathematic model for translating one body's inertial reference frame to another, e.g. local frame to a global frame. These are used for Eulerian transformations of vectors (i.e. using roll, pitch, and yaw). To begin deriving these matrices, let's start with a two dimensional rotation matrix:

$$R(\theta) = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \quad (\text{F.1})$$

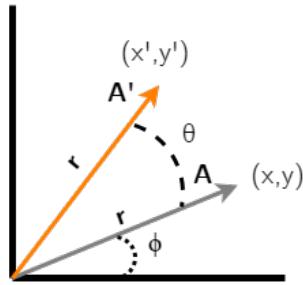
If we rotate a vector, $\mathbf{v} = [x, y]$ with a magnitude, r by θ degrees about the z-axis (as shown in Figure F.1), it will arrive at a new coordinate $[x', y']$. We can express \mathbf{v} in polar form as:

$$\begin{aligned} x &= r \cos \phi \\ y &= r \sin \phi \end{aligned} \quad (\text{F.2})$$

Similarly, the rotated vector, \mathbf{v}' in polar form is expressed as:

$$\begin{aligned}x' &= r \cos(\phi + \theta) \\y' &= r \sin(\phi + \theta)\end{aligned}\tag{F.3}$$

Figure F.1: 2 dimensional rotation of a vector, A , to a new set of coordinates, A' .



Expanding F.3 and using trigonometric identities:

$$\begin{aligned}x' &= r(\cos \phi \cos \theta + \sin \phi \sin \theta) \\&= r \cos \phi \cos \theta + r \sin \phi \sin \theta \\&= x \cos \theta + y \sin \theta \\y' &= r(\sin \phi \sin \theta + \cos \phi \sin \theta) \\&= r \sin \phi \cos \theta + r \cos \phi \sin \theta \\&= y \cos \theta + x \sin \theta\end{aligned}\tag{F.4}$$

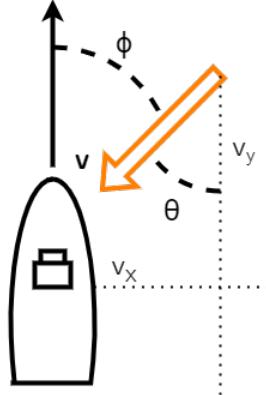
We can then express the resultant equations in the form of a 2×2 rotation matrix, R , yielding:

$$\begin{bmatrix}x' \\ y'\end{bmatrix} = \begin{bmatrix}\cos \theta & -\sin \theta \\ \sin \theta & \cos \theta\end{bmatrix} \begin{bmatrix}x \\ y\end{bmatrix} = R(\theta) \begin{bmatrix}x \\ y\end{bmatrix}\tag{F.5}$$

Example If we have a vessel moving in the \hat{x} direction at some velocity and it encounters a current, \mathbf{v} coming in from the starboard quarter, what is the current's influence on the

vessel?

Figure F.2: A vessel encounters a current (orange), \mathbf{v} , from the forward starboard quarter.



First, we can use the parallel axis theorem to determine that the current is impinging on the vessel at an angle, θ , relative to its longitudinal (y) axis. We can consider the current vector to be in the global frame and to find its influence on the vessel, we must rotate it to the vessel's local frame and get the resultant vector, $[v_x, v_y]$. Using a rotation matrix, we can determine that:

$$\begin{bmatrix} v_x \\ v_y \end{bmatrix} = R(\theta) \begin{bmatrix} v_E \\ v_N \end{bmatrix}$$

F.1 Three Dimensional Rotation Matrix

In three dimensional space, rotation can be performed about the x-, y-, or z-axis. A basic rotation that occurs around a single axis is defined as an “elementary rotation” and given by the following rotation matrices:

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

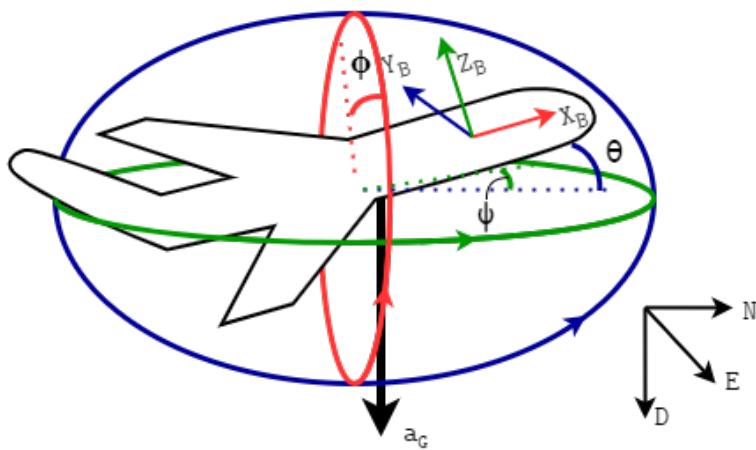
$$R_y(\theta) = \begin{bmatrix} \cos \theta & 0 & \sin \theta \\ 0 & 1 & 0 \\ -\sin \theta & 0 & \cos \theta \end{bmatrix}$$

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

To rotate a three dimensional vector from one frame to another, we must select the order of the axes to be rotated about, then multiply the rotation matrices and vector together.

Example What are the components of gravitational acceleration, ${}^G\mathbf{a}$ that affect an airplane in flight at some arbitrary roll (ϕ), pitch (θ), and yaw (ψ)?

Figure F.3: A plane rotated at some arbitrary roll (red), ϕ , pitch (blue), θ , and yaw (green), ψ , experiences gravitational acceleration in all three axes.



First, we define the gravitational acceleration vector as existing in the global North-East-

Down (NED) reference frame, ${}^G\mathbf{a} = [a_N, a_E, a_D] = [0, 0, 1]$. We will also assume that all of the rotations of the plane are relative to the same NED frame. We can then perform a yaw-pitch-roll rotation of ${}^G\mathbf{a}$ to get the gravitational acceleration in terms of the plane's coordinate frame, ${}^B\mathbf{a}$:

$$\begin{aligned} {}_B^G\mathbf{a} &= R_z(\psi)R_y(\theta)R_x(\phi)a_G \\ &= \begin{bmatrix} \cos \psi & -\sin \psi & 0 \\ \sin \psi & \cos \psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos \theta & 0 & \sin \theta \\ 0 & 1 & 0 \\ -\sin \theta & 0 & \cos \theta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \phi & -\sin \phi \\ 0 & \sin \phi & \cos \phi \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \\ {}_B^G\mathbf{a} &= \begin{bmatrix} \cos \psi \cos \theta & \cos \psi \sin \theta \sin \phi - \sin \psi \cos \theta & \cos \psi \sin \theta \cos \phi + \sin \psi \sin \phi \\ \sin \psi \cos \theta & \sin \psi \sin \theta \sin \phi + \cos \psi \cos \phi & \sin \psi \sin \theta \cos \phi - \cos \psi \sin \phi \\ -\sin \theta & \cos \theta \sin \phi & \cos \theta \cos \phi \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \end{aligned} \quad (\text{F.6})$$

Appendix G

Kalman Filter

G.1 Introduction

The Kalman Filter is a recursive algorithm introduced in the 1960's as a method to track, estimate, and predict the state of a system and corresponding uncertainties [24]. This filter integrates a dynamic (linear) model of the system, control inputs, measurements, and biases/uncertainties into a single algorithm. This effectively fuses together system inputs and responses and extrapolates what the system is currently doing and expected to do. One key advantage of this algorithm is that it only requires the guess of the previous state to estimate the current state. This massively decreases the memory and processing costs as the history of inputs, measurements, and uncertainties does not need to be remembered or analyzed. However, it does have some limitations when the sensor data is noisy or the control inputs cannot be linearly mapped to the system state. Random errors in the sensor data may cause the filter to behave unpredictably and non-linearity prevents proper fusion entirely. This problem can be solved using an Extended Kalman Filter, but that is out of scope for now.

G.2 Kalman Filter in One Dimension

The uni-dimensional Kalman filter is a special, idealized case for the Kalman filter. It is more convenient as a teaching tool as it does not include the complex matrix and vector operations the general multi-dimensional Kalman filter requires. However, this only makes this type useful for tracking and estimating a single variable.

Multiple uni-dimensional Kalman filters can be run in parallel or chained together to mimic a multi-dimensional filter, however, this will unnecessarily increase the computational resources required for the calculations and will reduce the quality of the filtering overall.

G.2.1 The Kalman Gain

In a Kalman filter, the Kalman Gain, denoted as K_n , is calculated at each iteration of the filter. The Kalman gain is bounded by: $0 \leq K_n \leq 1$ and prioritizes predictions of the system state from the estimation or the measurement, based on related uncertainties¹.

$$\begin{aligned} K_n &= \frac{\text{Estimate Uncertainty}}{\text{Estimate Uncertainty} + \text{Measurement Uncertainty}} \\ &= \frac{p_{n,n-1}}{p_{n,n-1} + r_n} \end{aligned} \tag{G.1}$$

We can then define the State Update Equation for a system where \hat{x} is the predicted state and z is the measured state:

$$\begin{aligned} \hat{x}_{n,n} &= \hat{x}_{n,n-1} + K_n(z_n - \hat{x}_{n,n-1}) \\ &= (1 - K_n)\hat{x}_{n,n-1} + K_n z_n \end{aligned} \tag{G.2}$$

¹**Important note:** When measurement uncertainty is very large, and the estimate uncertainty is small, $K_n \ll 1$, hence big weight to the estimate and small weight to the measurement. When the opposite is true, $K_n \gg 1$, meaning a large weight to the measurement and a small weight to the estimate. This is how the Kalman filter can regulate and smooth out noisy data by knowing the uncertainties.

G.2.2 The Estimate Uncertainty Update Equation

The estimate uncertainty or covariance update equation predicts the uncertainty associated with the current estimate. The estimate uncertainty should approach (converge) to 0 with each filter iteration as the filter improves its guessing accuracy. However, if the measurement uncertainty is large ($K_n \ll 1$), the estimate uncertainty will converge more slowly. The opposite is true if the measurement uncertainty is small. Basically, the more precise your measurements are, the faster the Kalman filter will converge on the best estimate.

$$p_{n,n} = (1 - K_n)p_{n,n-1} \quad (\text{G.3})$$

where $p_{n,n}$ is the estimate uncertainty at the current state

K_n is the Kalman gain at the current state

$p_{n,n-1}$ is the estimate uncertainty of the previous state

G.2.3 The Estimate Uncertainty Extrapolation Equation

Another Kalman equation is how the filter predicts future uncertainties and is called the Estimate Uncertainty Extrapolation Equation or the Estimate Covariance Extrapolation Equation. Like with the State Extrapolation Equations, this is done with dynamic models and will be unique to every example. For a moving vehicle, one example might be:

$$p_{n+1,n}^x = p_{n,n}^x + \Delta t p_{n,n}^{\dot{x}} \quad (\text{G.4})$$

$$p_{n+1,n}^{\dot{x}} = p_{n,n}^{\dot{x}} \quad (\text{G.5})$$

where x is the vehicle displacement, and

(G.6)

\dot{x} is the vehicle velocity

G.2.4 Putting it all Together

First, the filter is initialized with a first guess ($\hat{x}_{0,0}$) and an associated uncertainty ($p_{0,0}$). These values are passed into the dynamic model equations and Equation G.4 to predict the state at the first measurement ($x_{1,0}$) and the associated uncertainty ($p_{1,0}$). This is shown as Step 0 in Figure G.1. Then, we can take a measurement (z_n) and record its uncertainty (r_n), using the latter to determine the Kalman gain with Equation G.1. We can then estimate the current state value using Equation G.2 using the recorded measurement and the Kalman gain (K_n). The current state estimate and its uncertainty can then be outputted from the filter and used in applications; this process is shown as Steps 1, 2a, and 2b in the figure below. These values are then fed into the dynamic model to predict the next state and estimate the associated uncertainty (Step 3 in the below diagram). Optionally, these predicted values can be outputted from the model as well, as the application requires. This process repeats for every measurement with n incrementing every time.

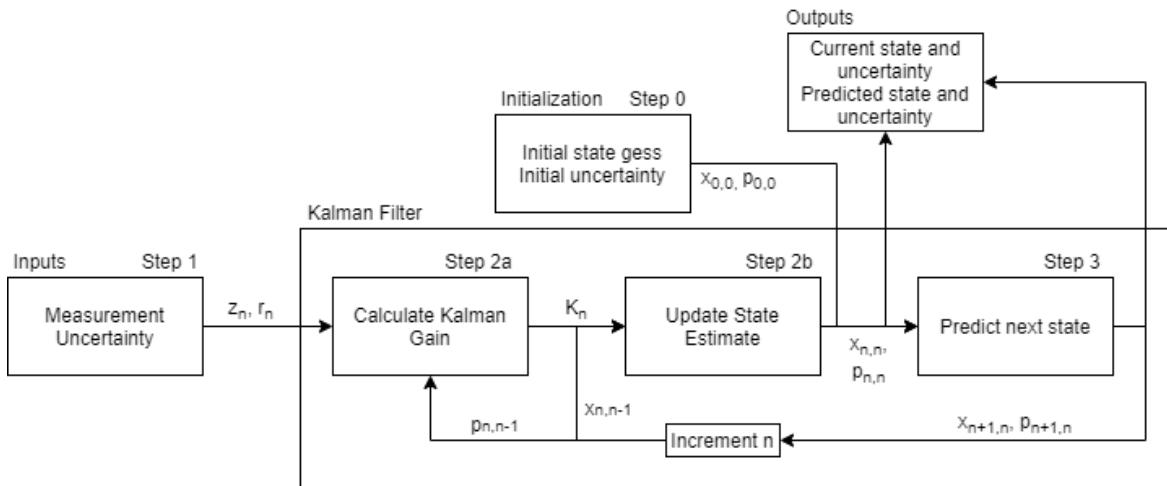


Figure G.1: Process diagram for a Kalman filter.

Example

In this example, we will be estimating the depth of an ROV using a 1D Kalman filter. We will be assuming the ROV depth remains constant and its location in space does not

matter or change. For example's sake, we shall know for an absolute fact that the ROV is at a depth of 50-meters ($x = 50$ m). We will also assume that our depth sensor has a standard deviation of 5-meters ($\sigma = 5$ m), therefore we will have a measurement variance or uncertainty of 25-meters ($r = 25$ m²)

Iteration 0

0 : Estimate depth: $\hat{x}_{0,0} = \mathbf{60}$ m

: Estimate uncertainty: $p_{0,0} = \mathbf{225}$ m²

3 : Predicted depth: $\hat{x}_{1,0} = \hat{x}_{0,0} = \mathbf{60}$ m

: Prediction uncertainty: $p_{1,0} = p_{0,0} = \mathbf{225}$ m²

Iteration 1

1 : Measure depth: $z_1 = \mathbf{48.54}$ m

: Measurement uncertainty: $r_1 = \mathbf{25}$ m²

2 : Kalman gain $K_1 = \frac{p_{1,0}}{p_{1,0} + r_1} = \frac{255}{255 + 25} = \mathbf{0.9}$

: Estimated depth: $\hat{x}_{1,1} = \hat{x}_{1,0} + K_1(z_1 - \hat{x}_{1,0}) = 60 + 0.9(48.54 - 60) = \mathbf{49.69}$ m

: Estimate uncertainty: $p_{1,1} = (1 - K_1)p_{1,0} = (1 - 0.9)225 = \mathbf{22.5}$ m²

3 : Predicted depth: $\hat{x}_{2,1} = \hat{x}_{1,1} = \mathbf{49.69}$ m

: Prediction uncertainty: $p_{1,0} = p_{1,1} = \mathbf{22.5}$ m²

Iteration 2

1 : Measure depth: $z_2 = \mathbf{47.11} \text{ m}$

: Measurement uncertainty: $r_2 = \mathbf{25} \text{ m}^2$

$$2 : \text{Kalman gain } K_2 = \frac{p_{2,1}}{p_{2,1} + r_2} = \frac{22.5}{22.5 + 25} = \mathbf{0.47}$$

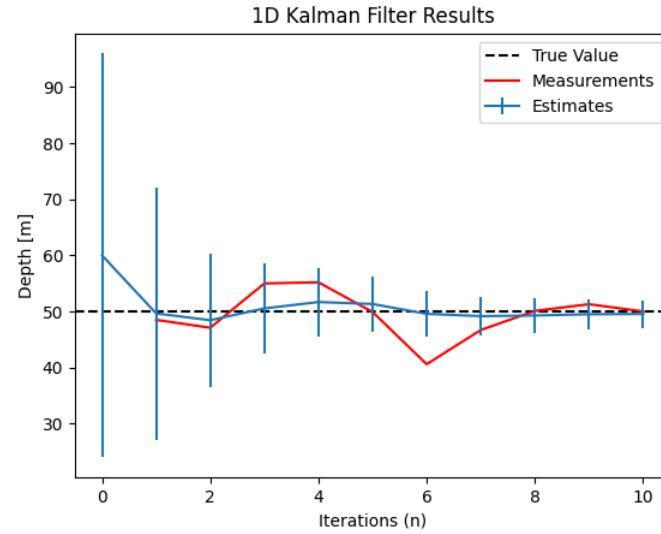
: Estimated depth: $\hat{x}_{2,2} = \hat{x}_{2,1} + K_2(z_2 - \hat{x}_{2,1}) = 49.69 + 0.47(47.11 - 49.69) = \mathbf{48.47} \text{ m}$

: Estimate uncertainty: $p_{2,2} = (1 - K_2)p_{2,1} = (1 - 0.47)22.5 = \mathbf{11.84} \text{ m}^2$

3 : Predicted depth: $\hat{x}_{3,2} = \hat{x}_{2,2} = \mathbf{48.47} \text{ m}$

: Prediction uncertainty: $p_{3,2} = p_{2,2} = \mathbf{11.84} \text{ m}^2$

n	Measurements		Current Estimates		Predictions		
	z_n	r_n	K_n	$\hat{x}_{n,n}$	$p_{n,n}$	$\hat{x}_{n+1,n}$	$p_{n+1,n}$
0	—	—	—	60.0	225	60.0	225
1	48.5	25	0.90	49.7	22.5	49.7	22.5
2	47.1	25	0.47	48.4	11.8	48.4	11.8
3	55.0	25	0.32	50.6	8.03	50.6	8.03
4	55.2	25	0.24	51.7	6.08	51.7	6.08
5	49.9	25	0.20	51.3	4.89	51.3	4.89
6	40.6	25	0.16	49.6	4.09	49.6	4.09
7	46.7	25	0.14	49.2	3.52	49.2	3.52
8	50.1	25	0.12	49.3	3.08	49.3	3.08
9	51.3	25	0.11	49.5	2.74	49.5	2.74
10	50.0	25	0.10	49.6	2.47	49.6	2.47



We can see from the figure that the Kalman filter eventually converges on the true value and considerably smoothes out the noisy measurements. The error bars on the estimate plot (blue) also show that the Kalman filter increases its confidence with every iteration as it converges to the true value. The oscillation shown in the estimates are a result of the gains being slightly off from their ideal value. Tuning the process noise variable (q) can cause the filter to converge quickly and confidently to the true value in fewer iterations.

G.2.5 Process Noise

In the real world, there are uncertainties in the system's dynamic model. Uncertainty is caused by unanticipated changes in the system due to external factors. This can be drift caused by ocean current, wind blowing a rocket to the side, drag, friction, even time dilation in extreme cases. Generally, these uncertainties can be combined into the Process Noise gain denoted by q . To account for process noise, it must be included in the Covariance Extrapolation Equation (Equation G.4).

If the model is not known to be good or is very noisy, we can increase the process noise gain to reduce the lag error.

$$p_{n+1,n} = p_{n,n} + q \quad (\text{G.7})$$

Further Reading

The book *Kalman Filter: From the Ground Up* [4] provides an excellent resource on the intuitive understanding of the Kalman Filter and some of its more advanced applications like multi-variate Kalman Filters, Extended Kalman Filters, and applying Kalman Filters in sensor fusion applications.

Appendix H

Traceability Matrix

A traceability matrix is an artifact within Systems Engineering that tracks the stakeholder requirements, capabilities, system requirements, and testing methodologies. The primary purpose is to give a Lead Systems Engineer other other team member quick access to all of the driving information for the product and trace the requirement lifecycle. Each matrix has eight columns that contain information about the requirement:

ID This is the identification number for the requirement. Stakeholder requirements will start with the prefix: “SR”. Capabilities will be triple-digit numbers with 100-level being “threshold”, 200-level being “reach”, and 300-level being “stretch” capabilities. Component-level requirements will be divided into sections and subsections in the format “X.X.X.X” depending on the section and depth of the requirement. For example, the first requirement in the first section, first subsection of the component-level matrix will have the ID “1.1.1”.

Description Each requirement and capability needs to have a brief, single sentence description that details the feature. The description should follow the guidelines laid out by NASA [38]. They should be unambiguous, testable, verifiable, and consistent in both terminology and can either be functional or non-functional. Functional requirements directly drive a feature of the product, whereas a non-functional requirement drives an aesthetic or

interface choice.

Weight The requirements and capabilities should all have a weight associated with them. These can be based on stakeholder preferences, developer capabilities, technology limitations, etc. The weight is a number greater than 0, but less than 1 and typically only expressed to two decimal places.

Priority Derived from the weight, the priority is the relative importance of a particular requirement or capability to others. It is calculated using Equation H.1. A higher priority means a requirement is more important to implement on the product. For capabilities, the priority is calculated slightly differently. The weight of a capability is compared to all of the other weights in the same category and the one(s) beneath it. For instance, a threshold capability's priority will only factor in the weights of other threshold capabilities. But, a stretch capability's priority will factor in the weights of all capabilities in the design.

$$P = \frac{w_c}{\sum w} \times 100 \quad (\text{H.1})$$

Owner The person who is responsible for a requirement or capability is called the owner. This individual is in charge of developing, implementing, verifying, testing, and validating the product feature. They do not have to be the only person working on that feature, but they are responsible for it. This individual should also be the one tracking and updating the requirement or capability status throughout the traceability matrix.

Verified? The verification of a requirement or capability requires an inspection to ensure that it is present. Component-level requirements will have a test number that should also be tracked in the traceability artifact to describe what test was performed that verified the requirement. Until a requirement or capability is successfully inspected, the answer to the column header is, “No”, but can be changed to “Yes” once verified. A requirement can be subject to a variety of methods to verify it is implemented:

Inspection: The nondestructive examination of a product or system using one or more of the five senses (visual, auditory, olfactory, tactile, taste). It may also include simple physical manipulation and measurements.

Demonstration: The manipulation of the product or system as it is intended to be used to verify that the results are as planned or expected.

Test: The verification of a product or system using a controlled and predefined series of inputs, data, or stimuli to ensure that the product or system will produce a very specific and predefined output as specified by the requirements.

Analysis: The verification of a product or system using models, calculations and testing equipment. Analysis allows someone to make predictive statements about the typical performance of a product or system based on the confirmed test results of a sample set or by combining the outcome of individual tests to conclude something new about the product or system. It is often used to predict the breaking point or failure of a product or system by using nondestructive tests to extrapolate the failure point.

Validated? Validation of a capability or requirement occurs when the feature is fully integrated to the system and tested. It is not enough that it is present, rather it must also contribute to the system as designed without impinging on another feature. This is another “Yes” or “No” column depending on if the feature has been fully integrated to the system.

Status The last major column to consider in the matrix is the status column. This lets the reader know if the capability or requirement has been reviewed or rejected, is undergoing testing, delivered, etc. The value of this column is pulled from Table H.1 based on the requirement’s stage in the lifecycle (see Figure H.1).

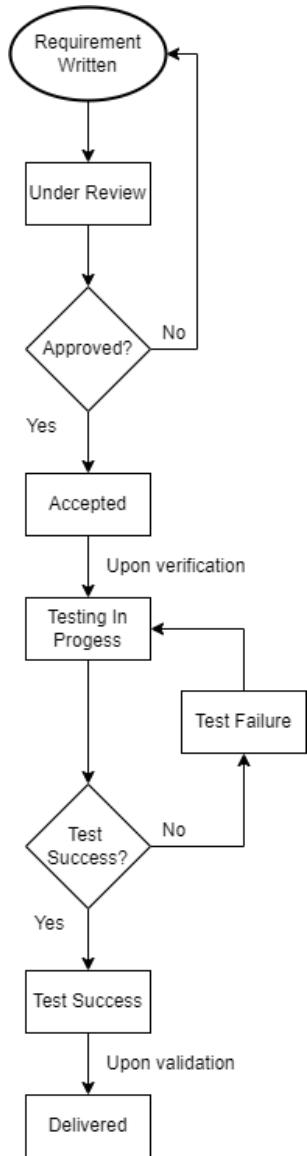
Table H.1: Description of requirement statuses

UR	Under Review	Requirement has been submitted and is under review
A	Accepted	Requirement has been reviewed and accepted into the artifact
R	Rejected	Requirement has been reviewed and rejected from the artifact
TIP	Testing in Progress	Requirement is currently being tested
TS	Test Successful	Requirement has been successfully tested, but not yet delivered
TF	Test Failure	Requirement failed testing and needs further development
D	Delivered	Requirement has been fully realized and delivered

H.1 Requirement Lifecycle

Requirements are initially created at the start of the project and are expected to be monitored and have their status updated as the project continues. Once a requirement is written, it is up the Lead Systems Engineer, or other team members, to review the requirement and determine if it is suitable for the project. The major considerations are 1) “is the requirement relevant?”, 2) “is the requirement unambiguous?”, 3) “is the requirement testable?”, and 4) “is the requirement complete?” If the requirement does not meet any of these criteria, it can be rejected for re-writing by its owner. Once the requirement is accepted, development for it can begin. It is up to the owner to ensure that the feature detailed by the requirement is properly designed and integrated into the larger system. When the owner verifies the feature and is ready to validate it with the system, they can begin the testing process and update the status to “TIP” in the traceability matrix. If the test was successful, the requirement status should reflect this; same for a test failure. If the validation fails, the requirement may need to be re-written or re-tested in a different system configuration. Finally, when the feature is fully integrated with the product and released, the status can be set to “delivered” (D) and the feature can be considered fully completed.

Figure H.1: Lifecycle of a requirement



Appendix I

Wave Estimating Algorithm for Vessels Experiment (WEAVE)

I.1 Introduction

As a buoy floats on top of waves, it experiences an acceleration proportional to the hydrodynamic forces exerted on it by $\mathbf{F} = m\mathbf{a}$. As a result, the buoy is displaced by a measurable amount as a wave passes through it. A wave buoy uses an accelerometer to measure the vertical acceleration component of this motion. By double integrating the signal with respect to time via the following equation, we can get the vertical displacement $z(t)$.

$$z(t) = \frac{1}{2}a_z(t)\Delta t^2 + e \quad (\text{I.1})$$

Where $a_z(t)$ is the vertical acceleration time series, Δt is the time since the last value update, and e is the measurement error introduced by the sensor errors and integration errors (more on this later).

The equation shows that the error is multiplied over time, causing the reading to drift. In-

dustry standard buoys like the DataWell WaveRider 4¹ use high precision, regularly-calibrated sensors to minimize and account for the measurement error.

These types of uni-axial measurement buoys also rely on the measurement axis always being parallel to the Earth vertical axis. While the WaveRider 4 buoy accomplishes this by suspending the accelerometer on a platform in a fluid specifically tuned to act as a massive dampener, this comes at an increase to size and cost. Additionally, should this platform become misaligned to the global vertical axis, the measurements will always be of a component of the vertical acceleration and therefore introduce a bias in the calculations.

In this document, a procedure to use body frame coordinate rotation will be introduced that will eliminate the error and bias of a misaligned vertical axis measurement. The procedure will be implemented in two different ways: 1) a spectral analysis using the procedure laid out in Longuet-Higgins [28] and used by the WaveRider 4 buoy, 2) a novel oscillatory motion analysis based upon the work of Madgwick [31] that provides an instantaneous estimate of the wave height and direction.

I.2 Background

Inertial Measurement Units (IMUs) consist of an array of sensors that measure inertial forces acting on a body. Each measurement axis of the IMU is referred to as a degree of freedom (DOF). Advanced IMUs utilize 9 DOFs of measurements and also called Magnetic, Angular Rate, and Gravity (MARG) arrays. These sensors use a tri-axial magnetometer, gyroscope, and accelerometer, respectively to determine a body's pose or orientation in 3D space, relative to a magnetic and gravitational field.

Using sensor fusion algorithms, it is possible to fuse the MARG array data into a unified representation of the body pose called the Attitude and Heading Reference System (AHRS). There are several such algorithms, most notably the complementary filter, the Kalman Filter,

¹<https://datawell.nl/products/directional-waverider-4/>

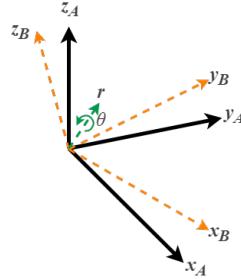
the Mahony filter, and the Madgwick filter. Madgwick's filter is ideal for embedded applications because of its high accuracy, low performance cost, and ability to incorporate gyroscopic and magnetic drift compensation during runtime. This makes the attitude estimate over accurate over time and across a larger range of body motions. Therefore, this algorithm will be used exclusively during this study for sensor fusion.

I.2.1 Quaternion Representation

The crux of Madgwick's filter is a gradient descent estimation of the body quaternion. The quaternion is a 4-dimensional representation of orientation that is free from infinite discontinuities found in 3-dimensional counterpart, Euler angles. The quaternion represents a 3D coordinate frame's rotation about a unit vector in the base coordinate frame. This is represented graphically in the Figure I.1 where the mutually orthogonal axes, A_x , A_y , A_z , B_x , B_y , B_z represent the 3D frames A and B , respectively. When frame B is rotated by some angle, θ , about the unit vector, $^A\mathbf{r}$, it now has an orientation relative to frame A that can be expressed as the quaternion:

$${}^B\mathbf{q} = \begin{bmatrix} q_1 & q_2 & q_3 & q_4 \end{bmatrix} = \left[\cos \frac{\theta}{2} \quad -r_x \sin \frac{\theta}{2} \quad -r_y \sin \frac{\theta}{2} \quad -r_z \sin \frac{\theta}{2} \right] \quad (\text{I.2})$$

Figure I.1: Illustration of the rotation from a base coordinate frame to a local coordinate frame about the axis, \mathbf{r}



Note that we adopt the notation introduced by Craig [8]. The preceding superscript denotes the “base” or “from” coordinate frame and the preceding subscript denotes the “derived” or “to” coordinate frame. In the previous example, Bq is the quaternion representing

the rotation *from* frame A *to* frame B .

I.2.2 Quaternion Application to Rotations

If, by convention, we assume all bodies that are not Earth have some rotation relative to Earth's central coordinate frame, then we can represent any body's orientation as ${}^B_G \mathbf{q}$ where G is the global frame and B is the body frame. If we also assume that the body is within Earth's gravitational and magnetic field, then we can use a MARG array to determine ${}^B_G \mathbf{q}$ via Madgwick's filter.

Since quaternions represent coordinate frame rotations, we can use them to rotate 3D vectors from one frame to another. For example, in any given orientation in Earth's gravitational field, there will always be components of gravitational acceleration within a body frame's acceleration. Therefore, an accelerometer in the body frame will always be measuring the gravitational acceleration vector, ${}^G \mathbf{a}$, and the body acceleration vector, ${}^B \mathbf{a}$. If we want to eliminate the gravitational signal, we can rotate the gravitational acceleration vector from the global coordinate frame to the body frame and subtract it from our readings. This will yield the body linear accelerations, ${}^B \mathbf{a}_l$, defined as:

$${}^G_B \mathbf{a} = {}^G_B \mathbf{q} \cdot {}^G \mathbf{a} \cdot {}^G_B \mathbf{q}^{-1} \quad (\text{I.3})$$

$${}^B \mathbf{a}_l = {}^B \mathbf{a} - {}^G_B \mathbf{a} \quad (\text{I.4})$$

Again, using the notation from Craig to denote the "from" and "to" frames, respectively. Also note that ${}^G_B \mathbf{q}^{-1}$ is the inverse quaternion as defined by:

$${}^G_B \mathbf{q}^{-1} = \begin{bmatrix} -q_1 & -q_2 & -q_3 & -q_4 \end{bmatrix}$$

Alternatively, we can rotate the body accelerations to the global frame and get global linear accelerations, ${}^G \mathbf{a}_l$, via a similar operation:

$${}^B_G \mathbf{a} = {}^B_G \mathbf{q} \cdot {}^B \mathbf{a} \cdot {}^B_G \mathbf{q}^{-1} \quad (\text{I.5})$$

$${}^B_G \mathbf{a}_l = {}^B_G \mathbf{a} - {}^G \mathbf{a} \quad (\text{I.6})$$

By convention, the global coordinate frame is typically referred to as North-East-Down (NED) and therefore, ${}^G \mathbf{a} = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}$

I.3 Spectral Analysis Theory

For the spectral analysis, the work by Longuet-Higgins will be heavily utilized. The displacement vector should be subsampled to 2.56 Hz. Over a 200-second interval, a total of $N = 512$ samples are collected. We can then obtain a frequency spectrum via the Fast Fourier Transform (FFT). The spectrum will have a range of 0 to $\frac{f_s}{2} = 1.28\text{Hz}$ and a resolution of $\frac{f_s}{N} = 0.005\text{Hz}$. The FFT yields Fourier coefficients via:

$$\sum_{k=0}^{N-1} w_k h_k e^{\frac{2\pi k l}{N}} \quad (\text{I.7})$$

Where $f_l = \frac{1}{N\Delta t}$ and $l = 0 \dots N-1$ and the w_k element is a windowing function. Datawell uses a Hann windowing function [6] which attenuates signals in the higher frequency domains:

$$w_k = \frac{1}{2} [1 - \cos \frac{2\pi k}{N}] \text{ where } k = 0 \dots N-1 \quad (\text{I.8})$$

We can normalize these coefficients via:

$$w_{norm} = \sqrt{f_s \sum_{k=0}^{N-1} w_k^2} \quad (\text{I.9})$$

Since we have time series data for all three axes, we can take the spectra for each series. Each series contains a vector of coefficients and can be expressed in vector notation as:

$$a_N = \alpha_N + i\beta_N \quad (\text{I.10})$$

$$a_E = \alpha_E + i\beta_E \quad (\text{I.11})$$

$$a_D = \alpha_D + i\beta_D \quad (\text{I.12})$$

We can then define the co-spectra (C) and the quad-spectra (Q) as follows:

$$C_{xy} = a_x \cdot a_y = \alpha_x \alpha_y + \beta_x \beta_y \quad (\text{I.13})$$

$$C_{xy} = a_x \times a_y = \alpha_x \beta_y - \beta_x \alpha_y \quad (\text{I.14})$$

Then, we can arrange them into 3 by 3 matrices to obtain the final definition of the acceleration spectra:

$$\begin{bmatrix} C_{NN} & C_{NE} & C_{ND} \\ C_{EN} & C_{EE} & C_{ED} \\ C_{DN} & C_{DE} & C_{DD} \end{bmatrix} \quad (\text{I.15})$$

$$\begin{bmatrix} 0 & 0 & Q_{ED} \\ 0 & 0 & Q_{ND} \\ Q_{DE} & Q_{DN} & 0 \end{bmatrix} \quad (\text{I.16})$$

Note that by definition, $Q_{NN} = Q_{EE} = Q_{DD} = 0$ and Q_{EN} and Q_{NE} represent eddy currents and are not considered.

These spectra can represent different wave parameters such as wave direction, direction spread, wave ellipticity, and power spectral density (PSD). The wave direction, θ_0 , is defined as the angle from which the waves are approaching relative to geographic north.

$$\theta_0 = \arctan \frac{-Q_{ED}}{Q_{ND}} \quad (\text{I.17})$$

The direction spread can be more difficultly determined via:

$$S = \sqrt{2 - 2m_1} \quad (\text{I.18})$$

$$\begin{aligned} \text{where } m_1 &= \sqrt{a_1^2 + b_1^2} \\ a_1 &= \frac{Q_{ND}}{\sqrt{(C_{NN} + C_{EE})C_{DD}}} \\ b_1 &= \frac{-Q_{ED}}{\sqrt{(C_{NN} + C_{EE})C_{DD}}} \end{aligned} \quad (\text{I.19})$$

The wave ellipticity describes the shape of the wave. Specifically, the particle trajectories. For wavelengths much smaller than the water depth, the particles will follow a circular trajectory, thus the ellipticity is near 1. However, as the wavelength approaches the water depth, the vertical displacements reduce and the ellipticity becomes less than 1, trending towards 0. We can determine this parameter via:

$$\epsilon = \frac{1}{k} = \sqrt{\frac{C_{DD}}{C_{NN} + C_{EE}}} \quad (\text{I.20})$$

Finally, the PSD describes the amount of energy present in the signal across the entire frequency spectrum. The PSD for the vertical axis is considered the wave spectrum and is calculated by:

$$E(f) = C_{DD} = \alpha_D^2 + \beta_D^2 \quad (\text{I.21})$$

By taking the integral of the PSD we can find the n-th spectral moment of the signal. The zeroth spectral moment, m_0 , can tell us a lot of information such as the significant wave height, H_s , and the peak wave period, T_p .

$$m_0 = \int_0^\infty f^0 E(f) df \quad (\text{I.22})$$

$$H_s = 4\sqrt{m_0} = H_{m_0} \quad (\text{I.23})$$

$$T_p = 5\sqrt{H_s} \quad (\text{I.24})$$

I.4 Oscillitory Motion Tracking Theory

The spectral analysis method is the industry standard and recommended by NOAA [20]. However, the spectra needs to be analyzed over time. The accuracy of the spectral analysis directly correlates with its sample interval; instantaneous intervals will never be completely representative whereas infinite intervals will be 100% accurate. The above methodology offers a balance of accuracy and reporting time. But, the sample interval of 200-seconds may be too long for some applications. Therefore, in this section we will explore an alternative approach where the wave height is directly computed instantaneously using accelerometer integration and feedback error correction.

We start by using the rotated linear accelerations derived in the Section I.1. These can be integrated over time to get velocity, \mathbf{v} , as shown below:

$${}^G\mathbf{v}(t) = {}^G\mathbf{a}(t)\Delta t \quad (\text{I.25})$$

The integration error accumulates over time and causes the signal to drift, or deviate from the true value. We can consider this drift a velocity signal with infinite period and apply a high-pass filter with an extremely small cutoff frequency to attenuate the error. The filter design is discussed in an upcoming subsection. We can perform another integration to get the buoy displacement in the global frame:

$${}^G\mathbf{s}(t) = {}^G\mathbf{v}(t)\Delta t + \mathbf{e} = {}^G\mathbf{a}(t)\Delta t^2 + \mathbf{e} \quad (\text{I.26})$$

The integration error for displacement is also fed by the error from velocity, so this value will drift faster over time. Again, we can consider the drift an infinite period signal and apply another high-pass filter. Madgwick, 2013 shows this method is effective for motions where a zero-velocity point occurs (e.g. at the bottom of a step). But, this method needs to be tested on a constantly moving object, like a wave buoy.

I.4.1 Filter Design

The goals of the velocity and displacement filters are to maximize the attenuation of the error signal while minimizing both the attenuation of the desired signal and the induced phase shift. The ideal filter will have these criteria, but they are impossible to design. Therefore, we must compromise between the two.

There are four main digital infinite impulse response filters: Bessel, Butterworth, Chebyshev (Type I and Type II), and Elliptic. Each filter can be expressed by its “order” or how many times the filter is applied to the signal. The higher the order, the steeper the transition and the closer the magnitude response becomes to ideal. But, this often comes at the cost of dramatically increasing phase shift. The main filters can generally be ranked in the order given above: from left to right, the transition becomes steeper with a smaller order; from right to left, the phase shift is more linear in the pass band.

If a more linear/minimized phase shift is desired, then a low order Bessel or Butterworth filter is best. Though, their transition areas can span many frequency decades. Since our cutoff frequency is extremely small, we want the most reactive filter with the narrowest transition. To make the Bessel and Butterworth filter fulfill this criterion, we need to substantially increase their orders, eliminating their beneficial phase shift characteristic.

The two remaining filters, Chebyshev and Elliptic, have sharper transition regions but at the cost of non-linear phase shifts and ripple in the pass and stop bands. For a bandpass filter design, we can define a couple of parameters: a low-stop frequency of 0.025 Hz, a low-pass frequency of 0.05 Hz, a high-stop frequency of 2.0 Hz, a high-pass frequency of 0.25 Hz, a stop attenuation of 100 dB, and a pass attenuation of 0.1 dB. With these values, different

bode plots for Chebyshev Type I (Figure I.2a), Chebyshev Type II (Figure I.2b), and Elliptic (Figure I.2c) filters were created.

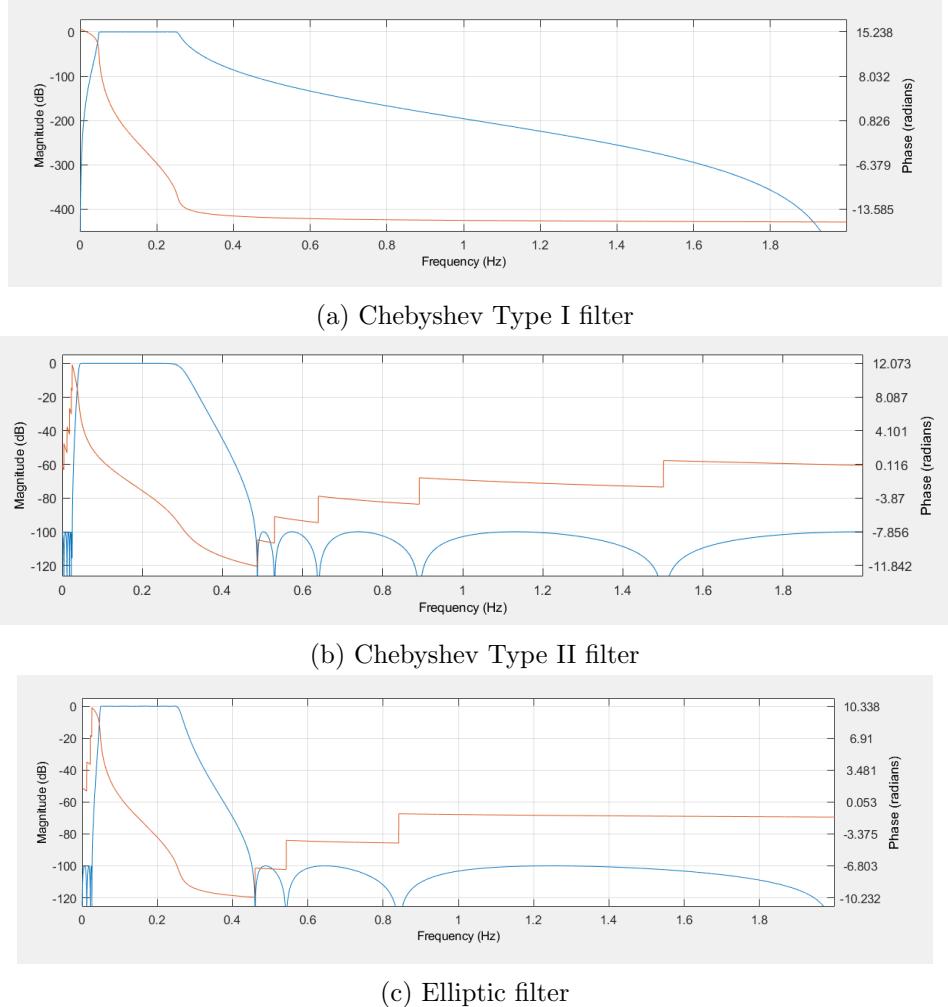


Figure I.2: Bode plots for Chebyshev and Elliptical filters using a low-stop frequency of 0.025 Hz, a low-pass frequency of 0.05 Hz, a high-stop frequency of 2.0 Hz, a high-pass frequency of 0.25 Hz, a stop attenuation of 100 dB, and a pass attenuation of 0.1 dB.

From these plots, we can see that both the Chebyshev Type II and Elliptic filter have undesired ripple on the stop bands and the phase shift is very non-linear. The Chebyshev Type I filter on the other hand has a much sharper attenuation of the stop bands without any ripple at the cost of some more phase shift. We will therefore be using the Chebyshev Type I filter for our analysis.

I.5 Methodology

The two wave analysis techniques will be performed in post-processing after a data collection period. Data will be collected using the Thetis inertial datalogger and the x-IMU3 datalogger [46]. The devices will be rigidly attached to the top of a Datawell WaveRider4 buoy and will be accepted as the ground truth for the experiment and used to validate the spectral analysis procedure.

The devices will be deployed for a total of six hours on five different days since the batteries have to be recharged after that length of deployment. Data will be offloaded and stored onto a computer after recovery. Both Thetis and the x-IMU3 will supply data at 64 Hz meaning each batch of data contains 1.4 million points for a total of 6.9 million data points. Using the Madgwick filter, the data points will be used to compute the buoy orientation and linear acceleration over time.

I.5.1 For Spectral Analysis

The linear acceleration dataset will be subsampled from 64 Hz to about 2.56 Hz using a rolling average filer of $N = 25$ samples. This will eliminate most of the high frequency noise while still preserving the wave motion signal. From here, the data analysis procedure from the previous section will be used to calculate the wave statistics, averaged over each hour of runtime. These can be directly compared to the data pulled from the wave buoy using a Root Mean Square Error (*RMSE*) as shown below:

$$RMSE_H = \sqrt{\frac{\sum (H_{sX} - H_{sB})^2}{N}} \quad (\text{I.27})$$

Where H_{sX} is the significant wave height calculated by the inertial data loggers, H_{sB} is the significant wave height reported by the buoy, and N is the number of samples.

$$RMSE_T = \sqrt{\frac{\sum (T_{pX} - T_{pB})^2}{N}} \quad (\text{I.28})$$

Where T_{pX} is the significant wave height calculated by the inertial data loggers, T_{pB} is

the significant wave height reported by the buoy, and N is the number of samples.

Note that due to the limited sample time, the comparisons are only valid for a small wave regime. The results section will detail the validity more in-depth, but for a more accurate analysis, these instruments must be deployed for longer under a wider array of waves.

I.5.2 For the Oscillatory Motion Tracking Analysis

The data will again be subsampled to about 4 Hz using a rolling average filer of $N = 16$. As before, this eliminates high frequency noise while preserving the much lower frequency wave motion signals (this can help prevent drift throughout the analysis).

We can take the first integral of our linear acceleration data which yields the buoy's linear velocities plus errors in the global frame. These can be filtered out by applying a bandpass Chebyshev Type I filter to the data. By observation of the Rayleigh Distribution for waves, we can assess that a majority of wave-related motion will occur in the 4- to 20-second range. We can therefore tailor the filter such that we have the sharpest cut-off around the range of our expected signal frequency while maximizing the stopband attenuation.

After applying the filter to the velocity data, we can take the integral again to yield the buoy displacement in the global frame. The same bandpass filter as used before can be applied to this new time series which will give us the best estimate of the buoy's displacement. The vertical displacement series represents the sea surface elevation time series, η .

From the displacement, we can perform a wave-by-wave analysis which uses the zero-upcrossing method to identify discrete windows of a wave. The psuedocode for the algorithm is given below. The analysis of all these windows yields wave statistics that we can compare between the inertial data loggers and the spectral analysis method.

This method is not flawless as statistically insignificant up-crossings called, false peaks, will be detected and recorded. We can filter out the false peaks by following the methodology laid out in Coats [41]. We can compute the root mean square of the wave heights using the equation below. Then, we can filter out all of the waves which height fall below the arbitrary threshold $0.1H_{\text{rms}}$.

Algorithm 1 Wave By Wave

```
1: function WAVEBYWAVE( $\eta, f_s, g$ )
2:   index  $\leftarrow 1$ 
3:   negIndices  $\leftarrow$  find all negative indices in  $\eta$             $\triangleright$  negIndices is a list of indices
4:   for n=1, ..., size(negIndices) do
5:     if  $\eta[\text{negIndices}[n]+1] > 0$  then
6:       upcrossTimes[index]  $\leftarrow$  negIndices[n]
7:       index++
8:   for u=1, ..., size(upcrossTimes)-1 do
9:      $\eta_u \leftarrow \eta[\text{upcrossTimes}[u]:\text{upcrossTimes}[u+1]]$ 
10:    Calculate  $H_u$ ,  $T_u$ , and others for  $\eta_u$ 
11:    Put wave parameters into wave structure
12:    Append the new wave structure to waves
13:   Return waves
```

$$H_{\text{rms}} = \sqrt{\frac{1}{N} \sum H_i^2} \quad (\text{I.29})$$

Additionally, we can remove false peaks by examining the time between them. From Rayleigh [9], we can be confident that the wave period will not be faster than 4-seconds, nor slower than 20-seconds. Therefore, any peaks that occur within 4-seconds of or 20-seconds after another can be discarded. After we remove all the false peaks, the array of waves must be sorted in descending order according to their height.

From the wave-by-wave analysis, we can determine the significant wave height by calculating the highest one third of wave heights using:

$$H_s = H_{\frac{1}{3}} = \frac{3}{N} \sum_{i=\frac{2}{3}N}^N H_i \quad (\text{I.30})$$

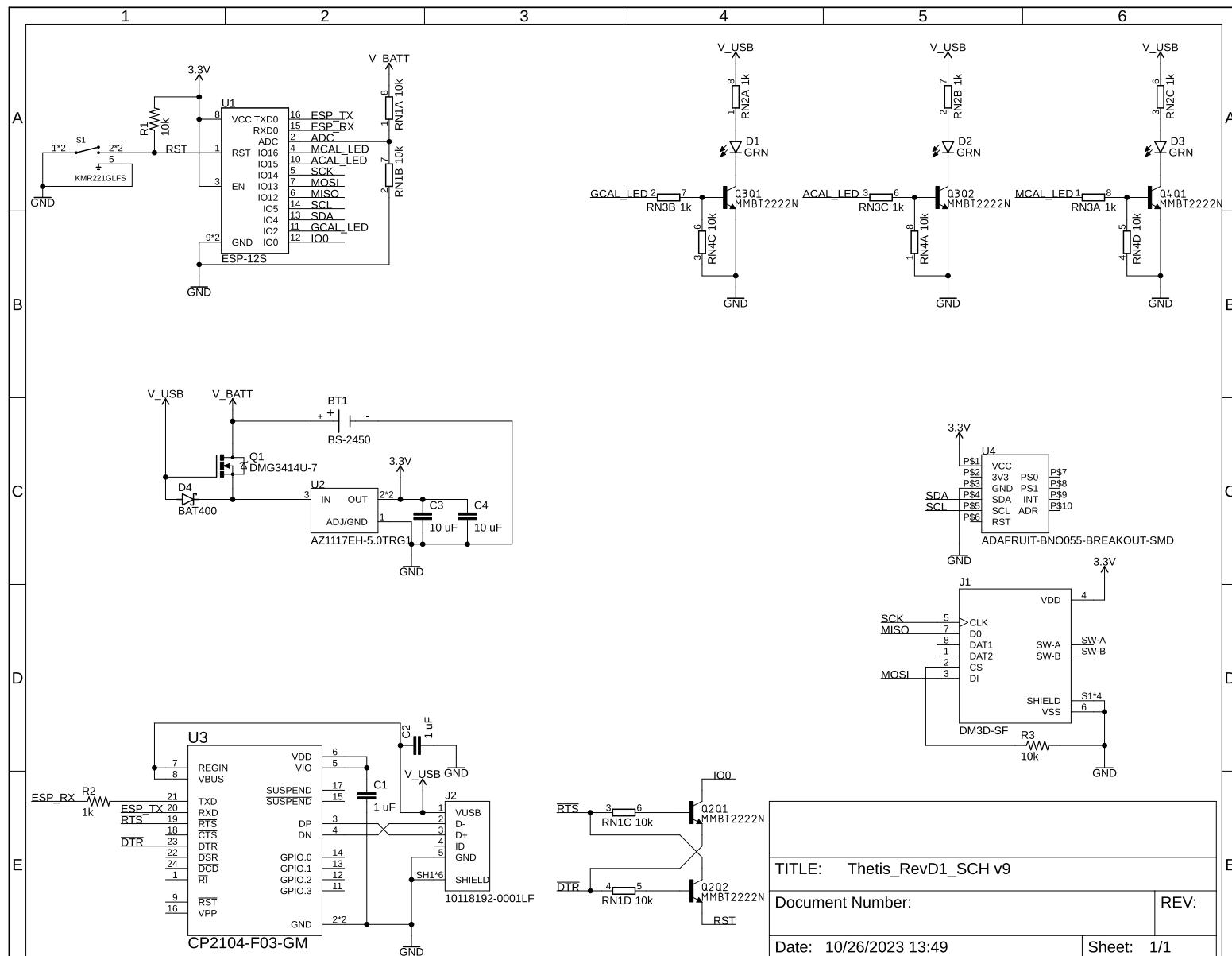
If we want to directly compare to the previous methods, we need to window our waves in the same time periods as the spectral analysis. We can compute the significant wave height for that window using the above equation and determine the peak period by creating a distribution of periods (that is, time between upcrossings) in that window; the peak wave period would be that with the most counts in its bin. As before, we can compare the inertial data loggers to each other via per cent difference, and to the wave buoy via the RMSE

method.

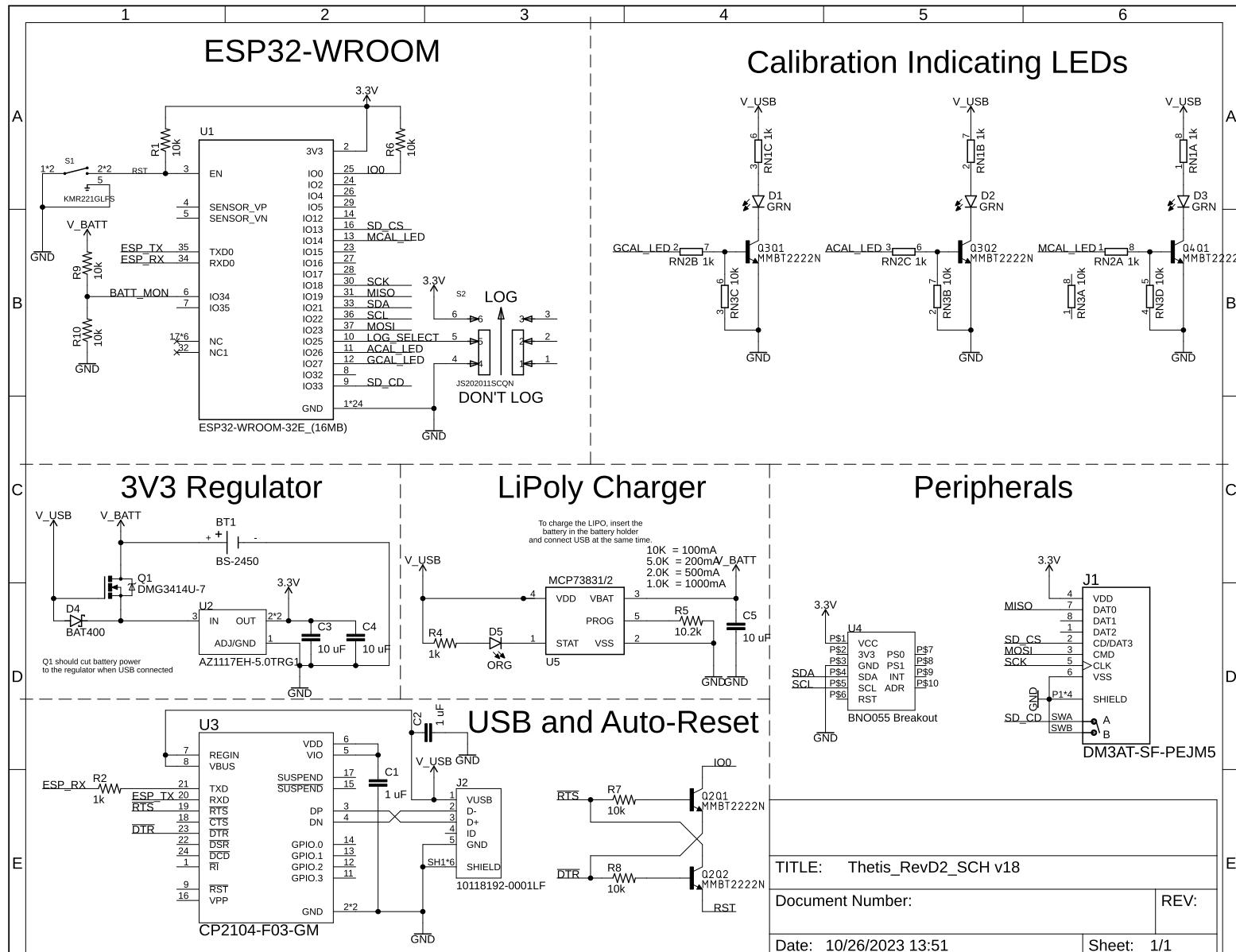
Appendix J

Revision Schematics

J.1 Revision D1 Schematic

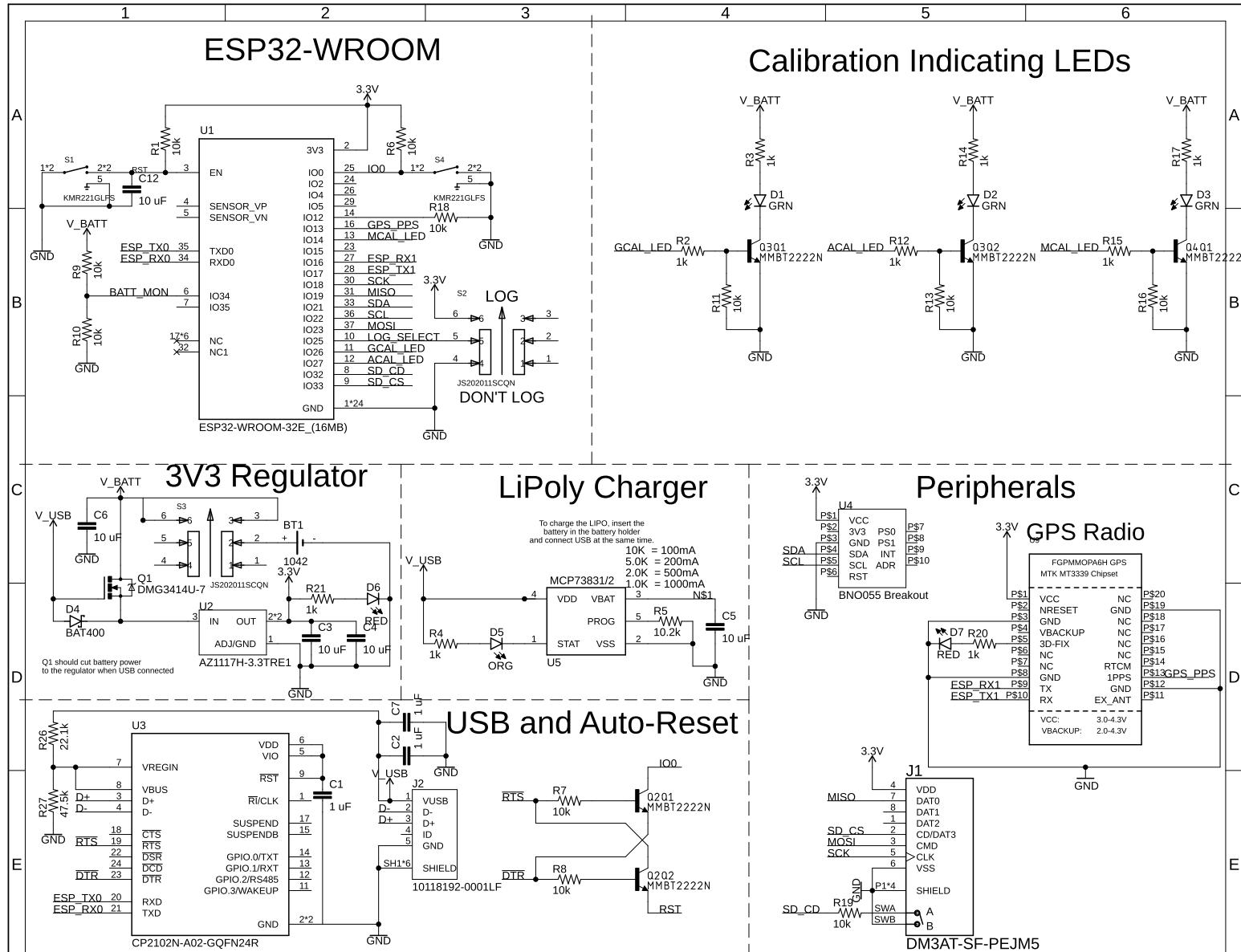


J.2 Revision D2 Schematic

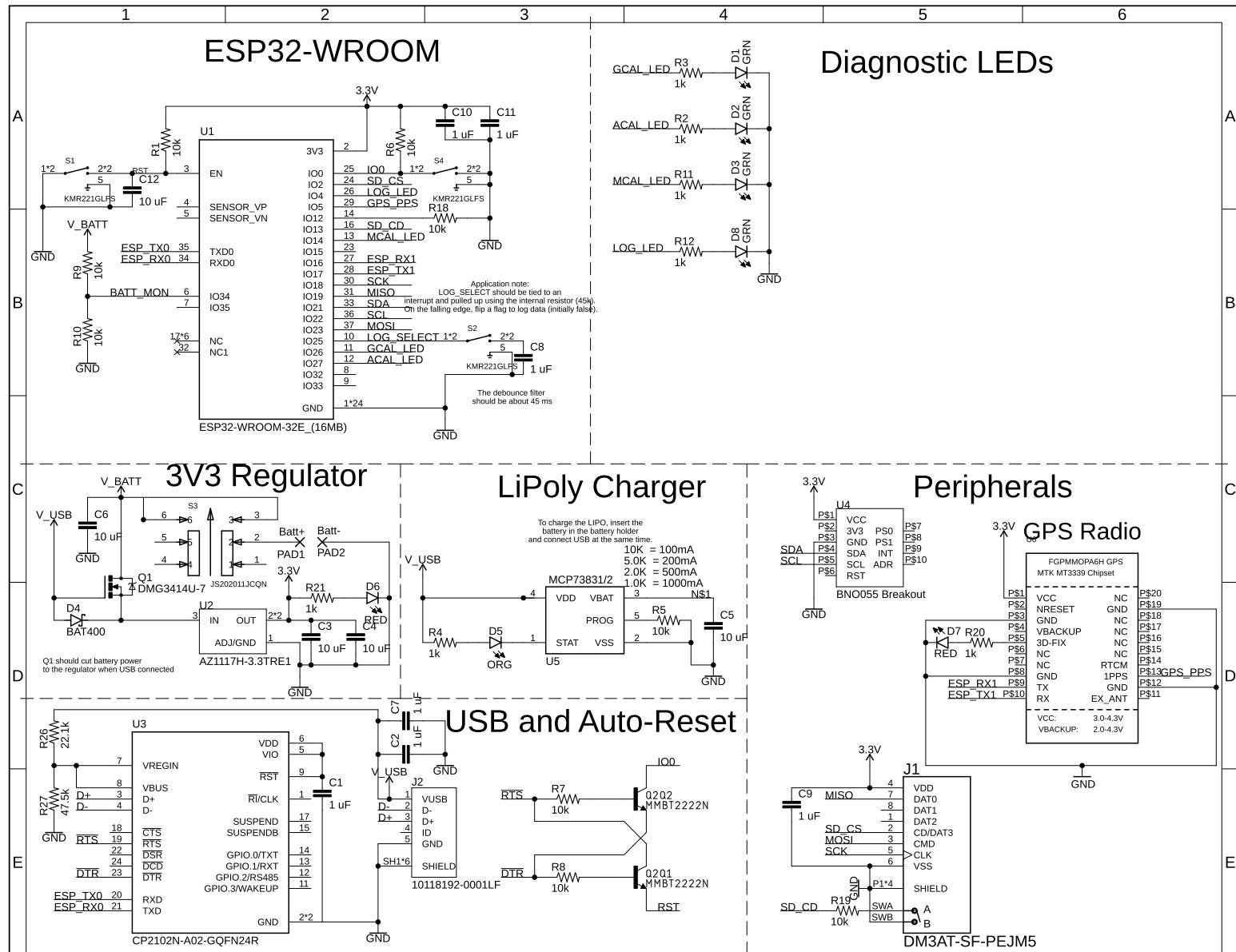


J.3 Revision E1 Schematic

181

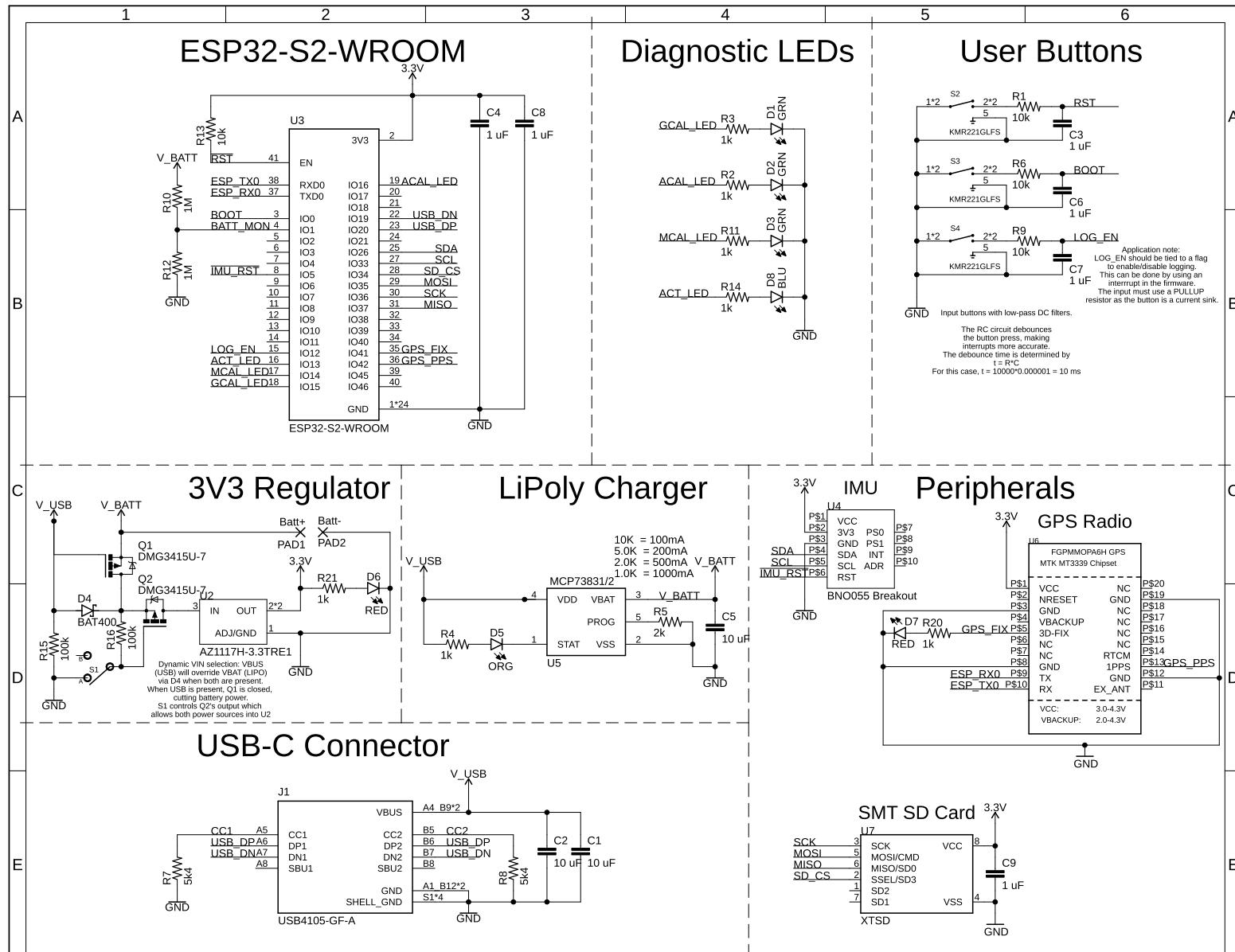


J.4 Revision F1 Schematic



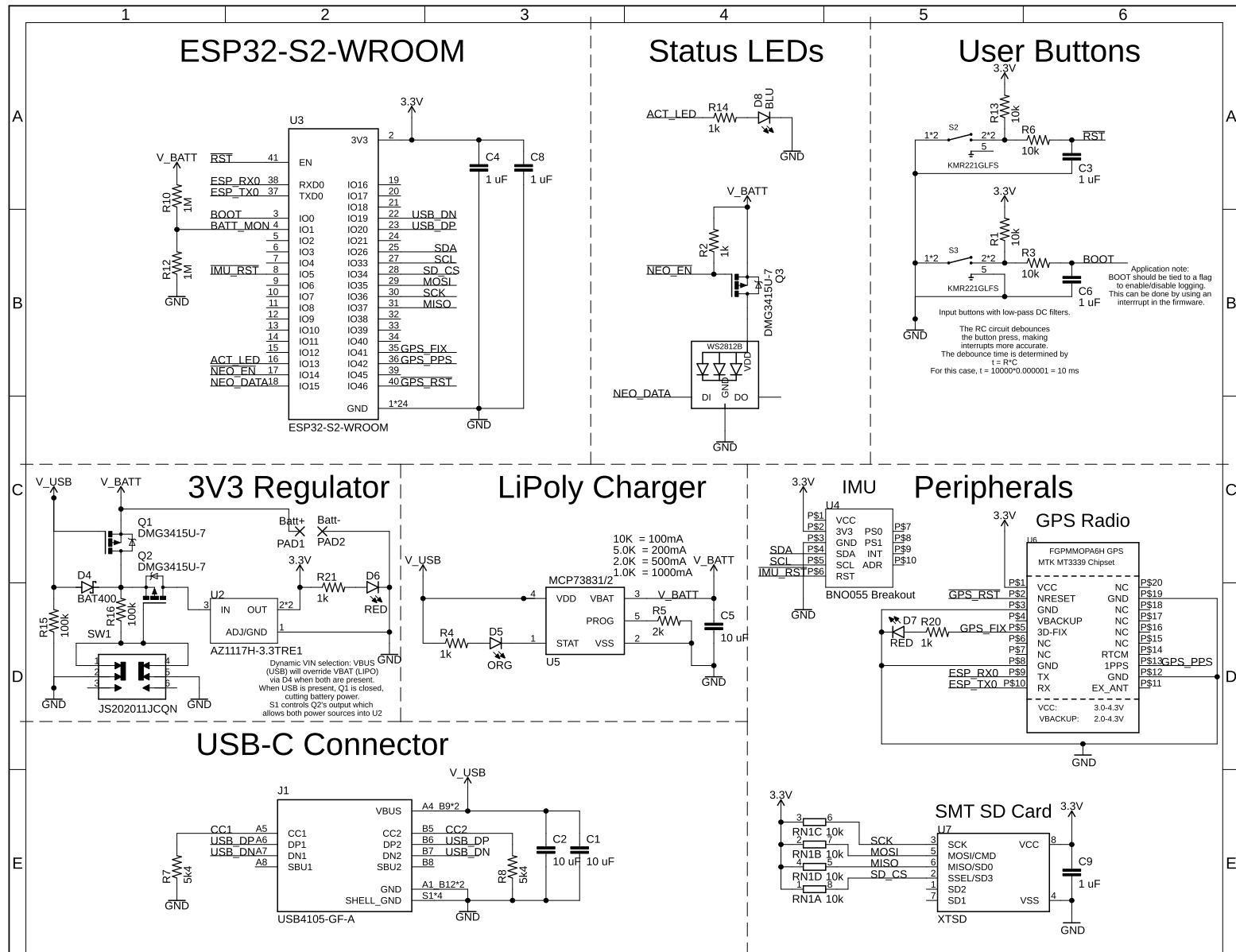
J.5 Revision F2 Schematic

183

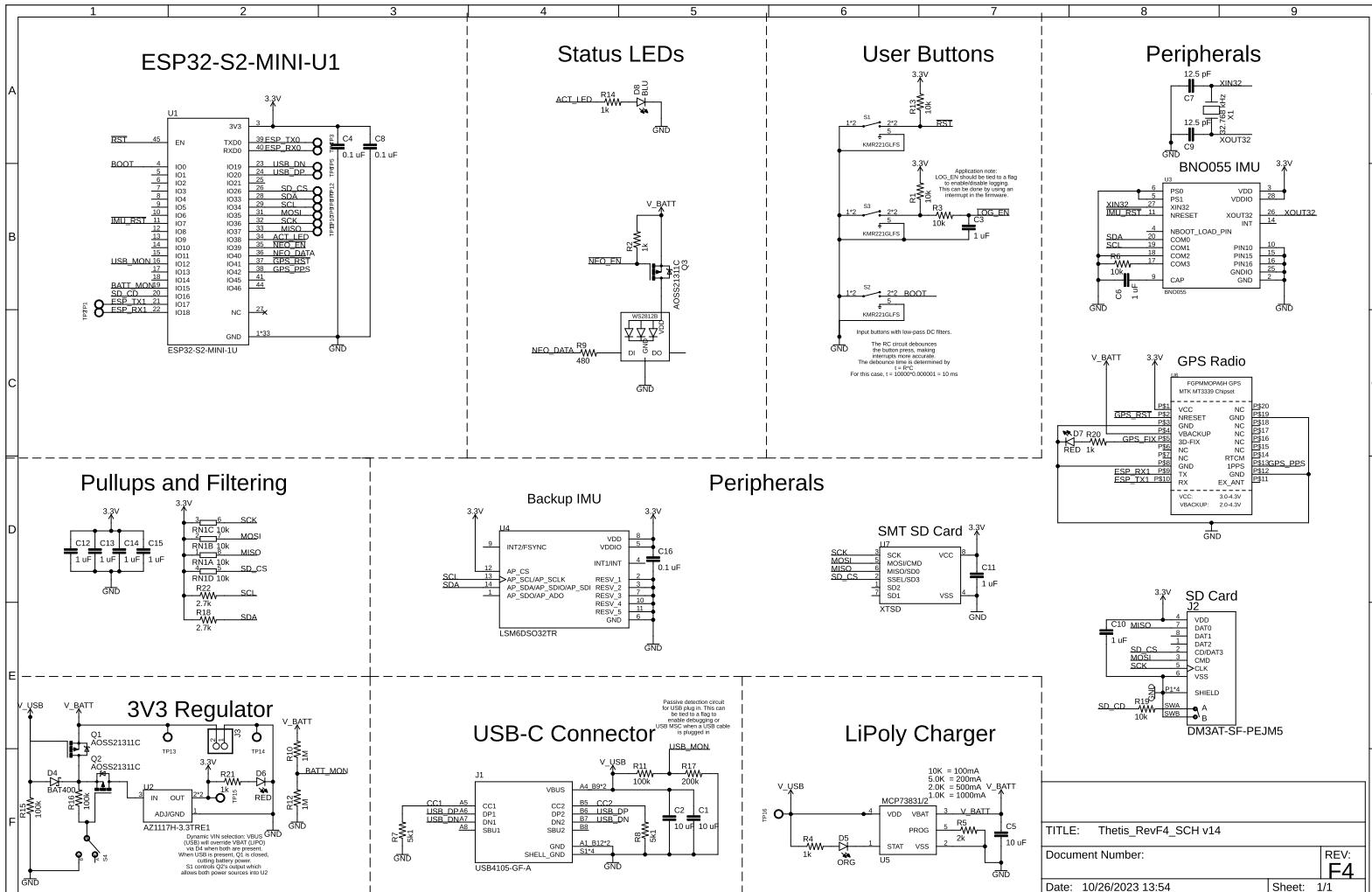


J.6 Revision F3 Schematic

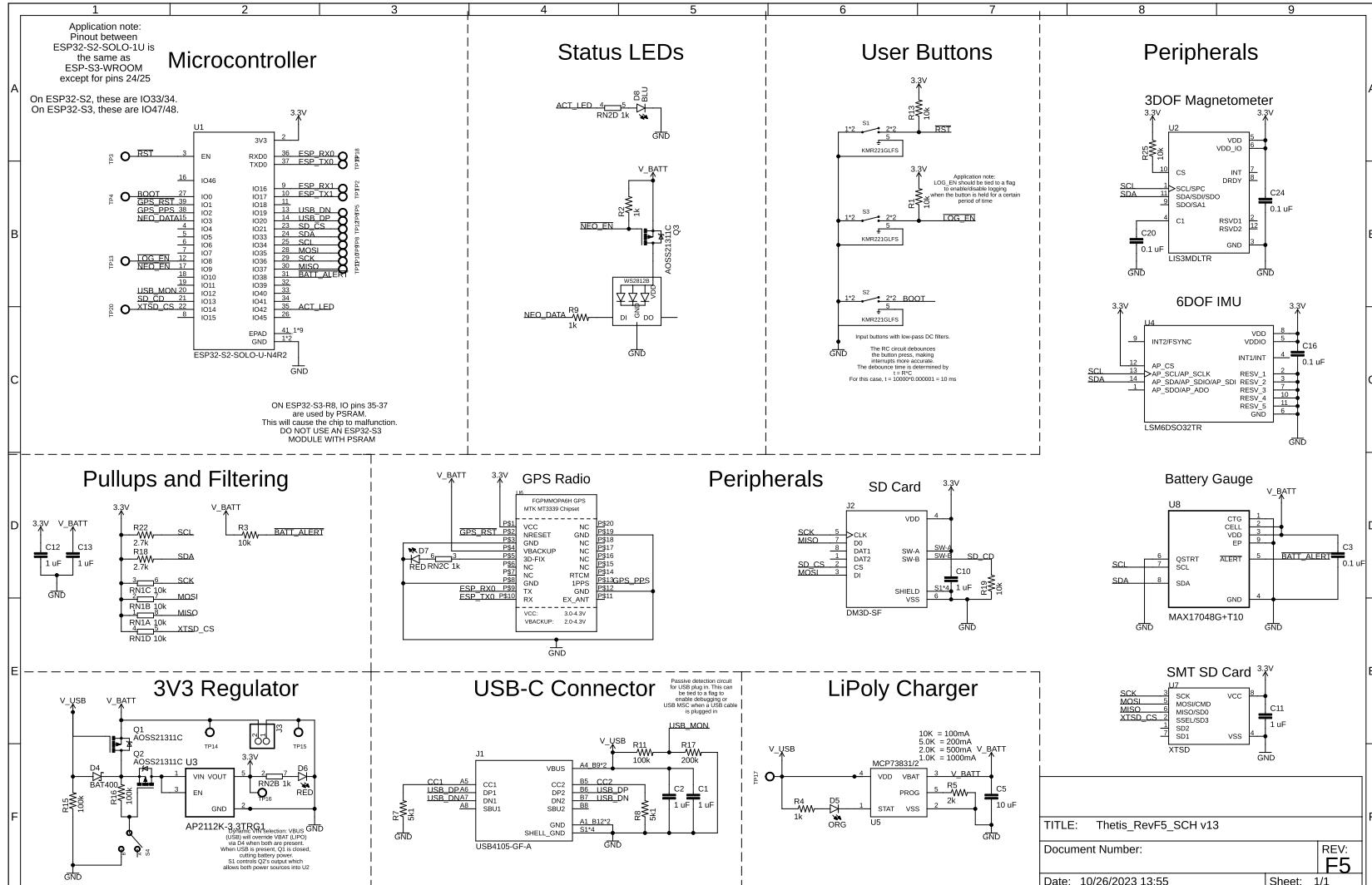
184



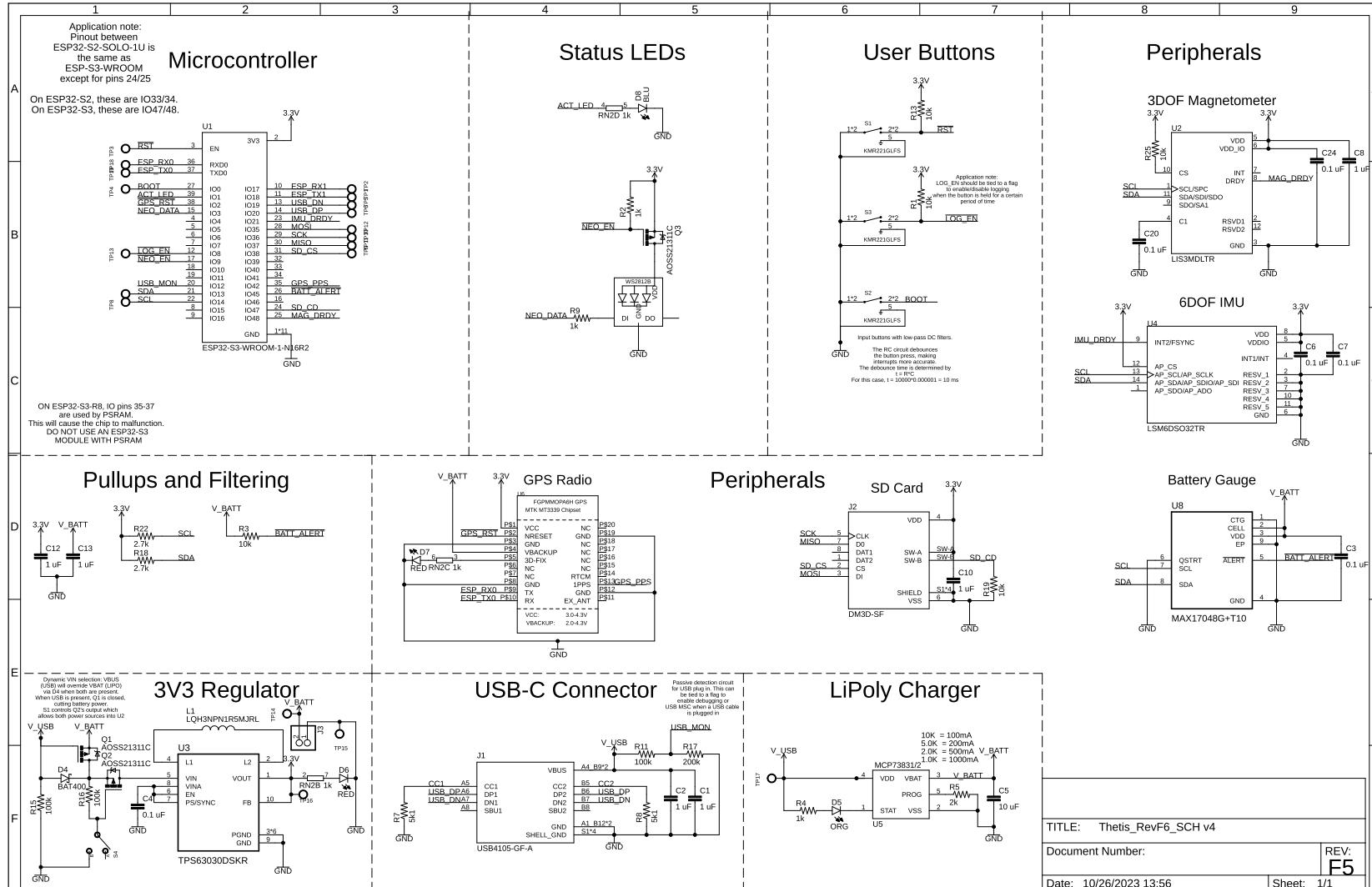
J.7 Revision F4 Schematic



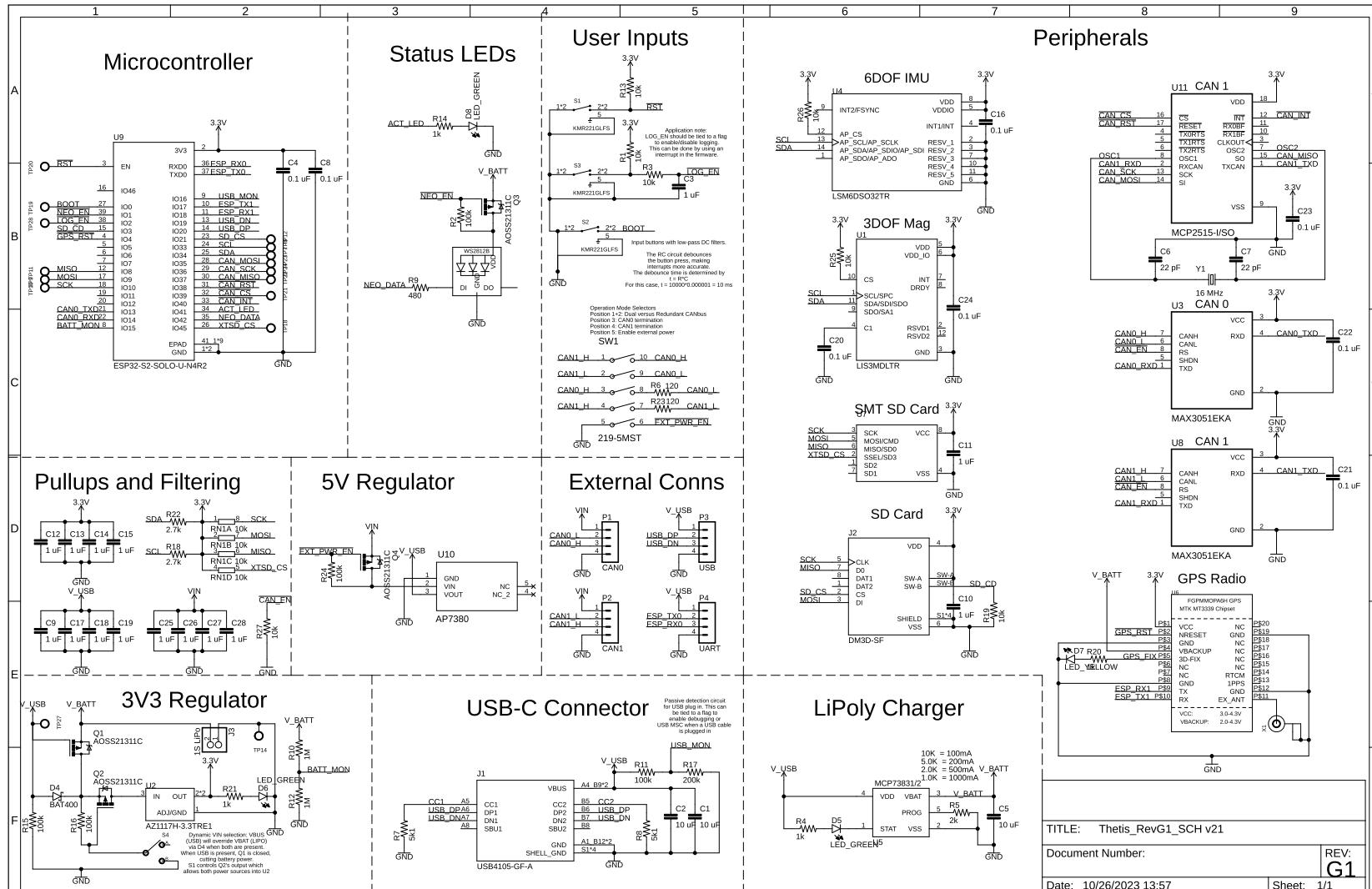
J.8 Revision F5 Schematic



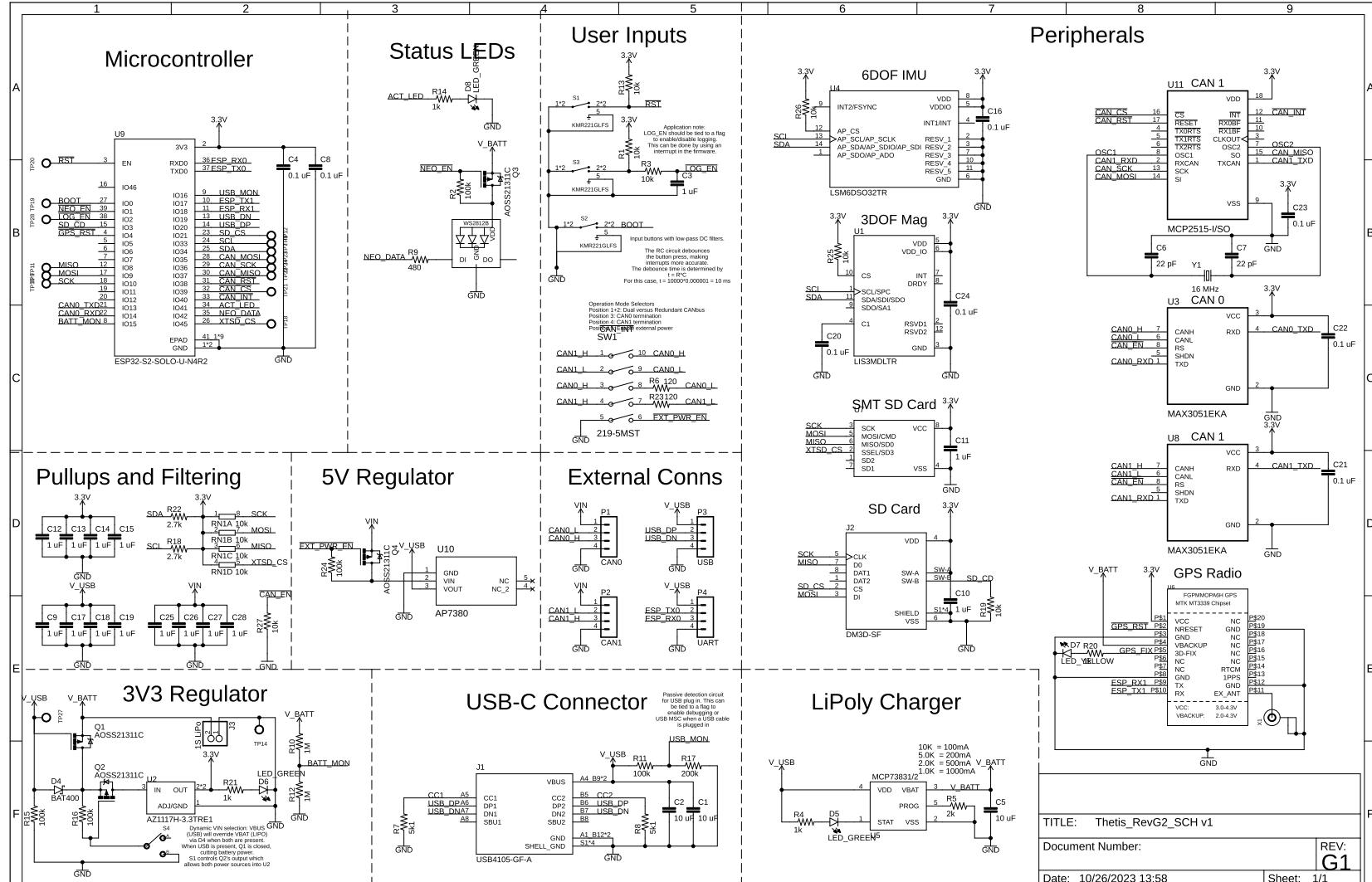
J.9 Revision F6 Schematic



J.10 Revision G1 Schematic



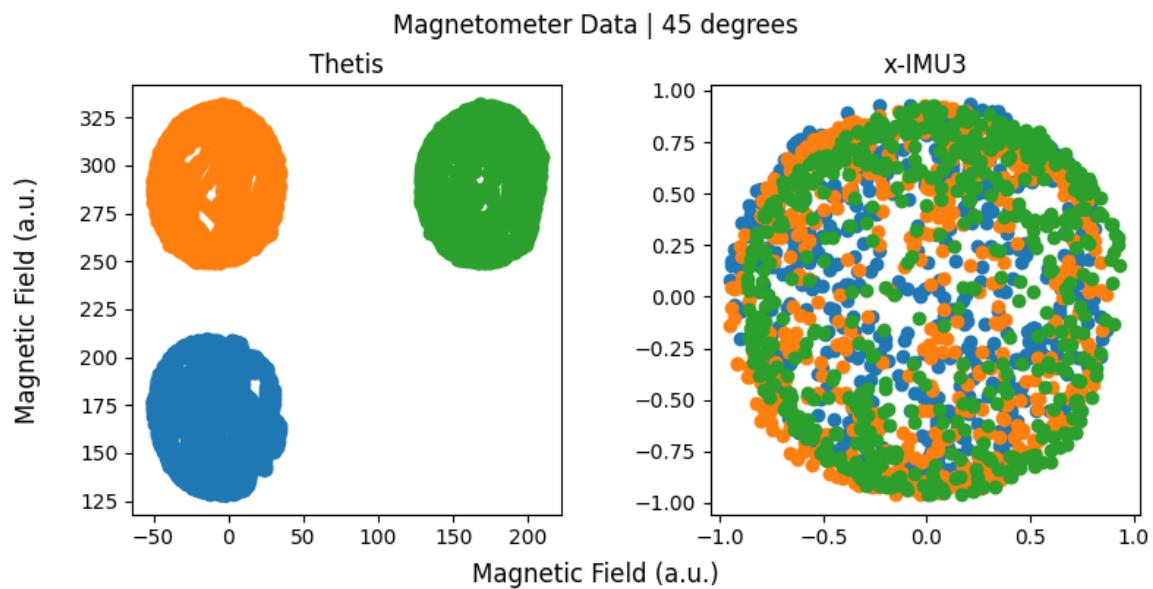
J.11 Revision G2 Schematic



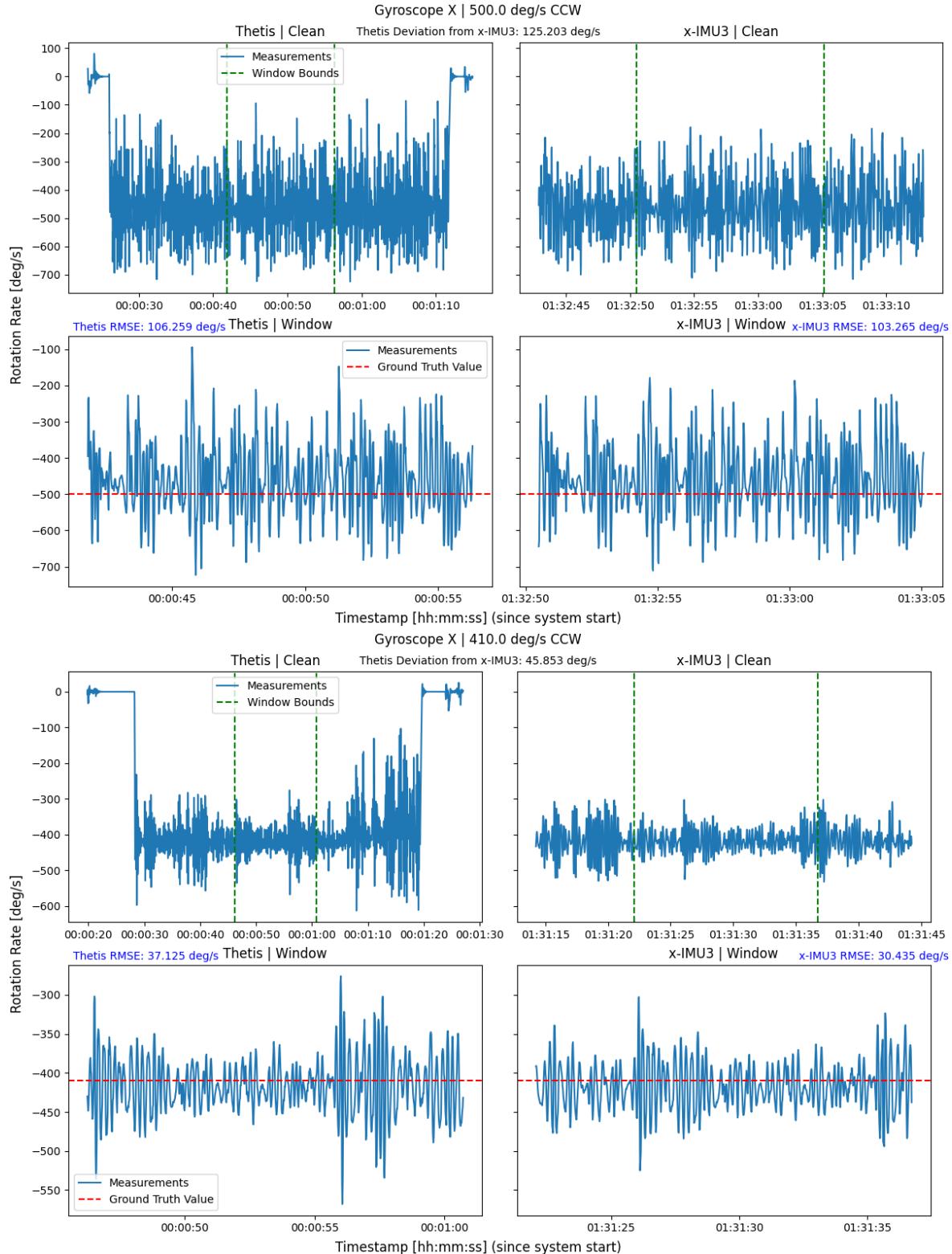
Appendix K

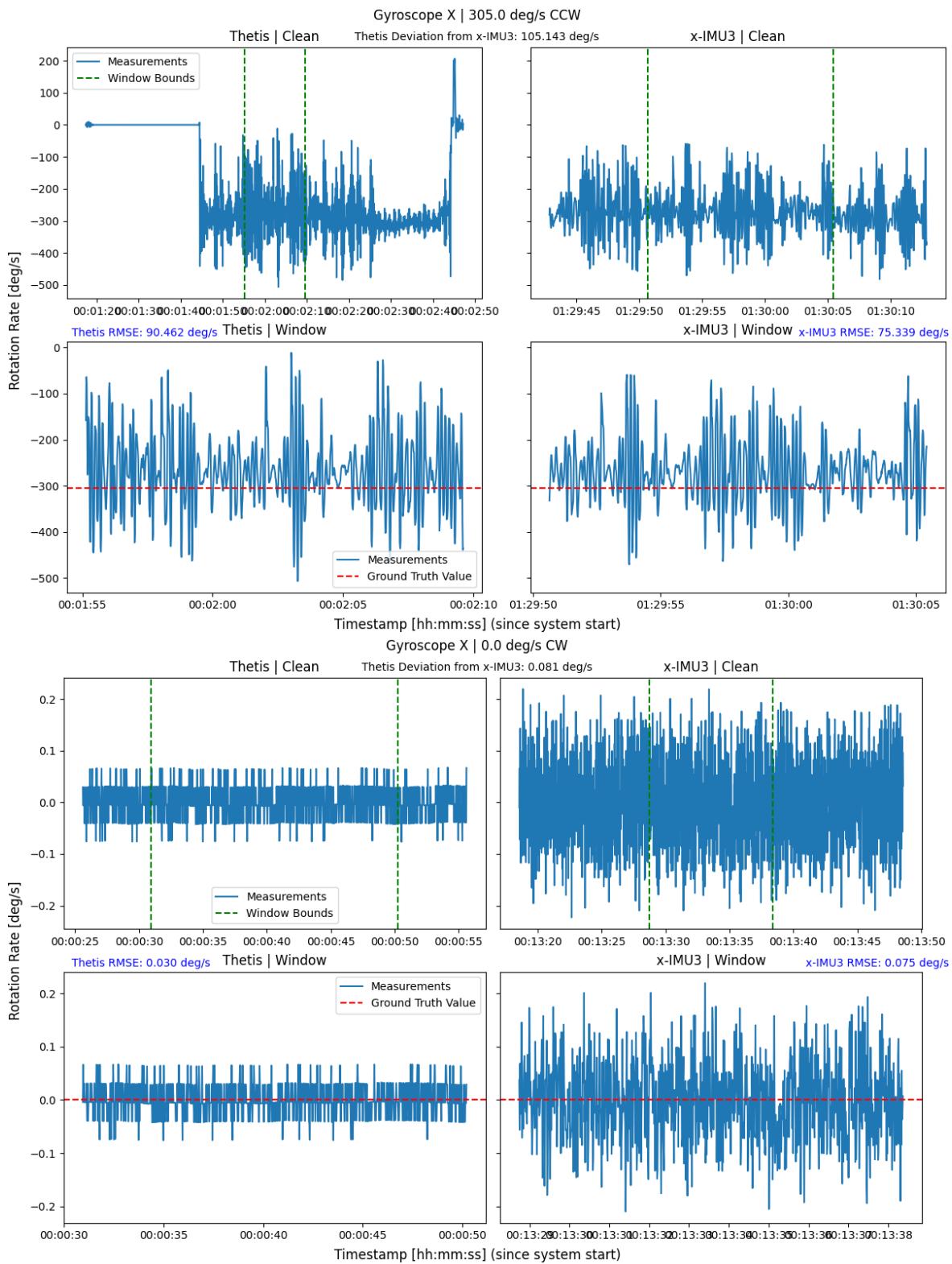
Raw Calibration Data Plots

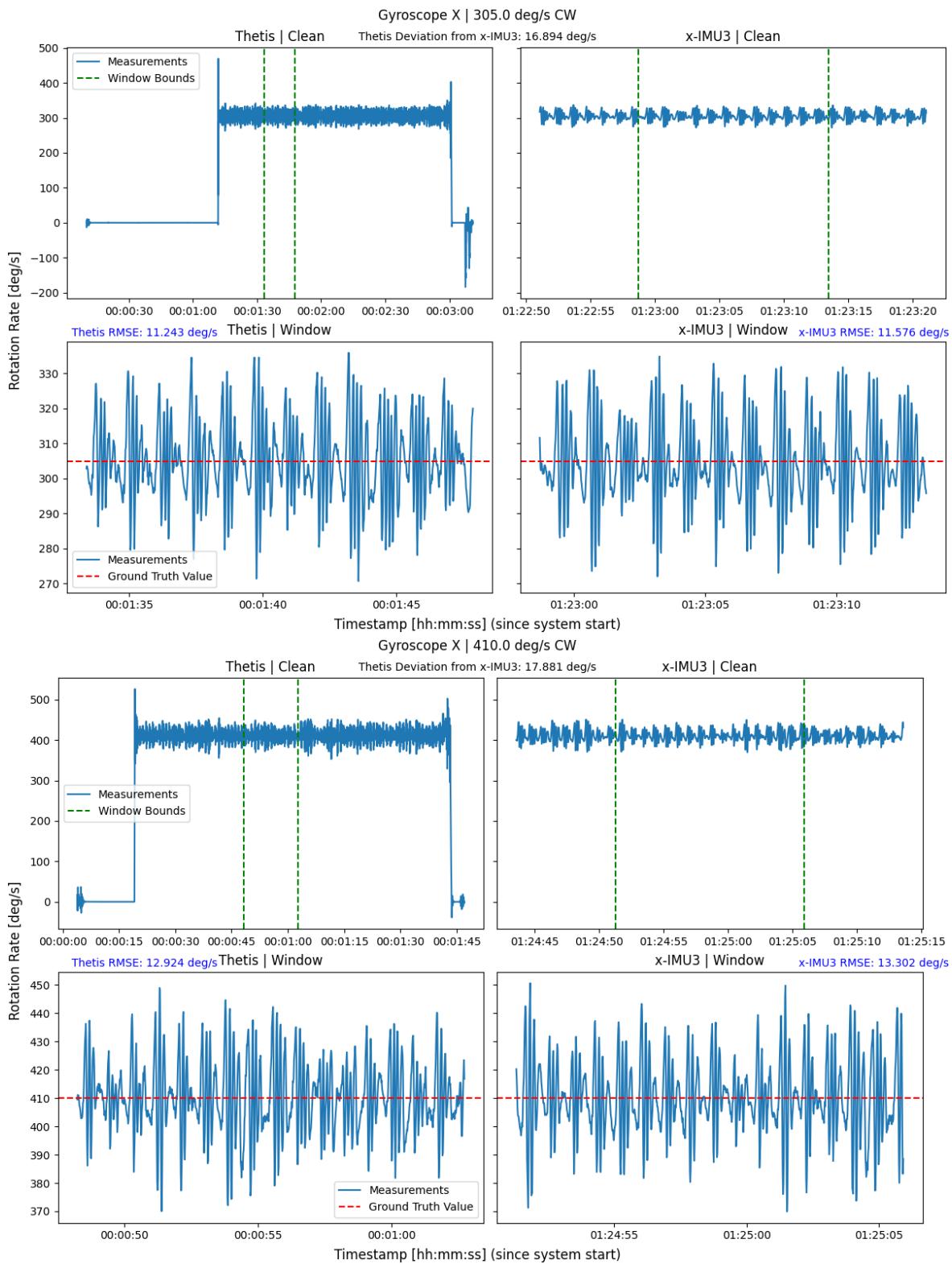
K.1 Magnetometer Data

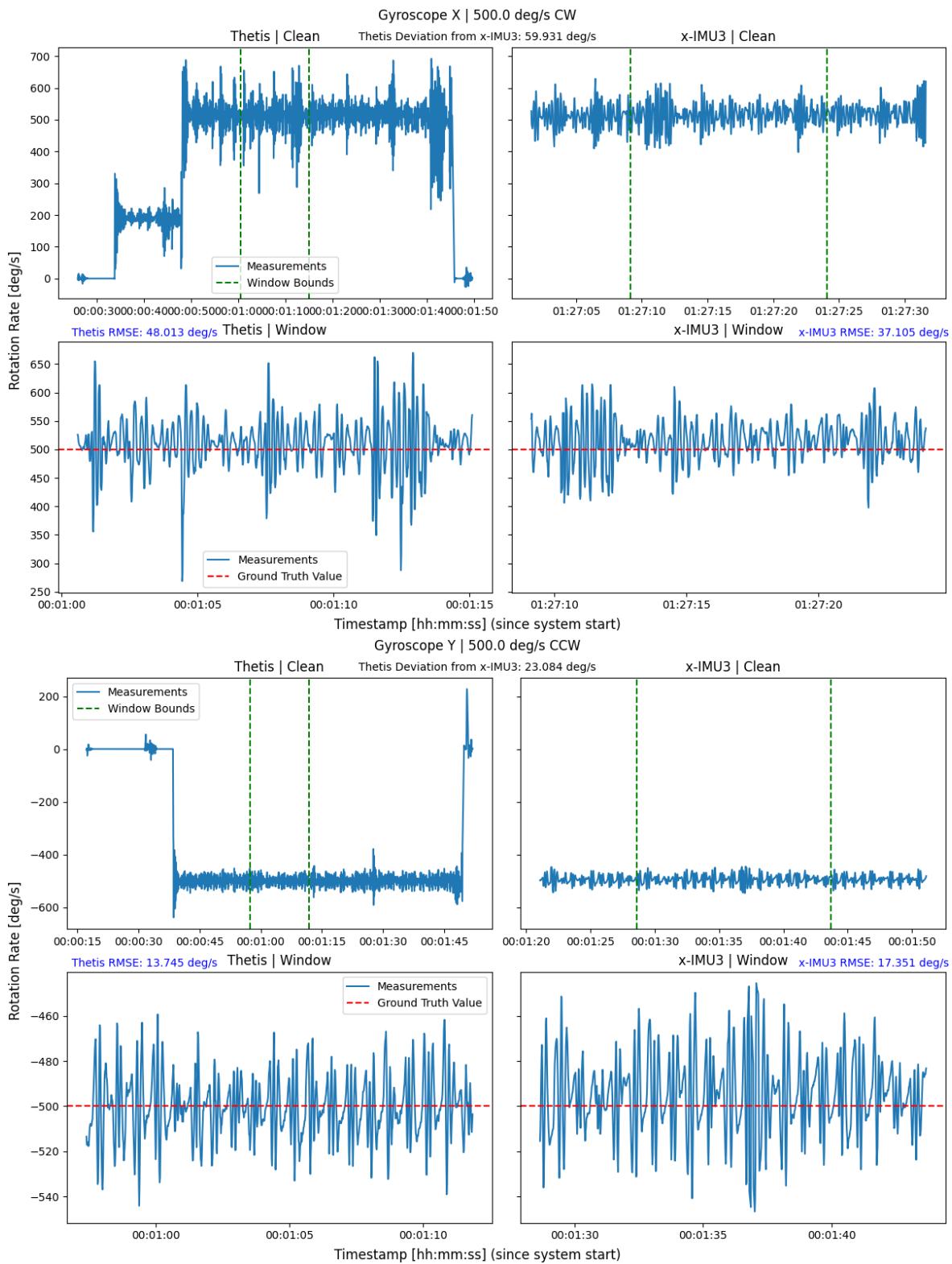


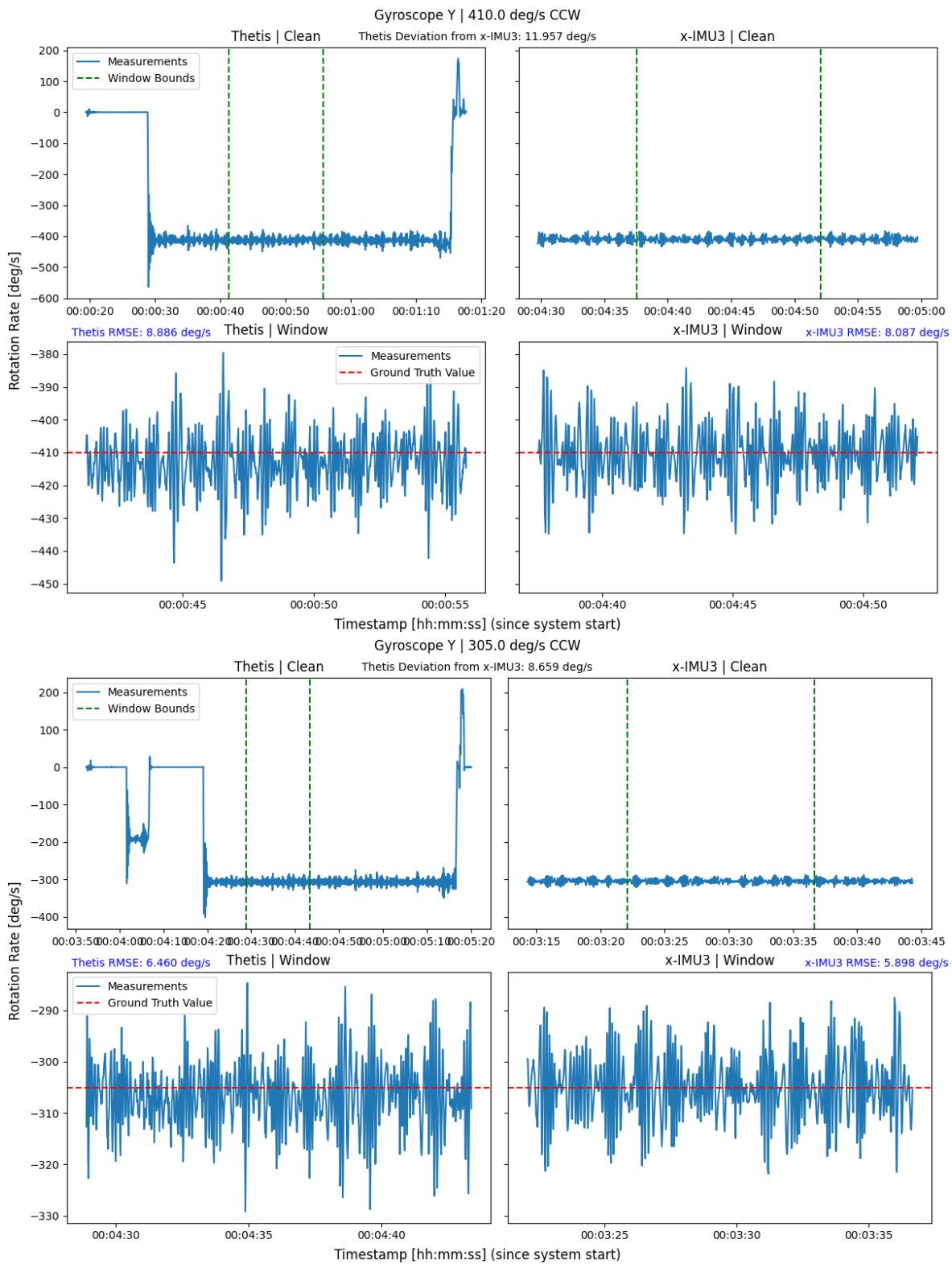
K.2 Gyroscope Data

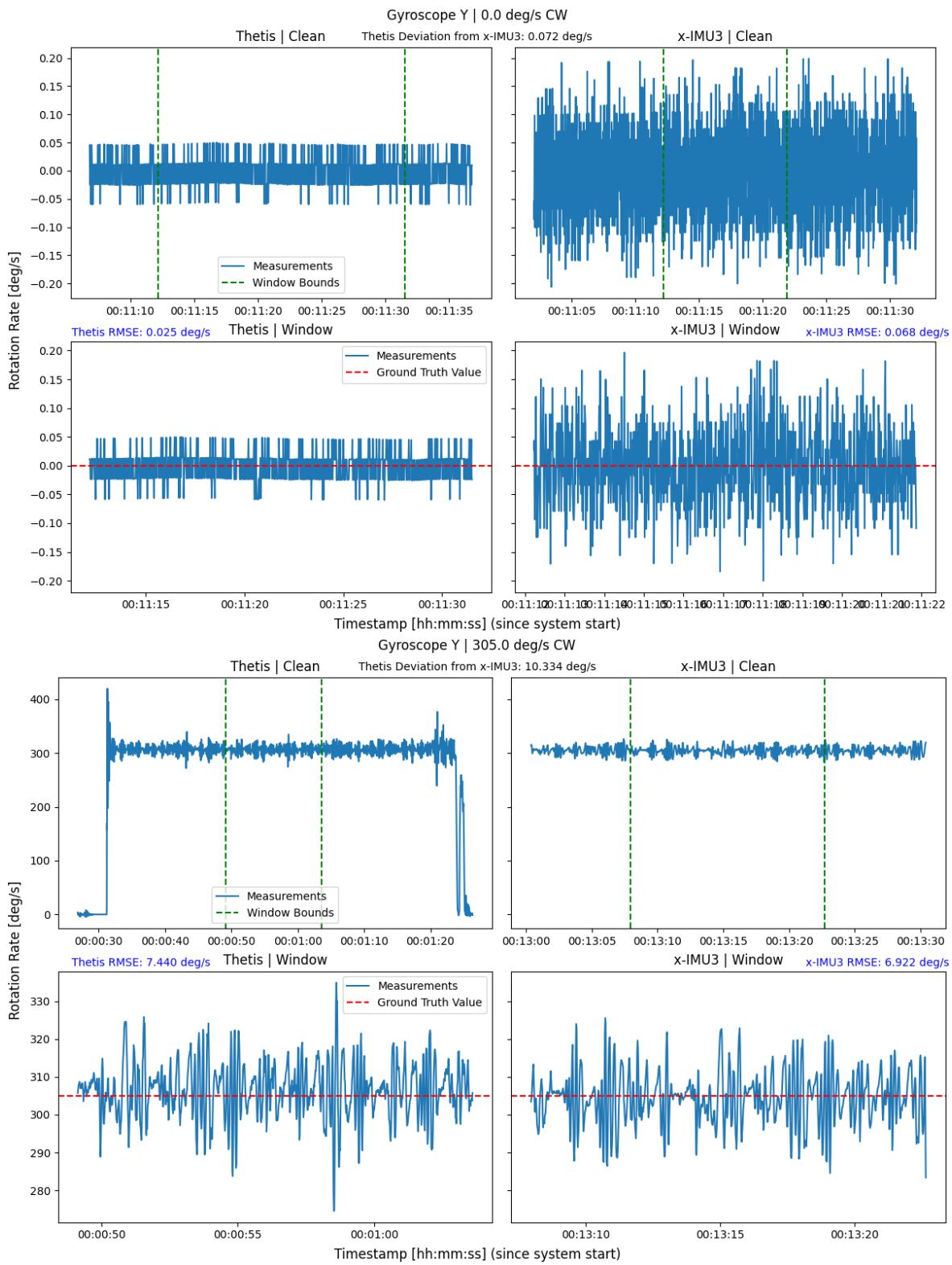


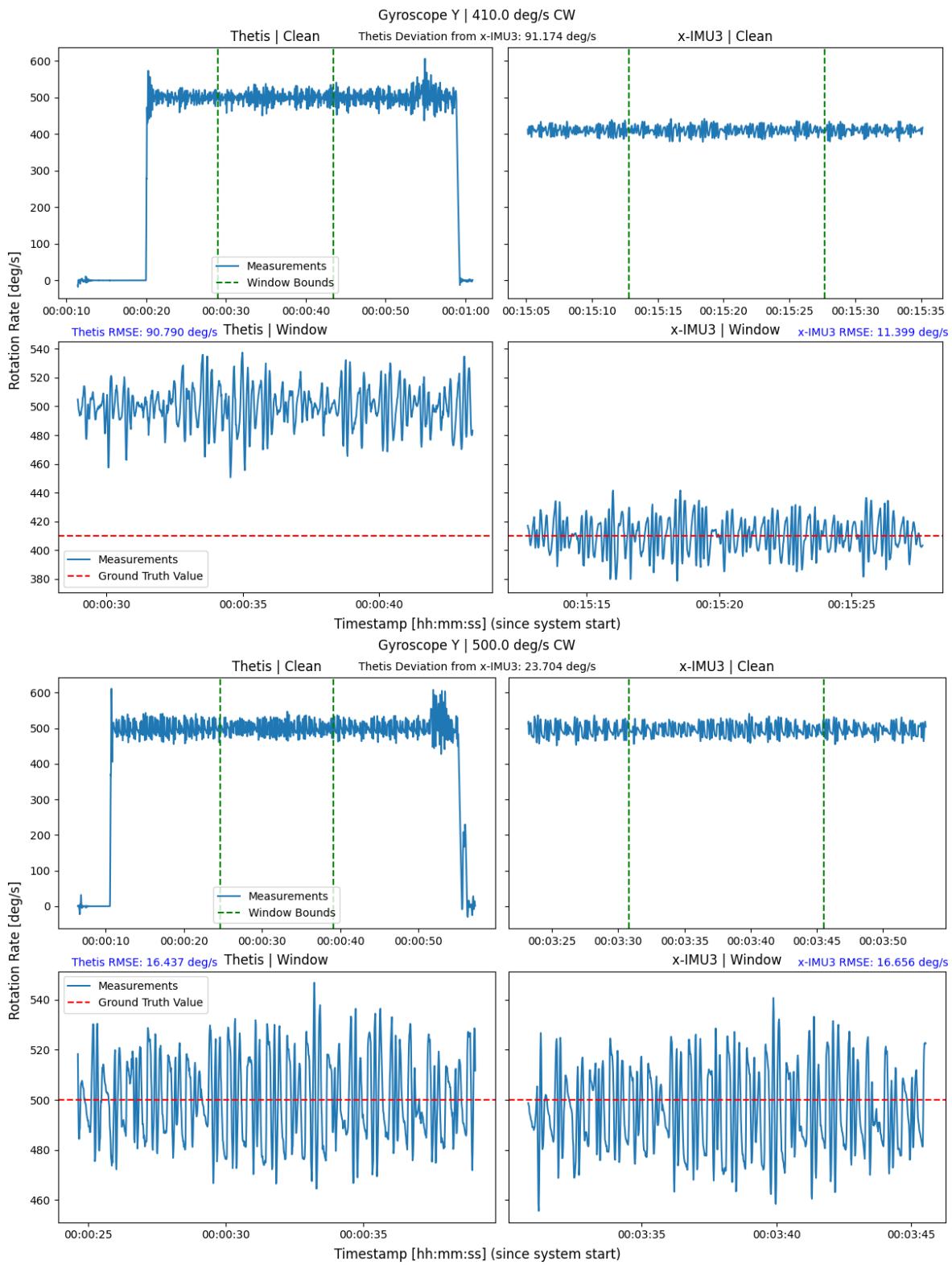


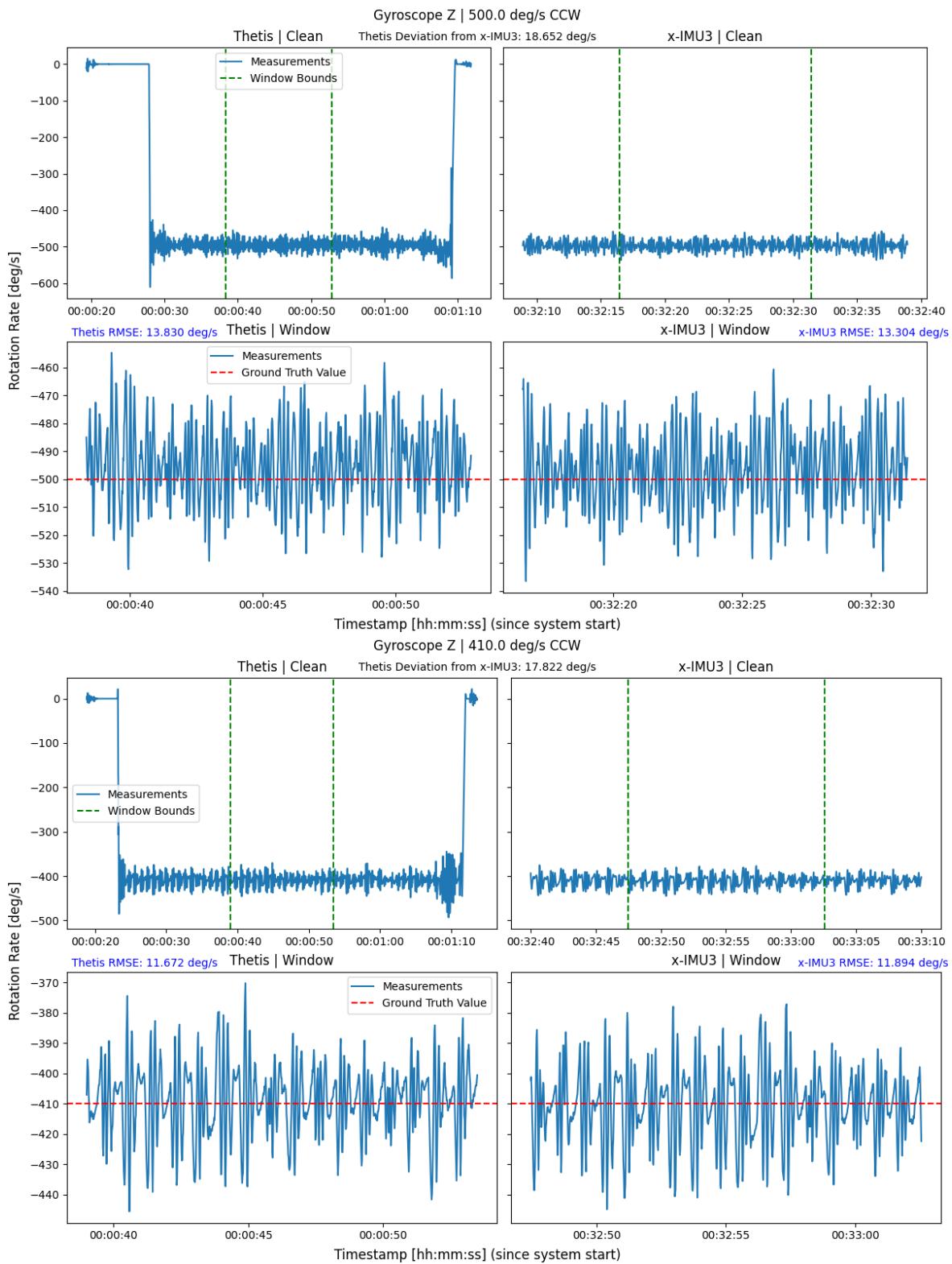


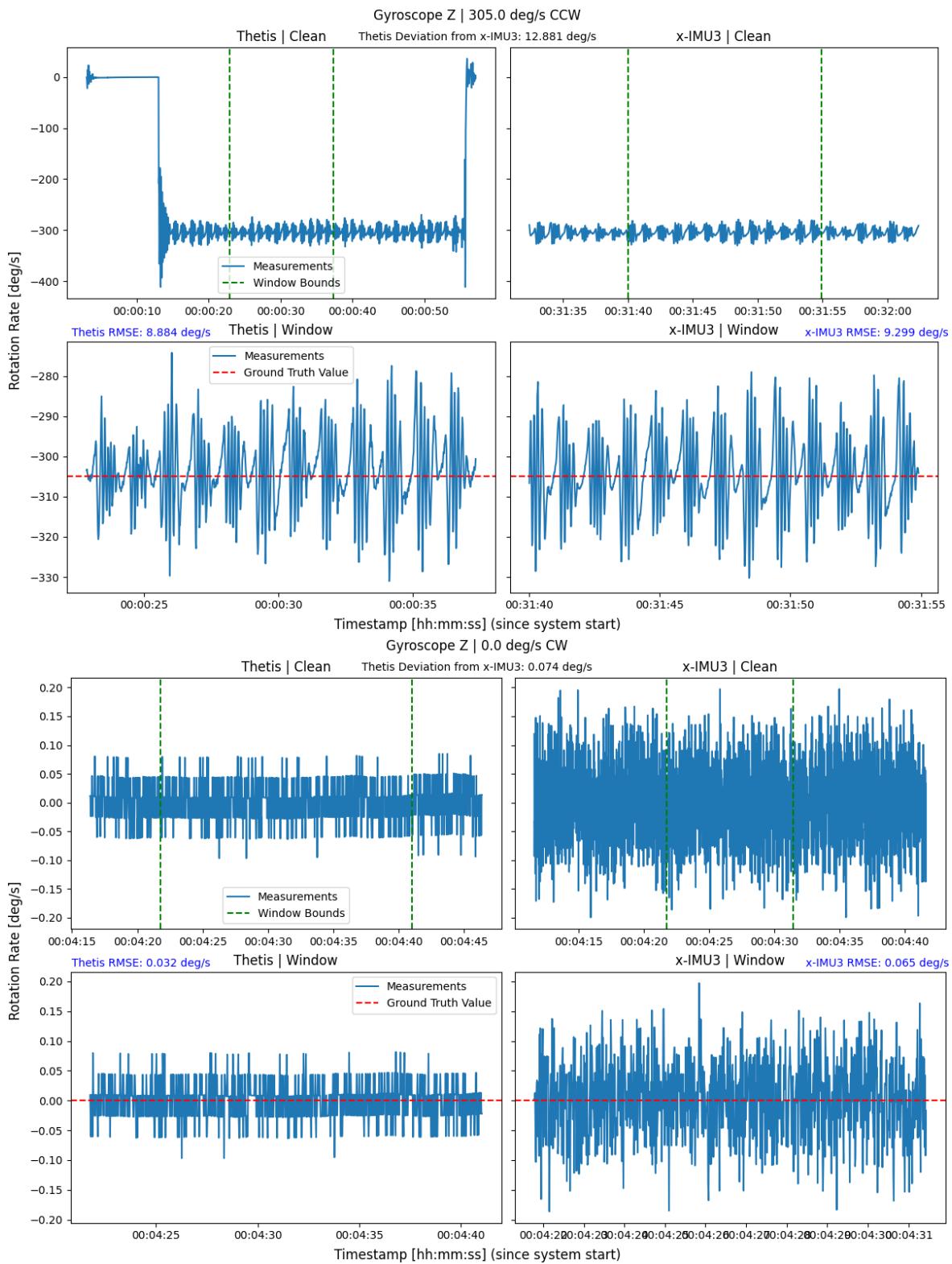


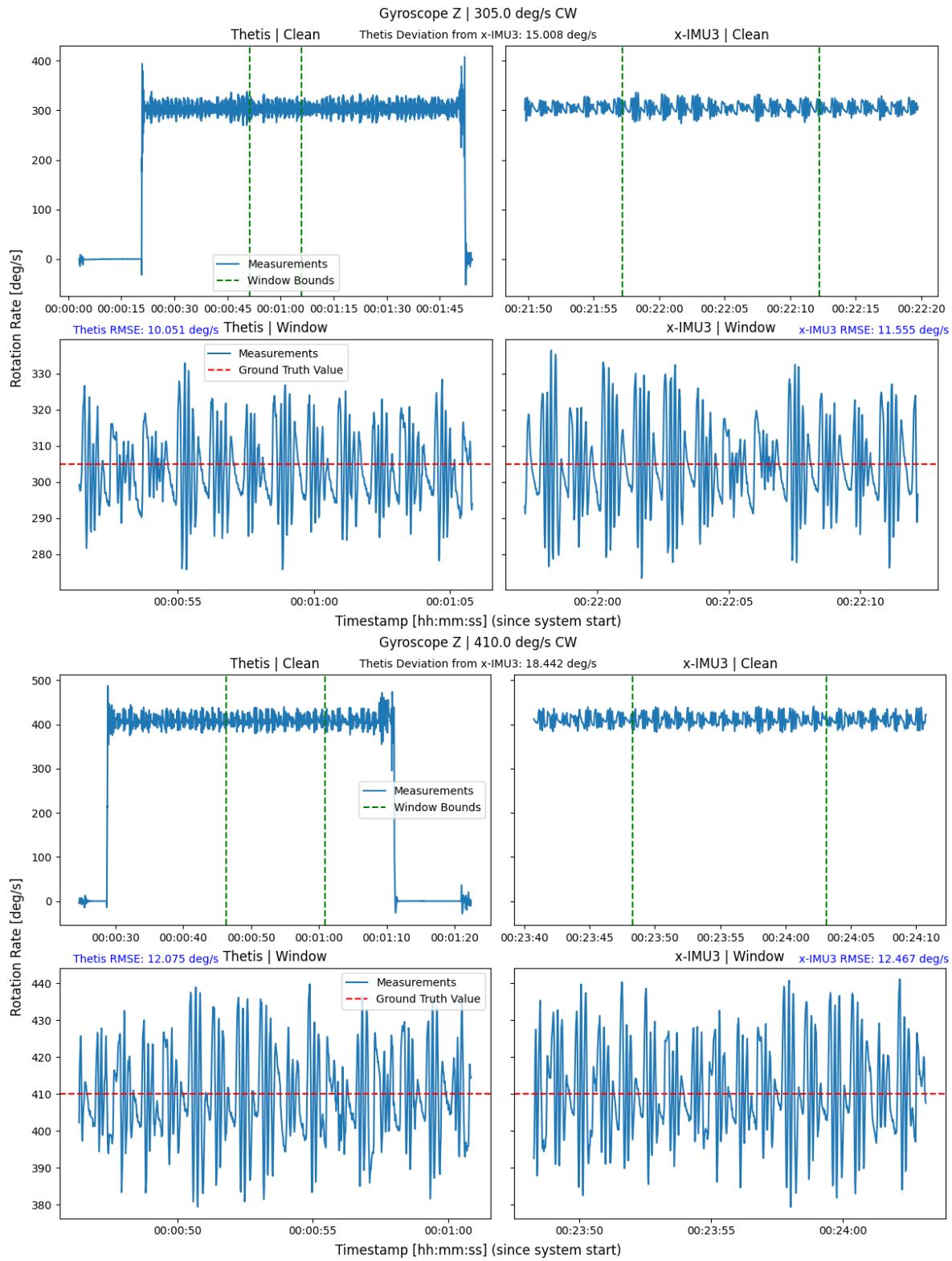


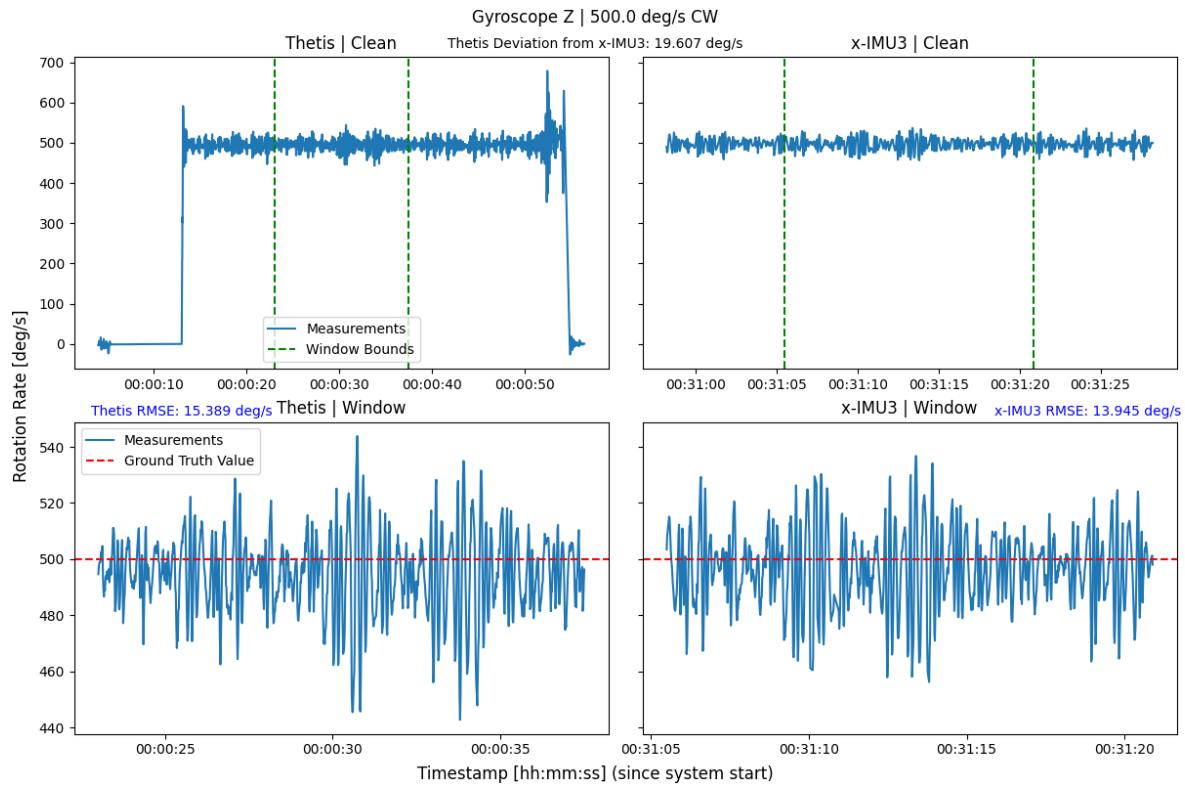




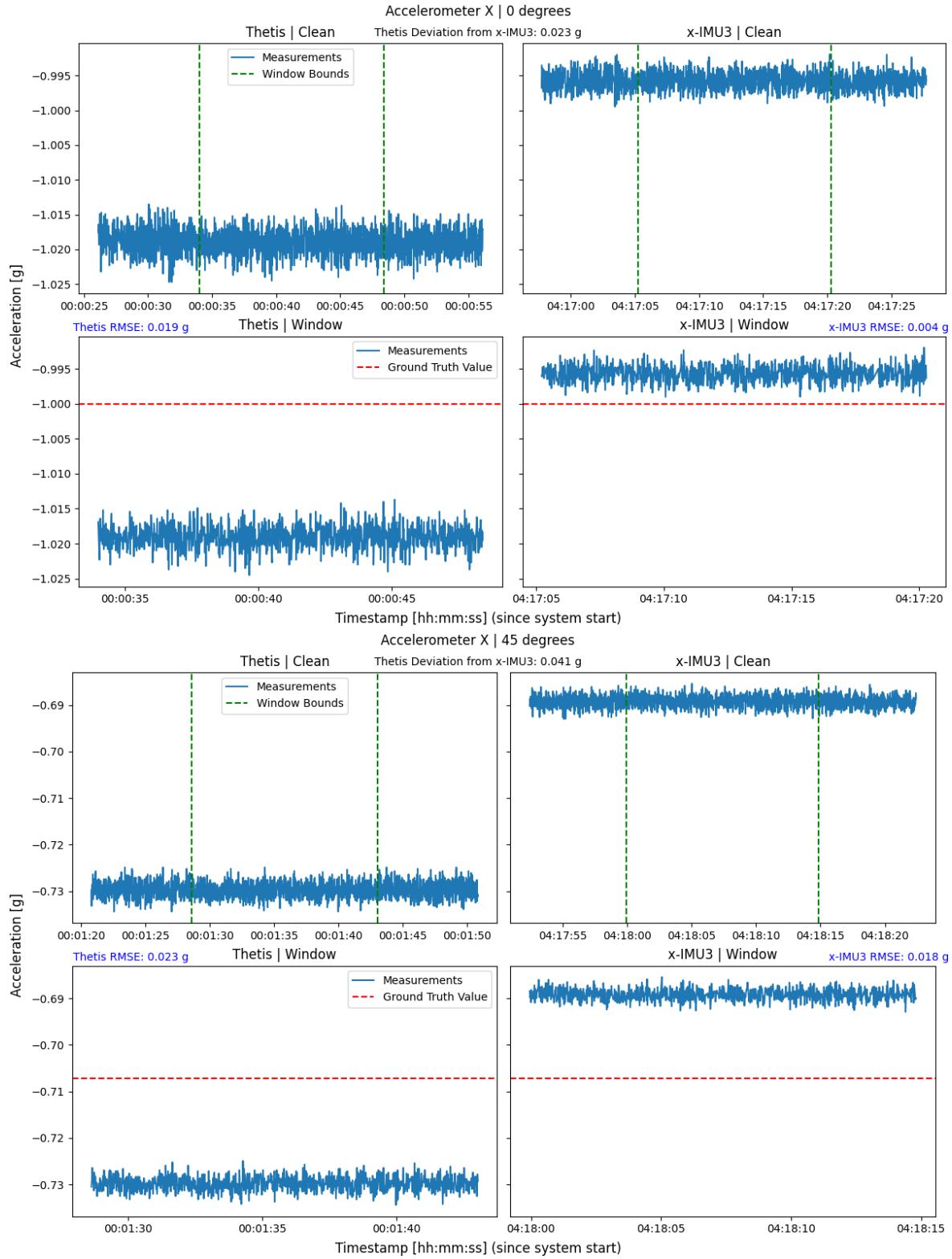


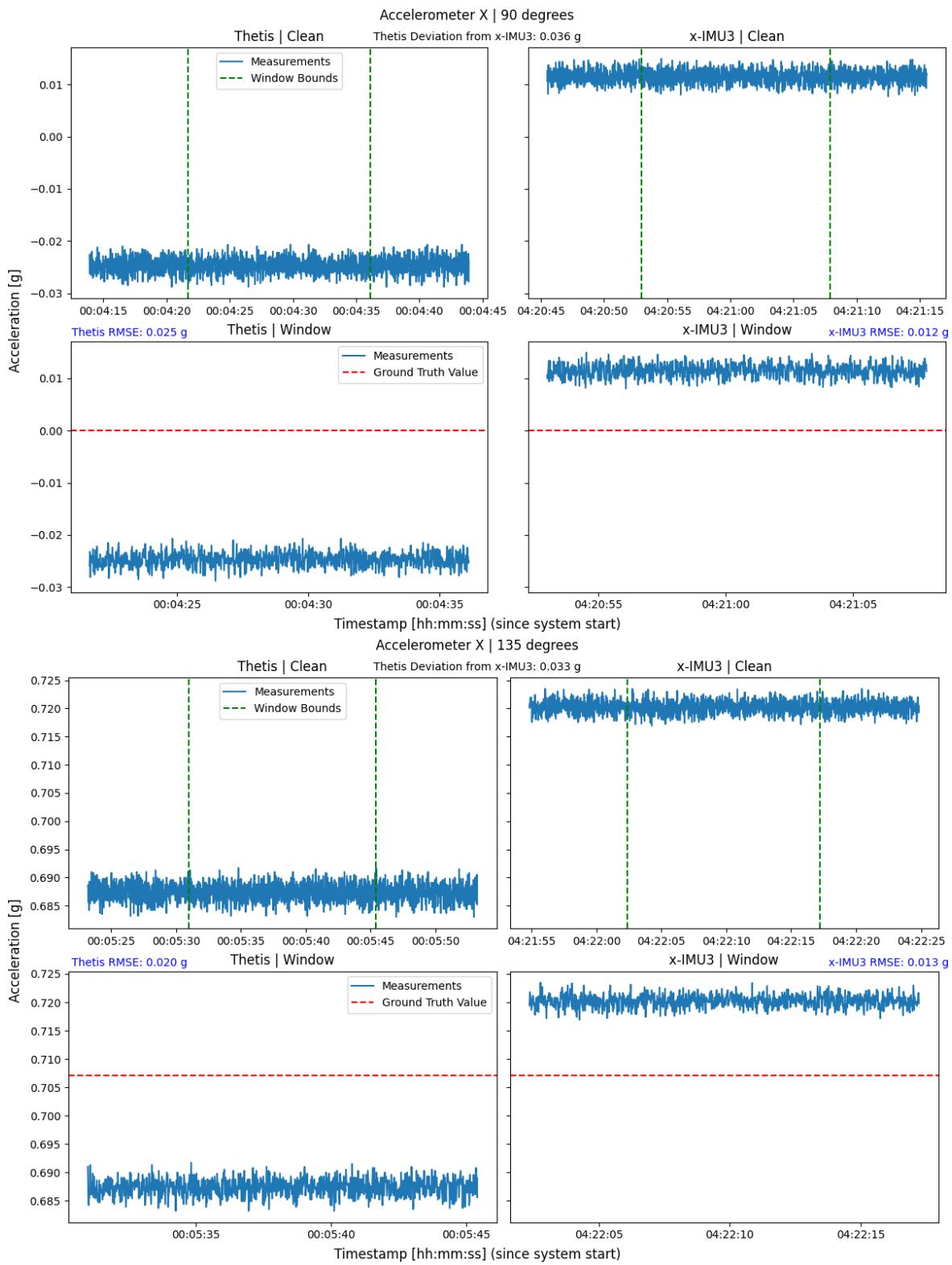


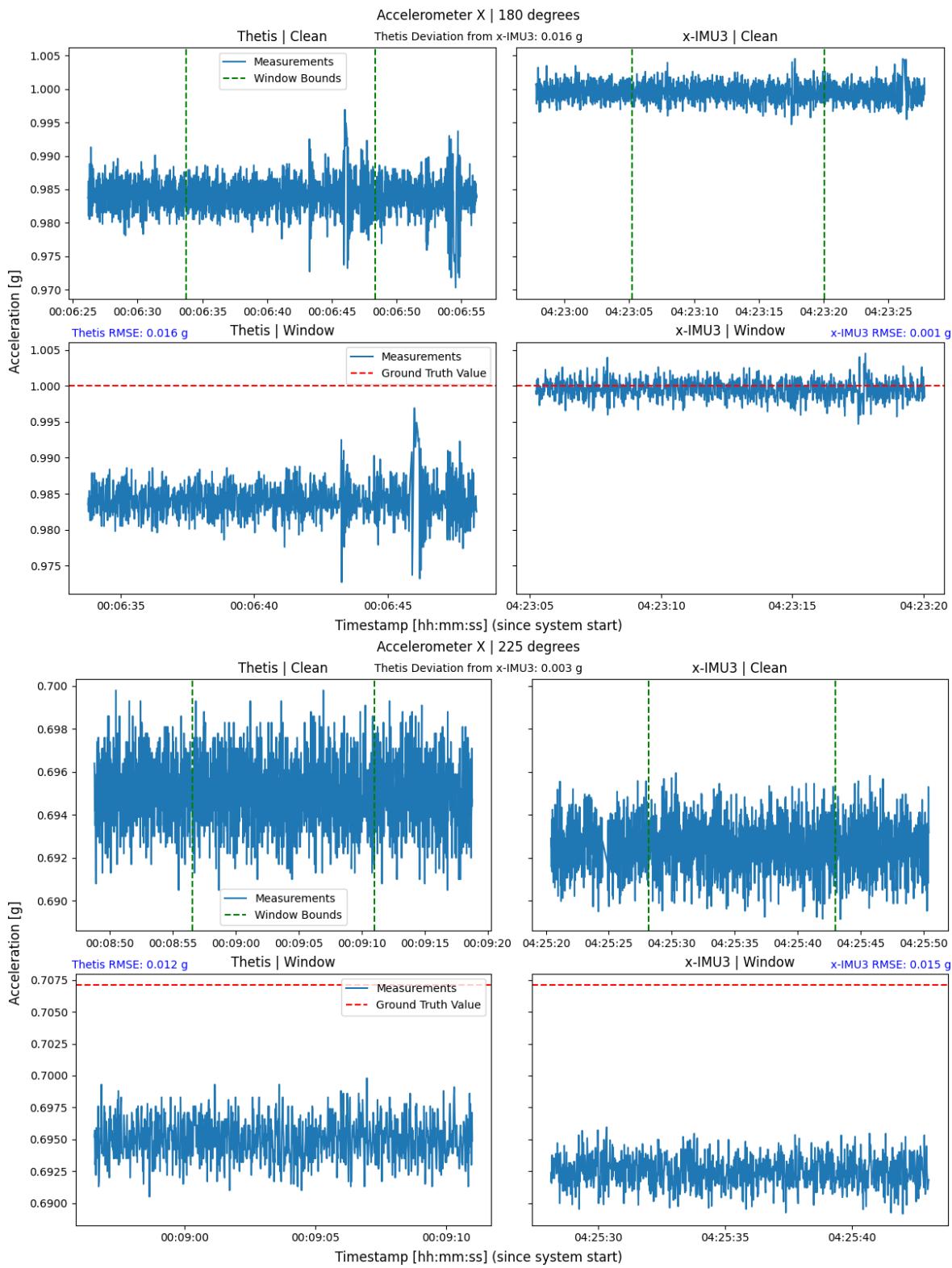


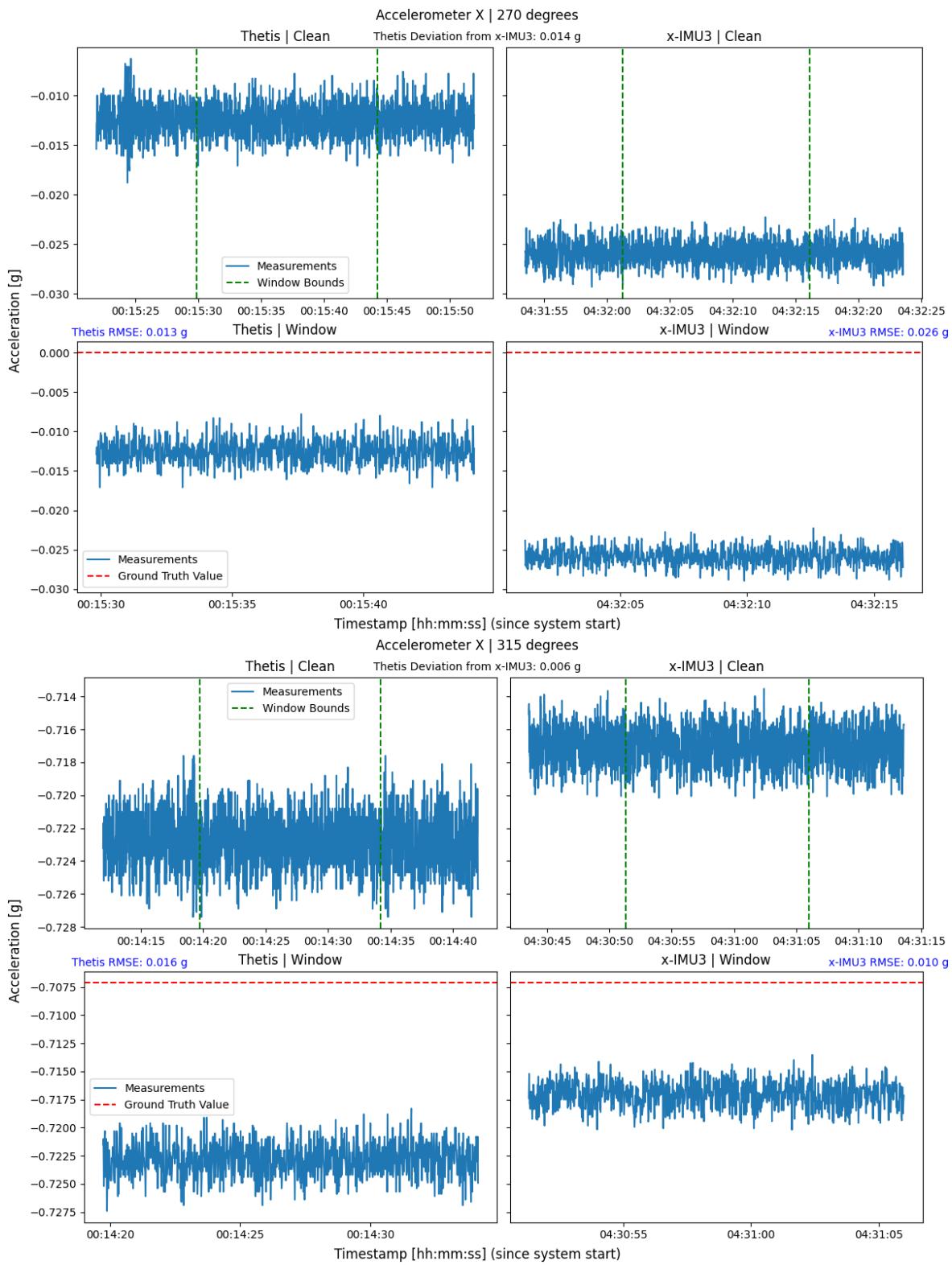


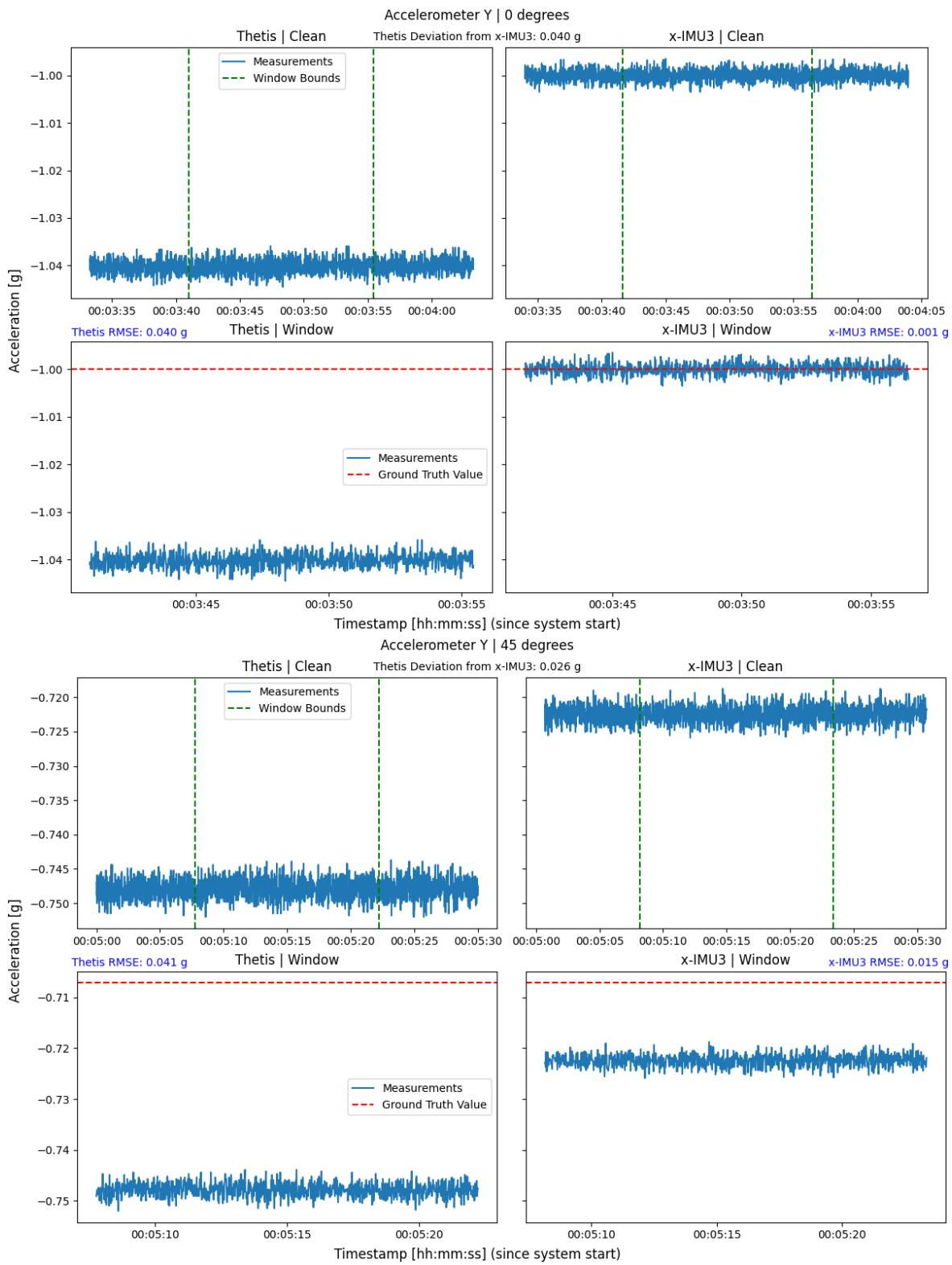
K.3 Accelerometer Data

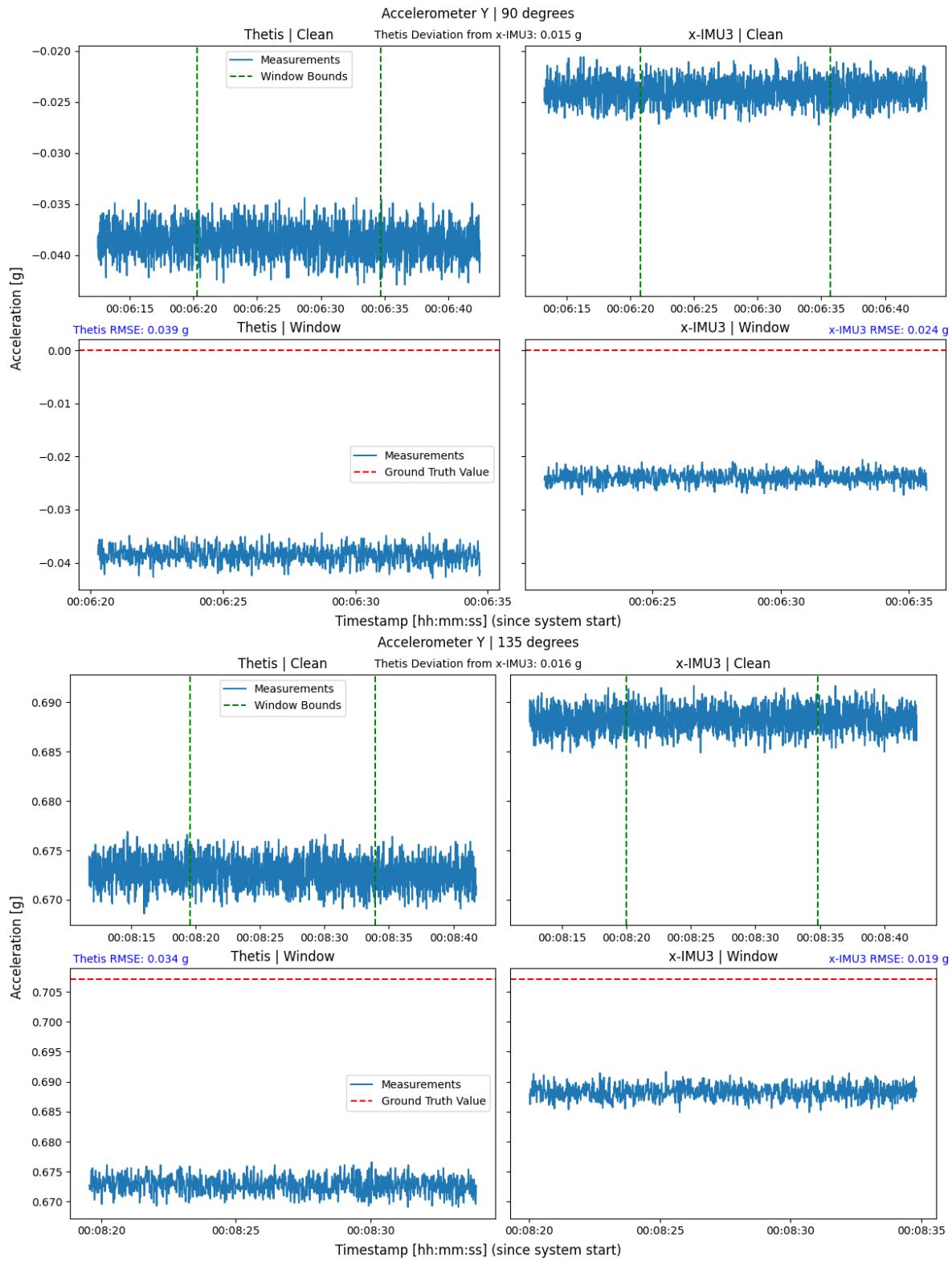


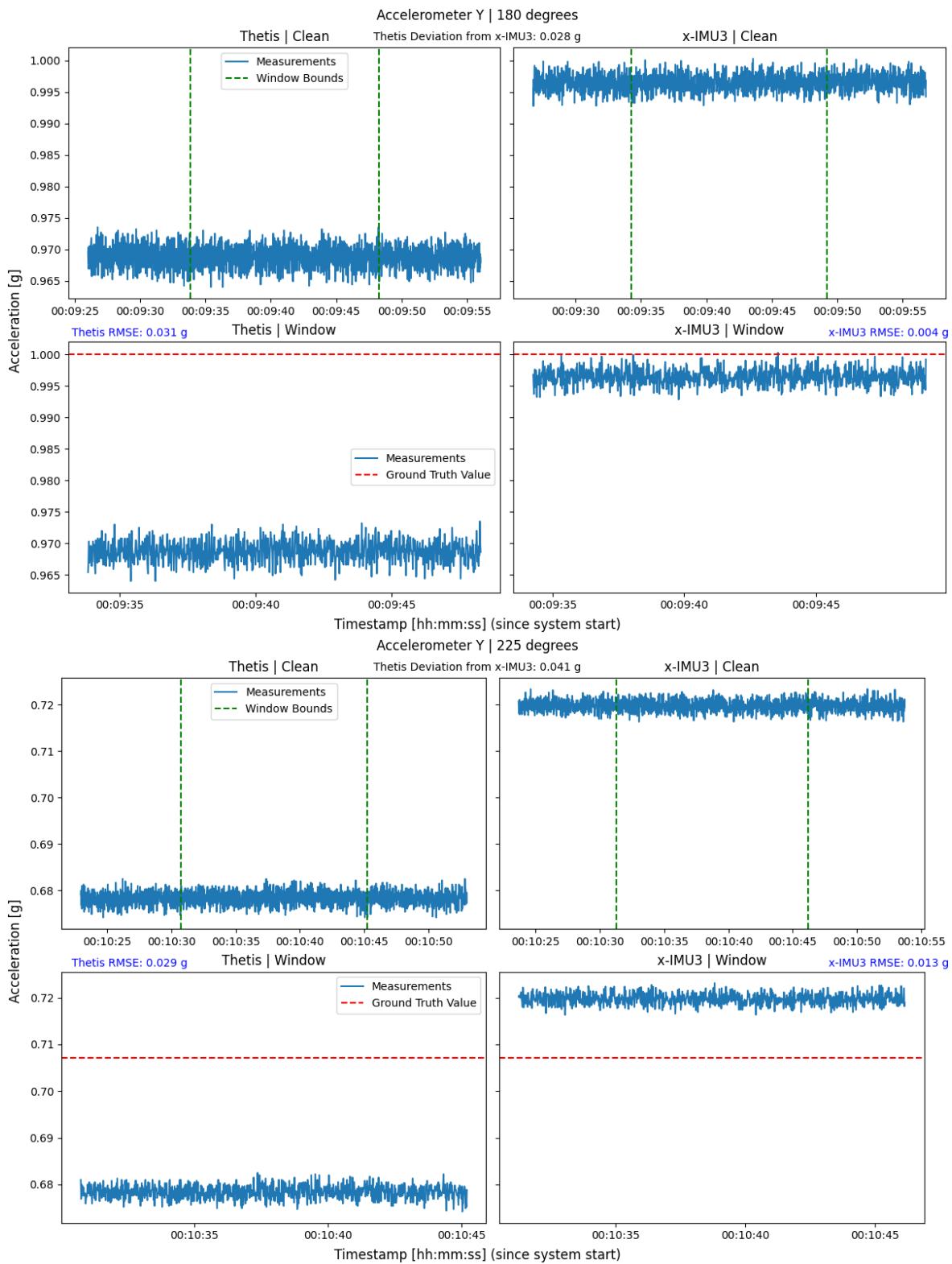


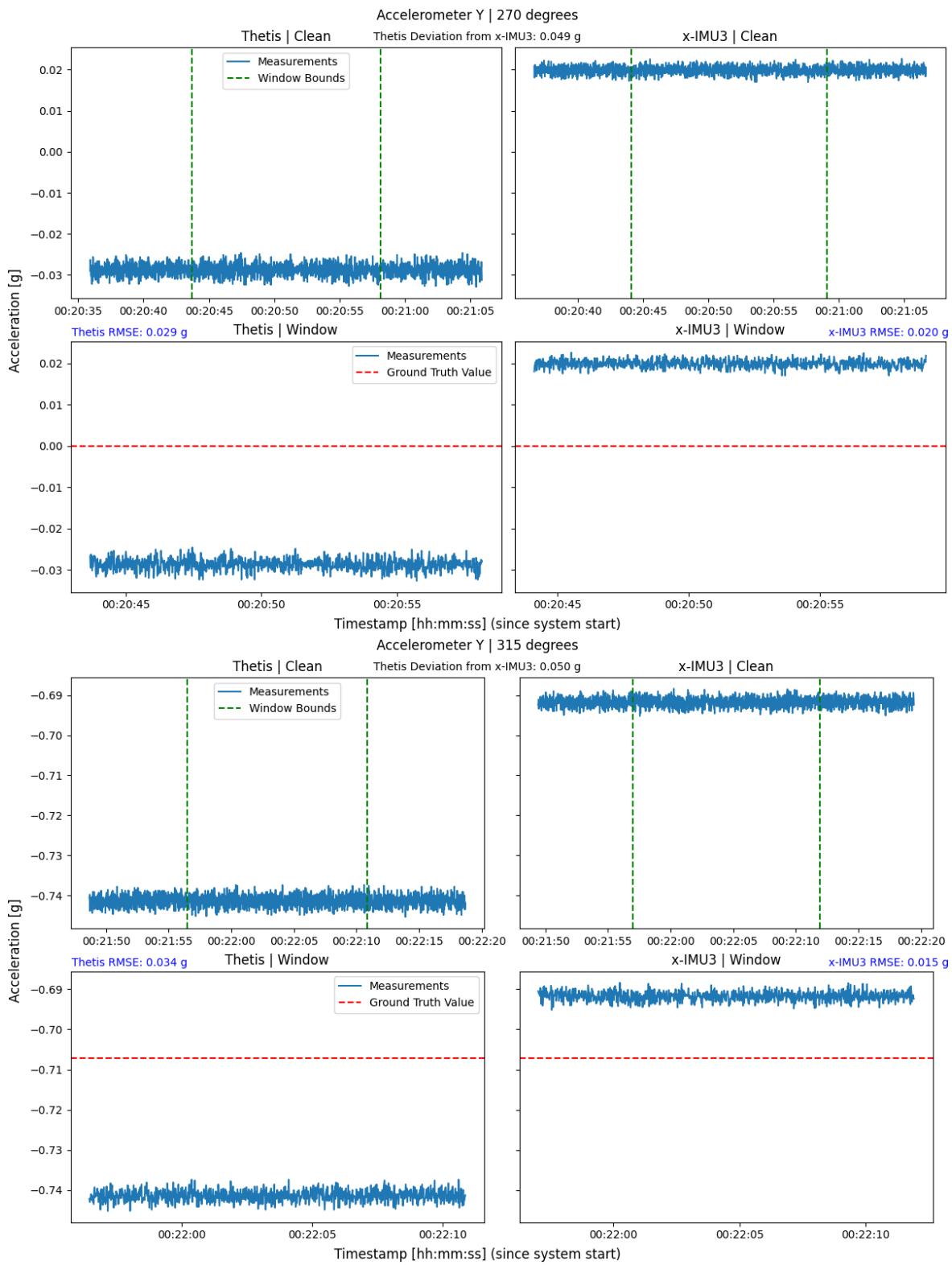


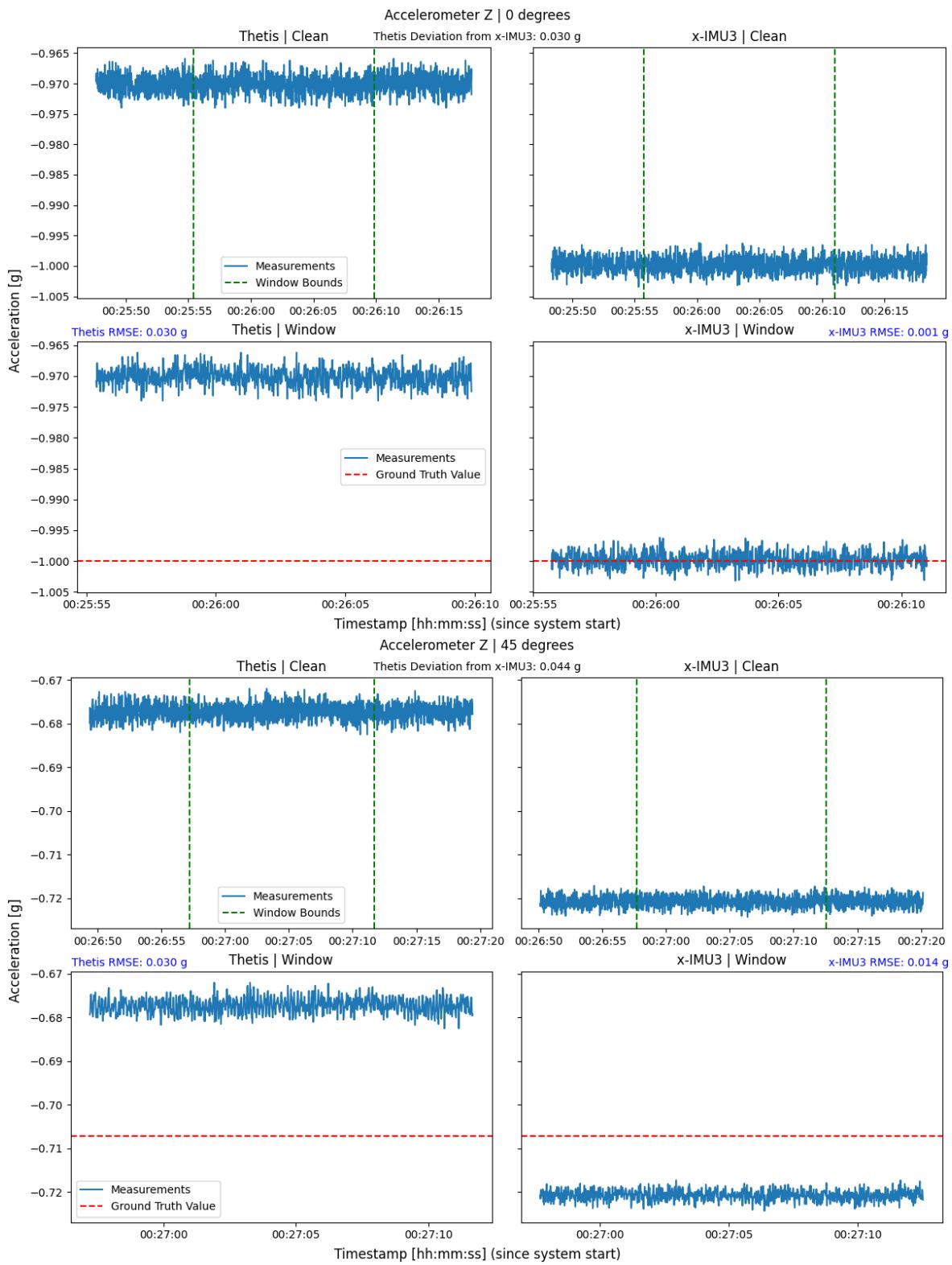


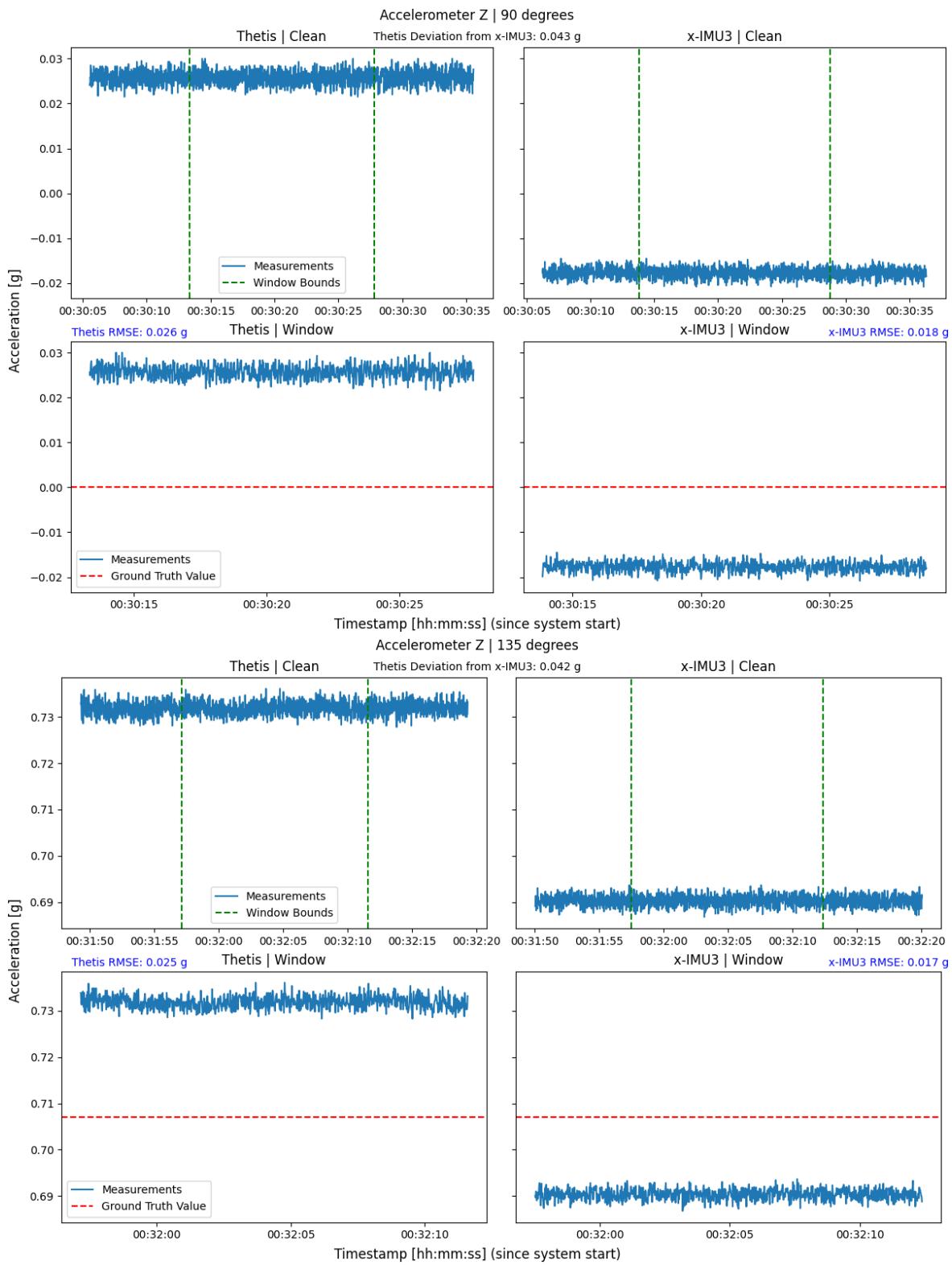


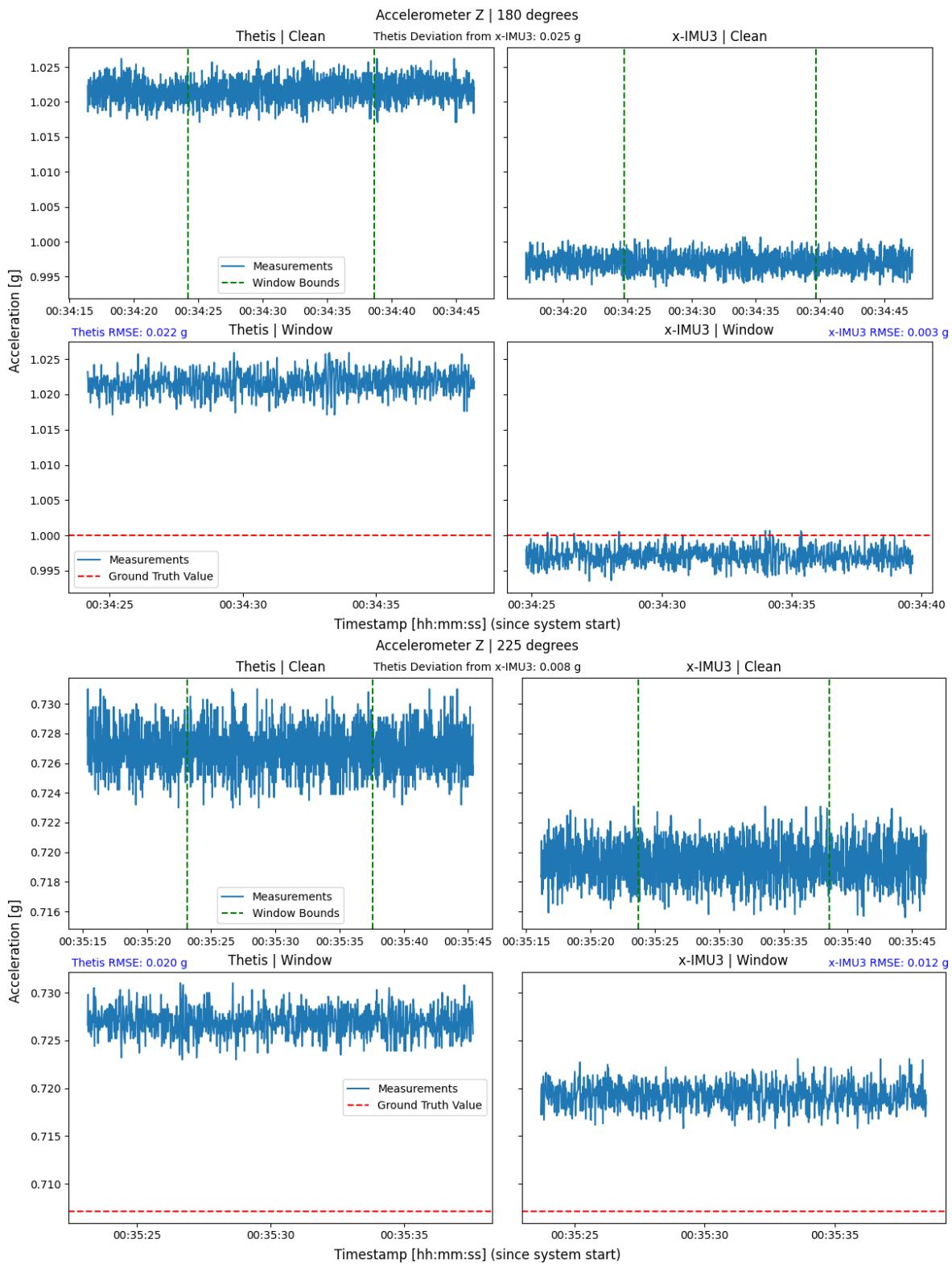


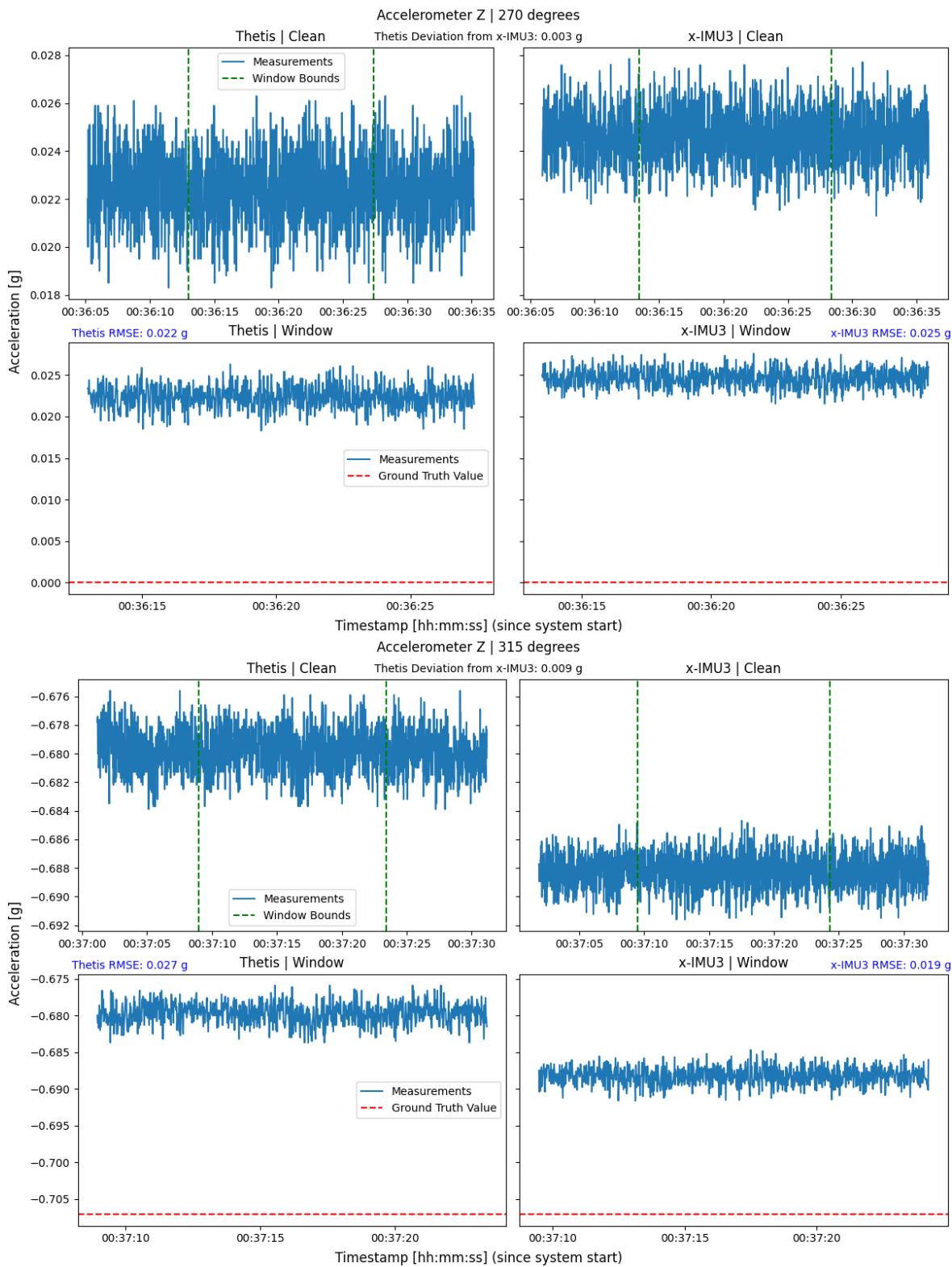












Appendix L

Software Installation and Setup

The Thetis instrumentation board is an open source all-in-one inertial data logger for three dimensional bodies. It was designed and developed for a Master’s thesis in Ocean Engineering and the Florida Institute of Technology from 2020 to 2023. This repository is the firmware used on that board to capture sensor measurements and compute body orientation and log all the appropriate data.

L.1 Installing the Development Environment

This firmware was developed in Microsoft Visual Studio Code using the PlatformIO extension on Windows 10 for compilation and uploading. To automatically install and initialize the environment, download the initialization scripts¹ and execute `1-dev_environment_setup.bat` as an administrator, following the on-screen prompts. It will go through the steps discussed in the following sections.

¹<https://github.com/Legohead259/Thetis-Firmware/tree/master/scripts>

L.1.1 Step 1: Install VS Code and git VCS

Download VisualStudio Code² from Microsoft and install it following the on-screen prompts.

Alternatively, it can be installed from the command prompt as an administrative user via:

```
winget install -e -id Microsoft.VisualStudioCode
```

Then, download git³ and install it using the on-screen prompts. You should keep everything to the default value except for the text editor. I would recommend Nano over Vim (unless you know how to exit Vim). This can also be installed from the command prompt via:

```
winget install -id Git.Git -e --source winget
```

Finally, **for developers only** download the GitHub CLI client⁴ and install it. This can be done from the command prompt via:

```
winget install --id=GitHub.cli -e
```

Close and reopen any command prompt windows and test your installations with the following command:

```
code --version && git --version && gh --version
```

If all the software has been installed correctly, you should see a short output similar to this:

```
1.83.1
```

²<https://code.visualstudio.com/download>

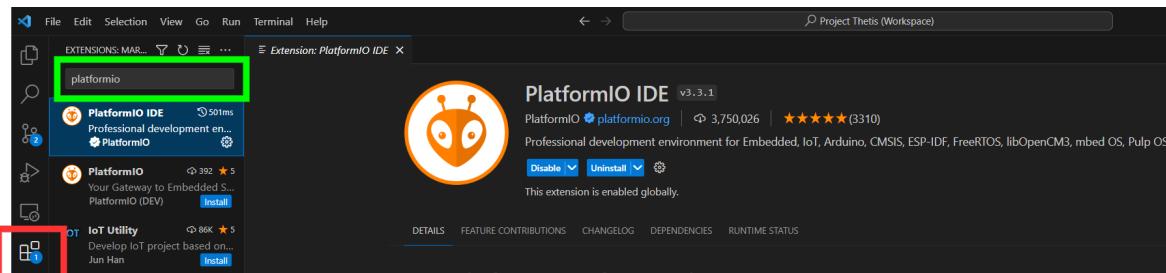
³<https://git-scm.com/downloads>

⁴<https://cli.github.com/>

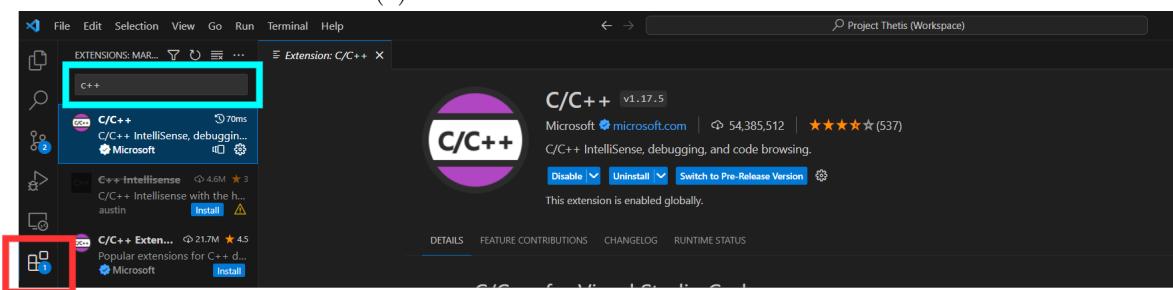
```
f1b07bd25dfad64b0167beb15359ae573aec2cc
x64
git version 2.42.0.windows.2
gh version 2.36.0 (2023-10-03)
https://github.com/cli/cli/releases/tag/v2.36.0
```

L.1.2 Step Two: Install Extensions for VSCode

If all the software has been installed correctly, we can now install the required extensions. PlatformIO⁵ is an advanced development environment for embedded systems. It has excellent integration with VSCode that allows for large projects to be more easily managed and programmed. To install PlatformIO through the VSCode GUI, open the application and look for the “Extensions” tab on the left-hand side (in red). Select it, then search for PlatformIO in the marketplace (green). Do the same for CPP Tools (blue) if not already installed.



(a) PlatformIO extension in VSCode



(b) C++ tools extension in VSCode

Figure L.1: Installation of extensions in VSCode

⁵<https://platformio.org/>

Alternatively, these can be installed in the command prompt with:

```
code --install-extension platformio.platformio-ide  
code --install-extension ms-vscode.cpptools
```

L.1.3 Step Three: Clone the Thetis-Firmware Repository to a Directory

Once the applications have been installed, you can clone the Thetis-Firmware repository [14] to your local machine. Navigate to a directory you want to install the firmware onto (e.g. “C:\USER\Desktop\Thetis\”) and open that location in a command prompt. (*Hint:* You can right-click within the folder and select “Open Git Bash Here” option.) Then, execute the command:

```
git clone https://github.com/Legohead259/Thetis-Firmware.git
```

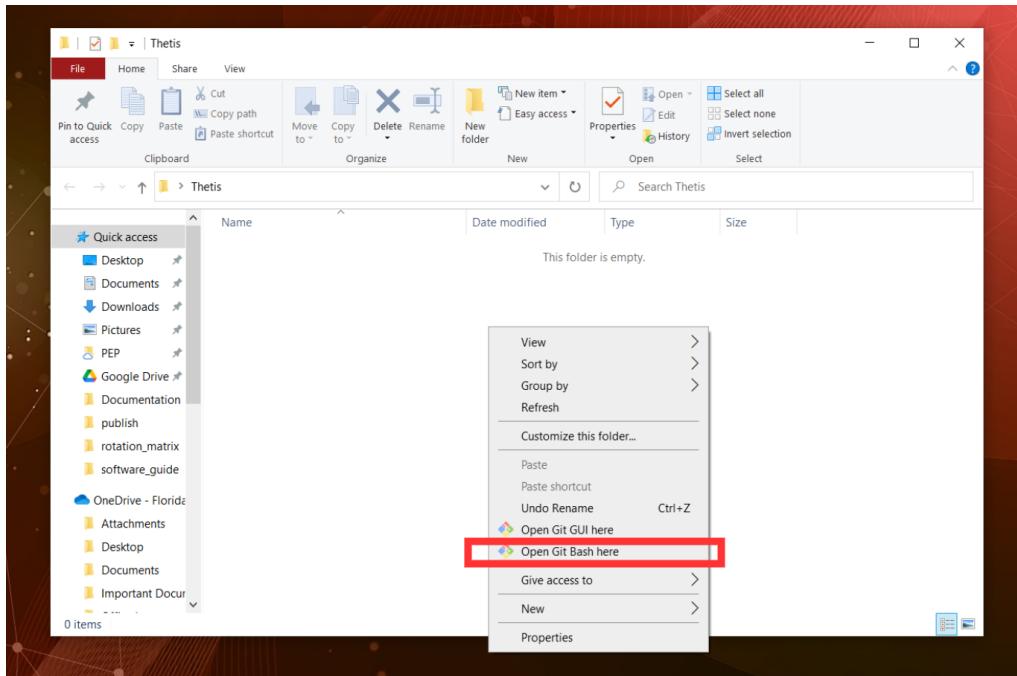


Figure L.2: Opening git bash in Windows File Explorer

Once the folder is cloned, open it in the command prompt and initialize the submodule repositories with the command:

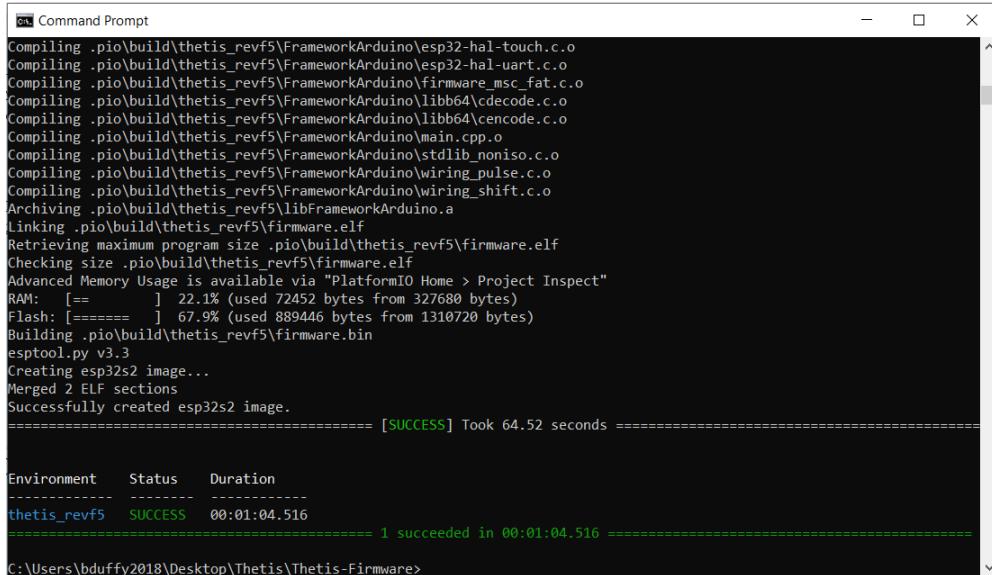
```
chdir Thetis-Firmware && git submodule update --init --recursive
```

L.1.4 Step Four: Check Proper PIO and Firmware Installation

Once the previous command finishes, execute the following command to compile the firmware and ensure everything is installed properly:

```
%USERPROFILE%\platformio\platformio.exe run -e thetis_revf5
```

If the code successfully compiles, you should see an output like this:



The screenshot shows a Windows Command Prompt window titled "Command Prompt". The output of the compilation process is displayed, starting with "Compiling .pio\build\thetis_revf5\FrameworkArduino\esp32-hal-touch.c.o" and ending with "C:\Users\bduffy2018\Desktop\Thetis\Thetis-Firmware>". The output indicates a successful build, with a total duration of 64.52 seconds.

```
Compiling .pio\build\thetis_revf5\FrameworkArduino\esp32-hal-touch.c.o
Compiling .pio\build\thetis_revf5\FrameworkArduino\esp32-hal-uart.c.o
Compiling .pio\build\thetis_revf5\FrameworkArduino\firmware_msc_fat.c.o
Compiling .pio\build\thetis_revf5\FrameworkArduino\libb64\cdecode.c.o
Compiling .pio\build\thetis_revf5\FrameworkArduino\libb64\cencode.c.o
Compiling .pio\build\thetis_revf5\FrameworkArduino\main.cpp.o
Compiling .pio\build\thetis_revf5\FrameworkArduino\stdlib_noniso.c.o
Compiling .pio\build\thetis_revf5\FrameworkArduino\wiring_pulse.c.o
Compiling .pio\build\thetis_revf5\FrameworkArduino\wiring_shift.c.o
Archiving .pio\build\thetis_revf5\libFrameworkArduino.a
Linking .pio\build\thetis_revf5\firmware.elf
Retrieving maximum program size .pio\build\thetis_revf5\firmware.elf
Checking size .pio\build\thetis_revf5\firmware.elf
Advanced Memory Usage is available via "PlatformIO Home > Project Inspect"
RAM: [==        ] 22.1% (used 72452 bytes from 327680 bytes)
Flash: [=====    ] 67.9% (used 889446 bytes from 1310720 bytes)
Building .pio\build\thetis_revf5\firmware.bin
esptool.py v3.3
Creating esp32s2 image...
Merged 2 ELF sections
Successfully created esp32s2 image.
===== [SUCCESS] Took 64.52 seconds =====

Environment      Status      Duration
-----           SUCCESS     00:01:04.516
===== 1 succeeded in 00:01:04.516 =====
C:\Users\bduffy2018\Desktop\Thetis\Thetis-Firmware>
```

Figure L.3: Output from a successful PlatformIO compile

If the command prompt shows that output, congratulations! You have successfully installed the basic development environment to upload new code to Thetis.

L.2 Advanced Development Environment Setup

FOR DEVELOPERS ONLY Now that the development environment is set up, you must fork the firmware repositories to your own GitHub account. This will preserve the original

source code and allow you to use GitHub to track any changes you make. Substantial changes can be made to the source repository through an appropriate pull request⁶.

To automatically configure the git environment, you can download and execute the `git_setup.bat` script. Or, you can follow along with the upcoming sections.

L.2.1 Configure Git Parameters

Open a new command prompt in the cloned `Thetis-Firmware/` folder. Before starting anything, we need to tell git CLI who you are. First, we call the following commands, replacing the `GITHUB_USER` and `GITHUB_EMAIL` with those for your GitHub account:

```
git config --global user.name GITHUB_USER  
git config --global user.email GITHUB_EMAIL
```

Then, we will authorize the GitHub CLI client to make requests to the GitHub servers on your behalf:

```
gh auth login -p "https" -w
```

This will give you a one-time passcode and ask to open a web browser to log into GitHub.

L.2.2 Fork Repositories and Reconfigure Submodules

Once you authorize GitHub CLI to use your account, we can begin forking the firmware repositories. First, execute the commands:

⁶<https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/proposing-changes-to-your-work-with-pull-requests/creating-a-pull-request-from-a-fork>

```
gh repo fork https://github.com/Legohead259/Thetis-Firmware.git --remote=true  
gh repo fork https://github.com/Legohead259/ThetisLib.git --remote=true  
gh repo fork https://github.com/Legohead259/Fusion-Arduino.git --remote=true  
gh repo fork https://github.com/Legohead259/Timer-Events-Arduino.git --remote=true  
gh repo fork https://github.com/Legohead259/xioAPI-Arduino.git --remote=true
```

Then, we must redirect the submodule URLs to the new repositories. Navigate to the `Thetis-Firmware/` root directory and call:

```
git submodule set-url lib/ThetisLib https://github.com/GITHUB_USER/ThetisLib.git
```

Again, replacing `GITHUB_USER` with your GitHub username. Then, navigate to the `Thetis-Firmware/lib/ThetisLib/` folder and execute the following:

```
git submodule set-url src/fusion https://github.com/GITHUB_USER/Fusion-Arduino.git  
git submodule set-url timing https://github.com/GITHUB_USER/Timer-Events-Arduino.git  
git submodule set-url src/xioAPI https://github.com/GITHUB_USER/xioAPI-Arduino.git  
git submodule sync
```

Return to the `Thetis-Firmware/` root directory and execute the submodule sync command again:

```
git submodule sync
```

This will redirect all submodule commands to your newly forked repositories and will allow you to make changes to any part of the firmware. You should now have your own dedicated git environment to modify Thetis and save those changes.

Appendix M

User Manual

This page left intentionally blank



Thetis User Manual

v1.0

October 31, 2023

Braidan Duffy

Contents

Acknowledgments	6
1 Overview	7
2 Hardware	8
2.1 Board	8
2.1.1 Interface Pads	9
2.2 Housing	10
2.2.1 IP67 rating	11
3 Technical Specification	12
3.1 Mechanical	12
3.1.1 Board	12
3.2 Temperature	12
3.2.1 No battery	12
3.2.2 With battery	13
3.3 Sensors	13
3.3.1 Gyroscope	13
3.3.2 Accelerometer	13
3.3.3 Magnetometer	14
3.4 AHRS	14
3.4.1 Update rate	14
3.4.2 Static accuracy	14
3.5 Data logger capacity	15
4 Calibration	15
4.1 Inertial sensors	15
4.2 Magnetometer	15
5 Human Interfaces	16
5.1 Buttons	16
5.2 Primary RGB LEDs	17
5.2.1 Booting (Purple)	17
5.2.2 Standby (Yellow)	17
5.2.3 Ready, No GPS (Blue)	17
5.2.4 Logging, No GPS (Blue)	18
5.2.5 Ready, GPS (Green)	18
5.2.6 Logging, GPS (Green)	19
5.2.7 Error (Red and Yellow)	19
5.2.8 User control	21
5.3 Secondary Diagnostic LED	21
6 Data logger	21
6.1 Start and stop	21
6.2 File name	22
6.3 File contents	22
7 Communication protocol	22
7.1 Command messages	22
7.1.1 Read all settings command	22
7.1.2 Read JSON file command	23
7.1.3 Read setting command	23
7.1.4 Write setting command	23
7.1.5 Default command	23
7.1.6 Apply command	23

7.1.7	Save command	23
7.1.8	Read time command	24
7.1.9	Write time command	24
7.1.10	Ping command	24
7.1.11	Ping response	24
7.1.12	Reset command	24
7.1.13	Shutdown command	25
7.1.14	Strobe command	25
7.1.15	Colour command	25
7.1.16	Initialise AHRS command	25
7.1.17	Heading command	25
7.1.18	Serial accessory command	25
7.1.19	Note command	25
7.1.20	Format command	26
7.1.21	Self-test command	26
7.1.22	Self-test response	26
7.1.23	Bootloader command	26
7.1.24	Factory command	27
7.1.25	Erase command	27
7.2	Data messages	27
7.2.1	Byte stuffing	27
7.2.2	Inertial message	28
7.2.3	Magnetometer message	29
7.2.4	Position Message	29
7.2.5	Quaternion message	30
7.2.6	Rotation matrix message	30
7.2.7	Euler angles message	31
7.2.8	Linear acceleration message	32
7.2.9	Earth acceleration message	32
7.2.10	AHRS status message	33
7.2.11	High-g accelerometer message	34
7.2.12	Temperature message	34
7.2.13	Battery message	35
7.2.14	RSSI message	36
7.2.15	Serial accessory message	36
7.2.16	Notification message	36
7.2.17	Error message	37
8	Sample rates, message rates, and timestamps	37
8.1	Sample rates	37
8.2	Message rates	38
8.3	Sample averaging	38
8.4	Timestamps	38
8.5	Synchronisation	38
9	Network announcement message	38
10	Device Settings	39
10.1	Individual Settings	39
10.1.1	Calibration date (read-only)	39
10.1.2	System clock calibration (read-only)	39
10.1.3	RTC calibration (read-only)	40
10.1.4	Battery voltmeter sensitivity (read-only)	40
10.1.5	Gyroscope misalignment (read-only)	40
10.1.6	Gyroscope sensitivity (read-only)	40
10.1.7	Gyroscope offset (read-only)	40

10.1.8 Accelerometer misalignment (read-only)	40
10.1.9 Accelerometer sensitivity (read-only)	41
10.1.10 Accelerometer offset (read-only)	41
10.1.11 Soft iron matrix (read-only)	41
10.1.12 Hard iron offset (read-only)	41
10.1.13 High-g accelerometer misalignment (read-only)	41
10.1.14 High-g accelerometer sensitivity (read-only)	41
10.1.15 High-g accelerometer offset (read-only)	42
10.1.16 Device name	42
10.1.17 Serial number (read-only)	42
10.1.18 Firmware version (read-only)	42
10.1.19 Bootloader version (read-only)	42
10.1.20 Hardware version (read-only)	42
10.1.21 Serial mode	43
10.1.22 Serial baud rate	43
10.1.23 Serial RTS/CTS enabled	43
10.1.24 Serial accessory number of bytes	43
10.1.25 Serial accessory termination byte	43
10.1.26 Serial accessory timeout	43
10.1.27 Wireless mode	44
10.1.28 Wireless firmware version (read-only)	44
10.1.29 External antennae enabled	44
10.1.30 Wi-Fi region	44
10.1.31 Wi-Fi MAC address (read-only)	44
10.1.32 Wi-Fi IP address (read-only)	44
10.1.33 Wi-Fi client SSID	45
10.1.34 Wi-Fi client key	45
10.1.35 Wi-Fi client channel	45
10.1.36 Wi-Fi client DHCP enabled	45
10.1.37 Wi-Fi client IP address	45
10.1.38 Wi-Fi client netmask	45
10.1.39 Wi-Fi client gateway	46
10.1.40 Wi-Fi AP SSID	46
10.1.41 Wi-Fi AP key	46
10.1.42 Wi-Fi AP channel	46
10.1.43 Wi-Fi AP IP address	46
10.1.44 TCP port	46
10.1.45 UDP IP address	47
10.1.46 UDP send port	47
10.1.47 UDP receive port	47
10.1.48 UDP low latency	47
10.1.49 Synchronisation enabled	47
10.1.50 Synchronisation network latency	47
10.1.51 Bluetooth address (read-only)	48
10.1.52 Bluetooth name	48
10.1.53 Bluetooth pin code	48
10.1.54 Bluetooth discovery mode	48
10.1.55 Bluetooth paired address (read-only)	48
10.1.56 Bluetooth paired link key (read-only)	48
10.1.57 Data logger enabled	49
10.1.58 Data logger file name prefix	49
10.1.59 Data logger file name time enabled	49
10.1.60 Data logger file name counter enabled	49
10.1.61 Data logger max file size	49
10.1.62 Data logger max file period	49
10.1.63 Axes alignment	50

10.1.64 Gyroscope offset correction enabled	50
10.1.65 AHRS axes convention	50
10.1.66 AHRS gain	50
10.1.67 AHRS ignore magnetometer	50
10.1.68 AHRS acceleration rejection enabled	50
10.1.69 AHRS magnetic rejection enabled	51
10.1.70 Binary mode enabled	51
10.1.71 USB data messages enabled	51
10.1.72 Serial data messages enabled	51
10.1.73 TCP data messages enabled	51
10.1.74 UDP data messages enabled	51
10.1.75 Bluetooth data messages enabled	52
10.1.76 Data logger data messages enabled	52
10.1.77 AHRS message type	52
10.1.78 Inertial message rate divisor	52
10.1.79 Magnetometer message rate divisor	52
10.1.80 AHRS message rate divisor	52
10.1.81 High-g accelerometer message rate divisor	53
10.1.82 Temperature message rate divisor	53
10.1.83 Battery message rate divisor	53
10.1.84 RSSI message rate divisor	53
10.1.85 FTP! enabled	53
10.1.86 FTP! username	53
10.1.87 Log print level	54
10.1.88 Log file level	54
10.1.89 GPS RTC sync enabled	54
10.1.90 Accelerometer range	54
10.1.91 Gyroscope range	54
10.1.92 IMU data rate	54
10.1.93 Magnetometer performance mode	55
10.1.94 Magnetometer operation mode	55
10.1.95 Magnetometer data rate	55
10.1.96 Magnetometer range	55
10.1.97 Gauge reset voltage	55
10.1.98 Gauge activity threshold	55
10.1.99 Gauge hibernation threshold	56
10.1.100Gauge alert minimum voltage	56
10.1.101Gauge alert maximum voltage	56
10.1.102Fusion update rate	56
Glossary	57
Document version history	60
Disclaimer	61

Acknowledgements

I would like to acknowledge and thank Dr. Sebastian Madgwick, his company xio-Technologies, and his staff for their efforts in developing small embedded data loggers. The x-IMU3 became a major inspiration when it was announced earlier in 2023 and Seb was willing to chat with me about Thetis's design and its flaws. He has also been open to me using the x-IMU3 API on my board and the x-IMU3 user manual as a basis for this manual. His contributions and advice directly influenced version 2.0 of the firmware and the hardware revision F6. I am grateful for his correspondence and look forward to continuing our discussions on a variety of ideas.

1 Overview

Thetis is a nine Degree of Freedom (DOF) Inertial Measurement Unit (IMU) that is capable of reporting the acceleration, rotation rate, orientation, and position of a body. It was developed as part of a thesis project for the Ocean Engineering Department at the Florida Institute of Technology. The device is designed to be used in classroom and laboratory environments or out in the field, wherever inertial data logging is desired.

Thetis can communicate with a host computer over a USB or Wi-Fi connection, streaming data in real-time to an operator. For more embedded applications, Thetis has an on-board battery and microSD card storage, allowing it to remotely data log without an operator. A simple user interface allows for a single operator to deploy Thetis in any condition.

Sensors

- Gyroscope, $\pm 2000^{\circ}/\text{s}$, 100 Hz
- Accelerometer, $\pm 32 \text{ g}$, 100 Hz
- Magnetometer, $\pm 16 \text{ gauss}$, 100 Hz

Calibration

- 15-parameter calibration for: axis sensitivity, axis offset, inter-axis misalignment, and package misalignment.
- Hard-iron and soft-iron calibration
- On-board gyroscope bias correction algorithm

AHRS

- Algorithm outputs:
 - Quaternion
 - Rotation matrix¹
 - Euler angles
 - Linear acceleration¹
 - Earth acceleration¹
- Linear acceleration rejection¹
- Magnetic distortion rejection¹
- 100 Hz update rate
- Static accuracy:
 - 0.5° RMS inclination²
 - 1° RMS heading²

Communication

- USB (CDC)
- TCP (Wi-Fi)¹
- UDP (Wi-Fi)

Wi-Fi

- Client and AP mode
- Single band (2.4 GHz)
- WPA/WPA2-Personal

Data logging

- Supports microSDs up to 32 GB³
- Start/stop logging remotely¹
- Start/stop logging with physical button
- USB download⁴
- Wi-Fi download⁵
- CSV output

Battery

- Internal battery charged by USB
- 9 hours data logging⁶
- 4 hours Wi-Fi client⁷

Housing

- IP67
- Chassis mount via two M3 screws or bolts

Software GUI

- Connect to multiple xio-compatible IMUs (e.g. x-IMU3, Thetis)
- Real-time data graphs and 3D view
- Log data to CSV
- Windows, macOS, Ubuntu

Software API

- Rust, C, C++, C#, Python
- Code examples for other languages available

¹Supported, but not yet implemented.

²Remains to be validated.

³The product is supplied with an 4 GB microSD that can be upgraded by the user.

⁴USB downloading is currently in development and not yet supported.

⁵Wi-Fi downloading is currently in development and not yet supported.

⁶Estimated

⁷Estimated, assuming constant transmission.

2 Hardware

2.1 Board

Board components are annotated in Figure 1. A detailed mechanical drawing describing the board dimensions is available in the Appendix.

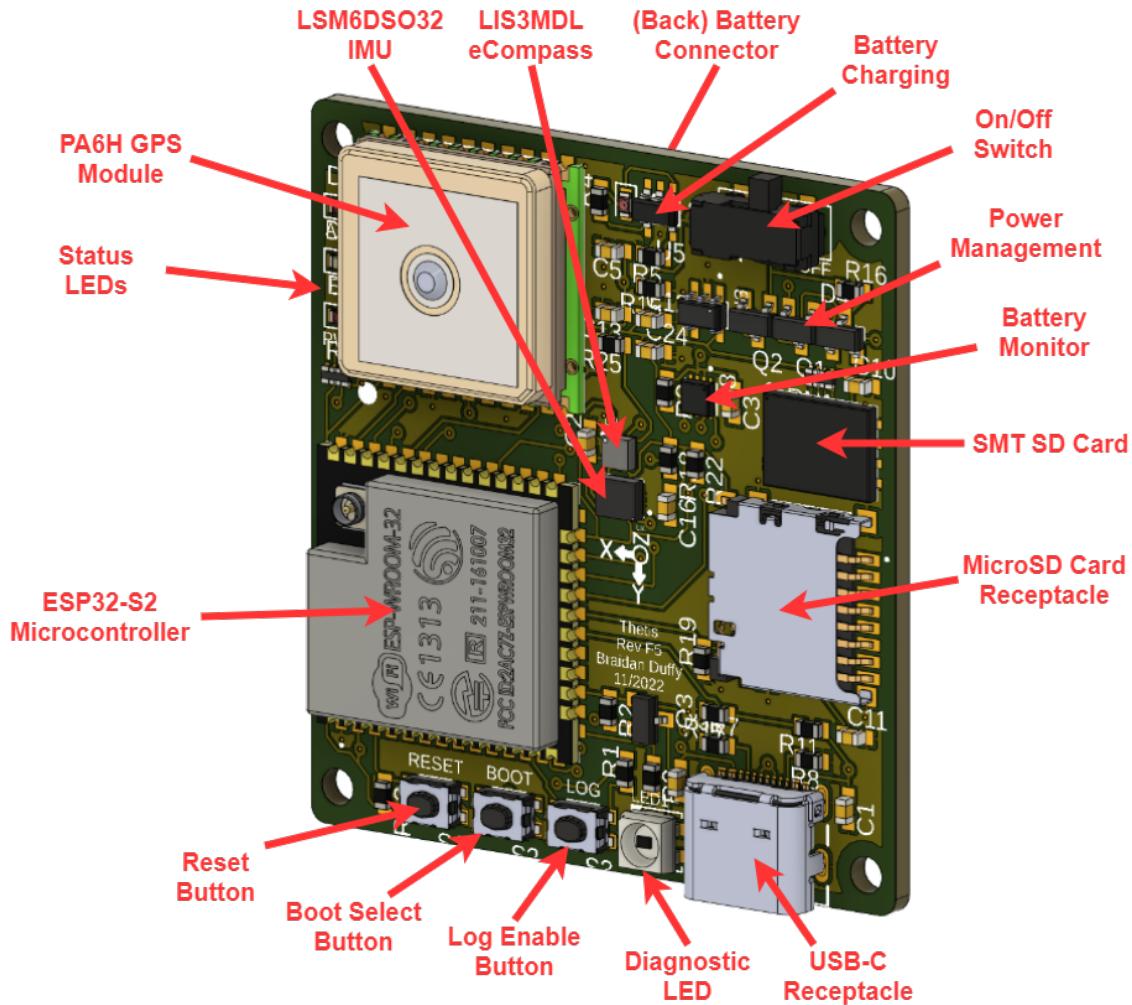


Figure 1: Revision F5 of Thetis with callouts for important components

2.1.1 Interface Pads

The PCB interface is on the back of the board beneath the EPS32-S2 microcontroller. Its pinout is annotated in Figure 2. The interface pads are just bare copper that are routed to their appropriate components. This allows for easier programming, diagnostics, and probing of the board during assembly and testing. It is recommended to use a carrier board and pogo pins to break these pads out for use. A mechanical drawing for the pad layout is in the Appendix.

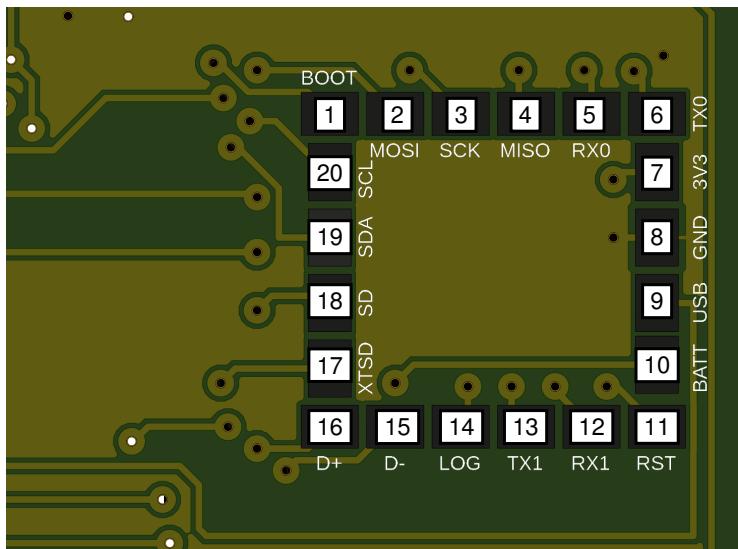


Figure 2: PCB interface pinout

Pad	Name	Bus	Type
1	Boot Select	GPIO	Input
2	MOSI	SPI	Output
3	SCK	SPI	Clock
4	MISO	SPI	Input
5	RX0	UART0	Input
6	TX0	UART0	Output
7	3V3		Power
8	GND		Power
9	VUSB		Power
10	VBATT		Power

Table 1: PCB interface pinout

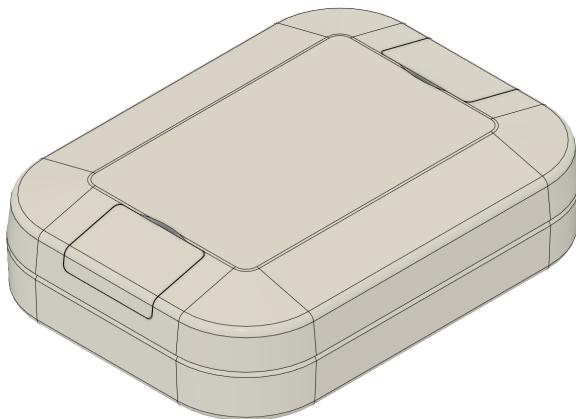
Pad	Name	Bus	Type
11	Reset	GPIO	Input
12	RX1	UART1	Input
13	TX1	UART1	Output
14	Log	GPIO	Input
15	D-	USB	Data
16	D+	USB	Data
17	XTSD CS	SPI	Output
18	microSD CS	SPI	Output
19	SDA	I2C	Data
20	SCL	I2C	Clock

Table 2: PCB interface pinout

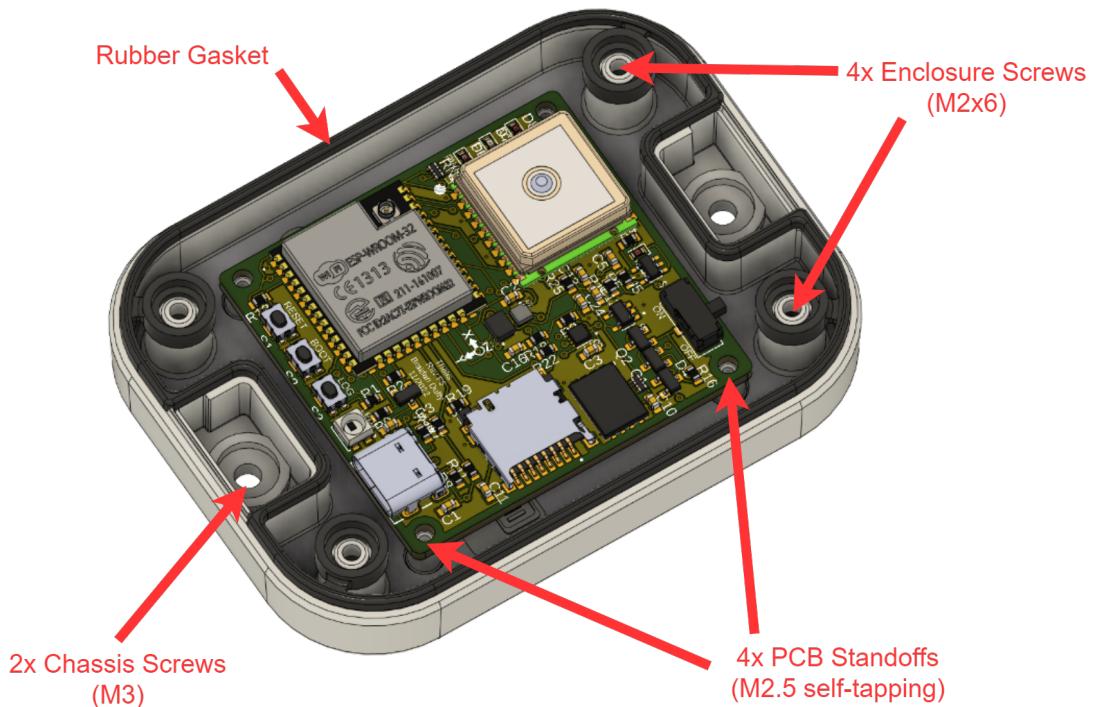
Warning - Incorrect connections to the PCB interface may cause permanent damage. This interface should only be used by an experienced engineer with the appropriate carrier board.

2.2 Housing

The housing components are annotated in Figure 3. A detailed mechanical drawing describing the housing dimensions is available in the Appendix.



(a) Sealed enclosure ready for deployment



(b) Enclosure with the top removed

Figure 3: Thetis enclosure

2.2.1 IP67 rating

The Ingress Protection 67 (IP67) rating is an international standard that describes the ability of the housing to protect against the ingress of solid particles and water. The first digit, 6 indicates complete protection against dust and solid particles. The second digit, 7 indicates protection from water for a maximum depth of 1 meter for up to 30 minutes.

In practical terms, this means that the housing can be used outdoors in all weather conditions and that it will survive accidental or temporary submersion in water. The housing has been tested in submerged applications up to 20-feet for 30 minutes, however typical applications should not exceed the IP rating. Always check that the gasket is in place, the 4 enclosure screws are suitably tightened, and the case is not cracked.



Warning - Over-torquing the enclosure screws may result in the plastic around them cracking. The enclosure screws should be finger tight with an extra quarter turn. Additionally, all four screws must be present for the case to be sealed - any less and the enclosure will leak when submerged!

3 Technical Specification

3.1 Mechanical

3.1.1 Board

Characteristic	Value	Notes
Size	1.82 × 1.74 × 0.48 in	1
Weight	3.2 g	-

Table 3: Board mechanical specification

Notes

1. A detailed mechanical drawing describing the board dimensions and locations of key components is available in the Appendix.

Characteristic	Value	Notes
Size	3.15 × 2.36 × 0.79 in	1
Weight	50 g	-

Table 4: Housing mechanical specification

Notes

1. A detailed mechanical drawing describing the housing dimensions and locations of key components is available in the Appendix.

3.2 Temperature

3.2.1 No battery

Characteristic	Value	Notes
Operating	-40°C to 85°C	1, 2
Storage	-40°C to 105°C	-

Table 5: Temperature specification (no battery)

Notes

1. The operating temperature of the device will always be greater than the surroundings due to heat generated by electronics.
2. The specified accuracy of the device is not achieved over the full operating temperature range. See Section 4 on page 15 for more information.

3.2.2 With battery

Characteristic	Value	Notes
Operating (discharging)	-20 °C to 60 °C	1, 2
Operating (charging)	0 °C to 45 °C	1, 2, 3
Storage	-20 °C to 25 °C	-

Table 6: Temperature specification (with battery)

Notes

1. The operating temperature of the device will always be greater than the surroundings due to heat generated by electronics.
2. The specified accuracy of the device is not achieved over the full operating temperature range. See Section 4 on page 15 for more information.
3. Charging at temperatures below 0 °C will reduce the capacity and cycle life of the battery.

3.3 Sensors

3.3.1 Gyroscope

Characteristic	Value	Notes
Range	±2000 °/s	-
Resolution	16-bit, 0.061 °/s	-
Sample rate	100 Hz ±0.3%	1

Table 7: Gyroscope specification

Notes

1. Each sample includes a timestamp for a reliable measurement of time independent of the sample rate error. See Section 8 on page 37 for more information.

3.3.2 Accelerometer

Characteristic	Value	Notes
Range	±32 g	-
Resolution	16-bit, 488 µg	-
Sample rate	100 Hz ±0.3%	1
Accuracy at 1 g	±5 mg	2, 3

Table 8: Accelerometer specification

Notes

1. Each sample includes a timestamp for a reliable measurement of time independent of the sample rate error. See Section 8 on page 37 for more information.
2. The accelerometer error is evaluated as the deviation of the measured magnitude of gravity for a 360° rotation around each axis aligned to the horizontal. The magnitude is calculated as $\sqrt{x^2 + y^2 + z^2}$.

3. Accuracy is specified for the calibrated temperature only. See Section 4 on the next page for more information.

3.3.3 Magnetometer

Characteristic	Value	Notes
Range	± 16 Gauss	-
Sample rate	80 Hz $\pm 8\%$	1
Noise	Unknown	-
Accuracy at 1 a.u.	Unknown	2, 3, 4

Table 9: Magnetometer specification

Notes

1. Each sample includes a timestamp for a reliable measurement of time independent of the sample rate error. See Section 8 on page 37 for more information.
2. The calibrated magnetometer units are arbitrary units (a.u.). 1 a.u. is equal to the magnitude of the ambient magnetic field during calibration, approximately 45 μ T.
3. The magnetometer error is evaluated as the deviation of the measured magnitude of the ambient magnetic field for a 360° rotation around each axis aligned to the vertical. The magnitude is calculated as $\sqrt{x^2 + y^2 + z^2}$.
4. Accuracy is specified for the calibrated temperature only. See Section 4 on the next page for more information.

3.4 AHRS

3.4.1 Update rate

Characteristic	Value	Notes
Update rate	64 Hz $\pm 0.3\%$	1, 2

Table 10: AHRS update rate

Notes

1. Each update includes a timestamp for a reliable measurement of time independent of the update rate error. See Section 8 on page 37 for more information.
2. The Attitude Heading Reference System (AHRS) update rate is fixed independent of message rate settings. AHRS outputs are not averaged when the message rate is less than the update rate.

3.4.2 Static accuracy

Characteristic	Value	Notes
Inclination	0.5° RMS	1, 2
Heading	1° RMS	1, 2

Table 11: AHRS static accuracy

Notes

1. Static accuracy is specified as the Root Mean Square (RMS) error for a 360° rotation around each axis.
2. Accuracy is specified for the calibrated temperature only. See Section 4 for more information.

3.5 Data logger capacity

The data logger capacity can be determined with the following equation:

$$t_{\text{samples}} = \frac{N_{\text{storage}}[\text{Bytes}]}{198[\text{Bytes}] \times 64[\text{s}^{-1}] \times 3600 [\frac{\text{s}}{\text{h}}]} \quad (1)$$

This yields the following times the data logger can record for with a given microSD card size:

Size	Value
1 GB	22 hours
4 GB	88 hours
8 GB	175 hours
16 GB	350 hours
32 GB	701 hours

Table 12: Data logger record times based on capacity of the microSD card

4 Calibration

Each Thetis instrumentation board is calibrated during production to achieve the specified accuracy. The calibration process uses custom equipment and open-source algorithms to calculate calibration parameters specific to each Thetis board. See Chapter 4 in the thesis for more information. Calibration is performed at room temperature. Accuracy will be reduced for operating temperatures that deviate from this temperature.

4.1 Inertial sensors

The inertial sensors are the gyroscope and accelerometer. Each inertial sensor is calibrated for axis sensitivity, axis offset, inter-axis misalignment, and package misalignment. The inertial calibration model is described by Equation (2) where i_c is the calibrated inertial measurement obtained from the uncalibrated inertial measurement, i_u , given the misalignment matrix, M , the sensitivity diagonal matrix, S , and the offset vector, \mathbf{b} . The inertial calibration model is expanded as Equation (3) to express the model as 15 scalar quantities. The units of i_c , i_u , and \mathbf{b} are degrees per second for the gyroscope, and g for the accelerometer. M and S are ratios and therefore have no units. The calibration parameters M , S , and \mathbf{b} for each inertial sensor can be accessed as device settings.

$$i_c = MS(i_u - \mathbf{b}) \quad (2)$$

$$\begin{bmatrix} i_{cx} \\ i_{cy} \\ i_{cz} \end{bmatrix} = \begin{bmatrix} m_{xx} & m_{xy} & m_{xz} \\ m_{yx} & m_{yy} & m_{yz} \\ m_{zx} & m_{zy} & m_{zz} \end{bmatrix} \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{bmatrix} \left(\begin{bmatrix} i_{ux} \\ i_{uy} \\ i_{uz} \end{bmatrix} - \begin{bmatrix} b_x \\ b_y \\ b_z \end{bmatrix} \right) \quad (3)$$

4.2 Magnetometer

The magnetometer is calibrated for soft and hard iron distortion. Soft iron characteristics are distortions that alter the intensity and direction of the magnetic field measured by the magnetometer. Soft iron calibration also accounts for magnetometer axis sensitivity, inter-axis misalignment, and package misalignment. Hard

iron characteristics are unintended magnetic fields generated by the device that offset magnetometer measurements. Hard iron calibration also accounts for the magnetometer axis offset.

The magnetometer calibration model is described by Equation (4) where \mathbf{m}_c is the calibrated magnetometer measurement obtained from the uncalibrated magnetometer measurement, \mathbf{m}_u , given the soft iron matrix, W^{-1} , the hard iron vector, \mathbf{v} . The magnetometer calibration model is expanded as Equation (5) to express the model as 12 scalar quantities. The units of \mathbf{m}_c , \mathbf{m}_u , and \mathbf{v} are a.u.. W^{-1} is a ratio and therefore has no units. The calibration parameters W^{-1} , \mathbf{v} can be accessed as device settings.

$$\mathbf{m}_c = W^{-1} \mathbf{m}_u - \mathbf{v} \quad (4)$$

$$\begin{bmatrix} m_{c,x} \\ m_{c,y} \\ m_{c,z} \end{bmatrix} = \begin{bmatrix} w_{xx} & w_{xy} & w_{xz} \\ w_{yx} & w_{yy} & w_{yz} \\ w_{zx} & w_{zy} & w_{zz} \end{bmatrix} \begin{bmatrix} m_{u,x} \\ m_{u,y} \\ m_{u,z} \end{bmatrix} - \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} \quad (5)$$

5 Human Interfaces

There are multiple human interface components on the Thetis board, as shown in Figure 4.

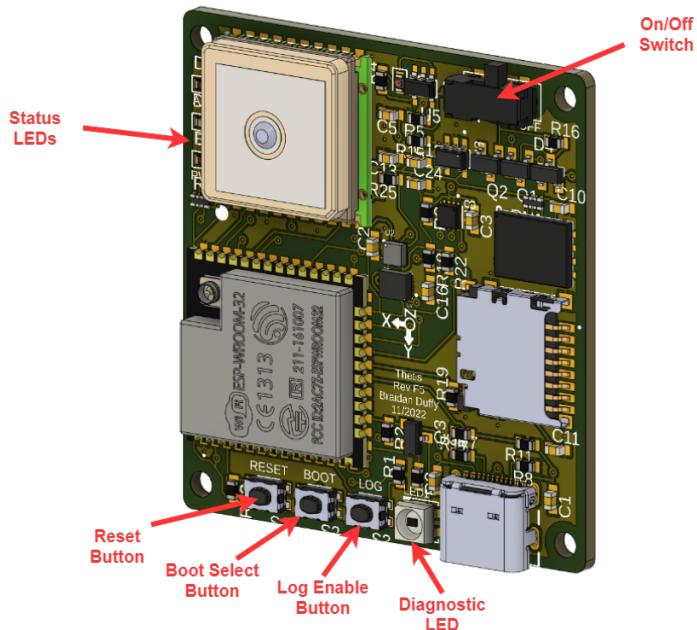


Figure 4: Thetis Revision F5 with callouts for human interface components

5.1 Buttons

Along the bottom edge are three buttons: reset, boot select, and log enable.

The reset button will perform a power-on reset of the microcontroller when pressed, restarting the firmware and clearing any errors.

The boot select button will put the microcontroller into the bootloader (safe/programming) mode if pressed during a reset or power cycle.

The log enable button will start or stop logging when pressed for half of a second. The current logging status is shown by the LEDs.

5.2 Primary RGB LEDs

The main diagnostic Red Green Blue (RGB) Light-Emitting Diode (LED) indicates the mode and status of Thetis using different colors and flashing behaviors.

5.2.1 Booting (Purple)

A steady purple LED, as shown in Figure 5 indicates that Thetis is switched on and booting.

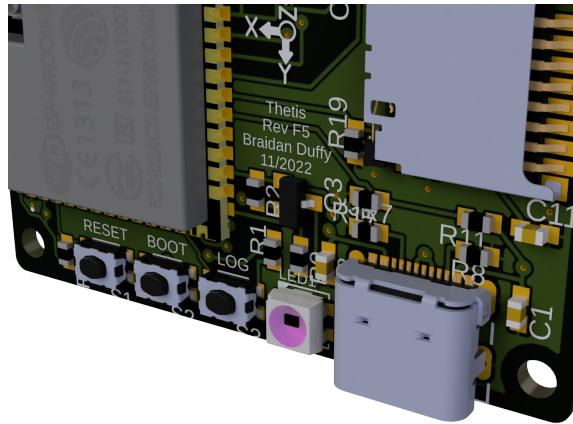


Figure 5: a steady purple LED indicating that Thetis is switched on and booting

5.2.2 Standby (Yellow)

A steady yellow LED, as shown in Figure 6 indicates that Thetis is switched on and in standby mode. This mode is active until the device passes a self-test, its sensors report that they are calibrated, and the fusion algorithm is online and stable. Currently, this mode is bypassed and ignored.

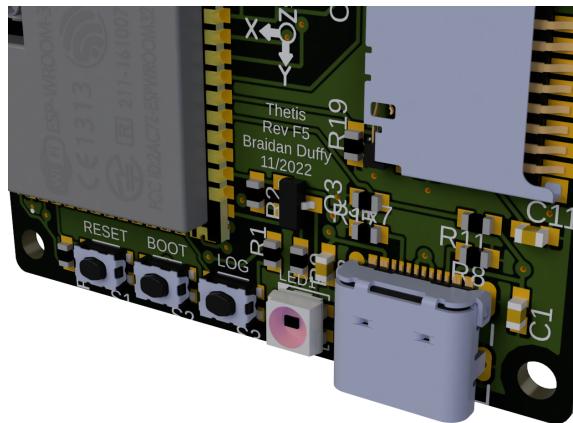
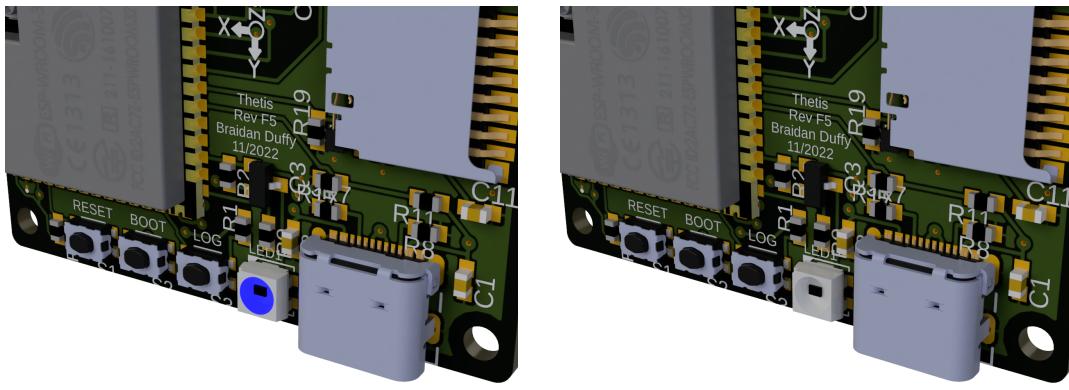


Figure 6: A steady yellow LED indicating that Thetis is switched on and in standby mode

5.2.3 Ready, No GPS (Blue)

A blinking (once per second) blue LED, as shown in Figure 7 on the next page indicates that Thetis is online, calibrated, and ready to start logging. However, it has not yet acquired a GPS fix. This mode will still allow logging, but the position messages will either not be populated or not accurate.



(a) The LED is blue for approximately 250 milliseconds

(b) Then turns off for 250 milliseconds

Figure 7: A blinking blue LED indicating that Thetis is switched on and ready to log data without a GPS fix

5.2.4 Logging, No GPS (Blue)

A steady blue LED, as shown in Figure 8 indicates that Thetis is switched on and logging data without a GPS fix. This mode will still log data as expected, but the position messages will either not be populated or not accurate.

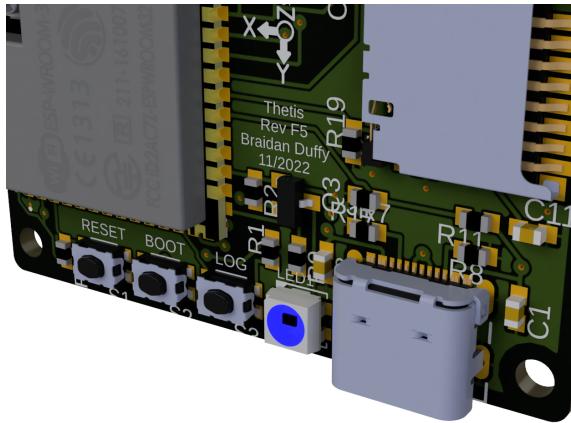
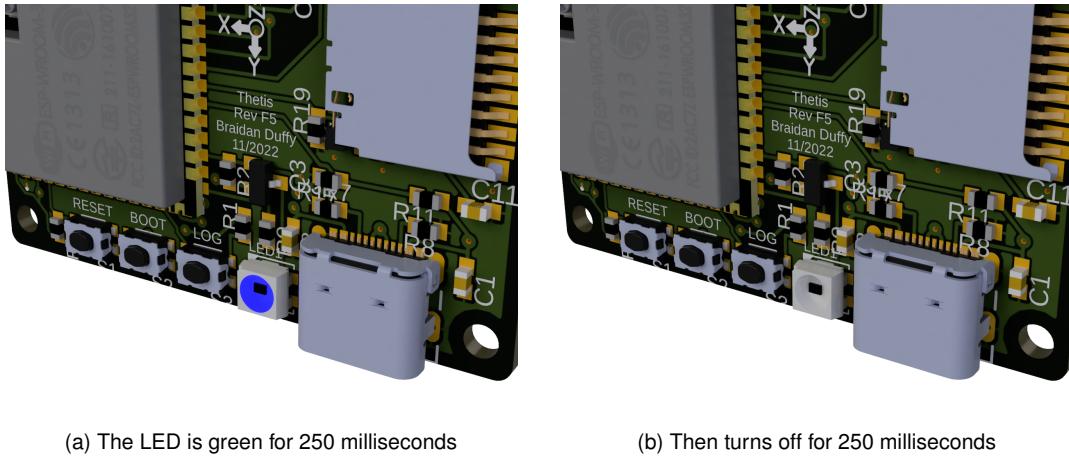


Figure 8: A steady blue LED indicating that Thetis is switched on and logging without a GPS fix

5.2.5 Ready, GPS (Green)

A blinking (once per second) green LED, as shown in Figure 9 on the next page indicates that Thetis is online, calibrated, and ready to start logging with a valid GPS fix. This will populate the position message with reasonably-accurate GPS data.



(a) The LED is green for 250 milliseconds

(b) Then turns off for 250 milliseconds

Figure 9: A blinking green LED indicating that Thetis is switched on and ready to log data with a GPS fix

5.2.6 Logging, GPS (Green)

A steady green LED, as shown in Figure 10 indicates that Thetis is switched on and logging data with a GPS fix. Position messages will be populated with reasonably-accurate GPS data in this mode.

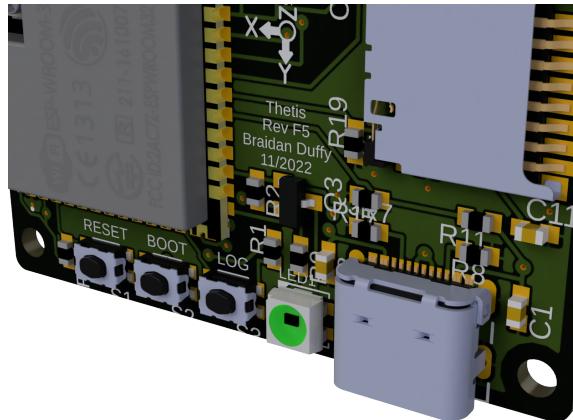
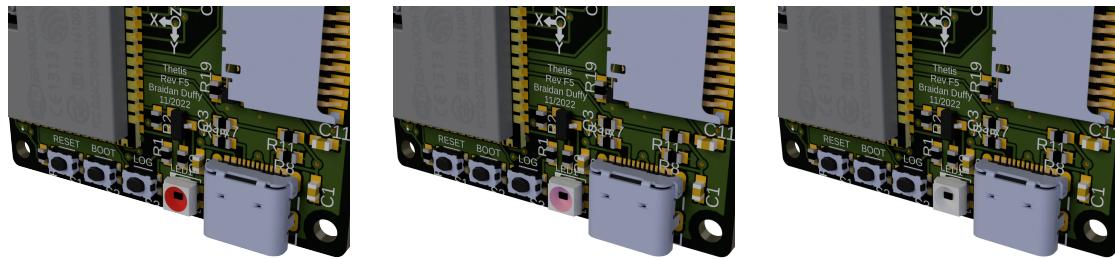


Figure 10: A steady green LED indicating that Thetis is switched on and logging with a GPS fix

5.2.7 Error (Red and Yellow)

When Thetis enters an error state, the main RGB LED will begin flashing red and yellow error codes based on the error type. The red blinks will precede the yellow blinks and there will be a short interval between code flashes. In the error state, Thetis will lock up and lose all functionality until the battery dies. The device will require a full power cycle or reset to clear the error code. In-depth diagnostic information may be available if data is being streamed to a host computer through the USB port and serial monitor terminal (e.g. PuTTY).



(a) The LED is red with 250 millisecond blinks for a certain number of times
 (b) Then turns yellow with 250 millisecond blinks for a certain number of times
 (c) Then turns off for one second

Figure 11: A blinking red and yellow LED indicates that Thetis is switched on and in an error state according to Table 13

Error	Red	Yellow	Description
General	1	1	General failure state; unhandled or unknown error
Settings	1	2	Board is initialized with or encounters invalid settings
Low Battery	2	1	Battery voltage is below the threshold limit and the system has not automatically shutdown
Battery Monitor	2	2	The battery monitoring IC fails to initialize
Temperature	2	3	The reported system temperature exceeds a threshold ⁸
Card Mount	3	1	The data log filesystem fails to mount to the microSD card
Card Type	3	2	The data log filesystem initializes, but reports an incorrect microSD card type or format
File Operations	3	3	A requested filesystem operation fails to execute
Wi-Fi Radio	4	1	The Wi-Fi radio fails to initialize or encounters errors
BlueTooth Radio	4	2	The BlueTooth radio fails to initialize or encounters errors ⁸
GPS Radio	4	3	The GPS radio fails to initialize or encounters an error
IMU	5	1	The IMU fails to initialize or encounters an error
Magnetometer	5	2	The magnetometer fails to initialize or encounters an error
Fusion	5	3	The sensor fusion algorithm encounters an error ⁸
CANbus	6	1	The CANbus transceiver fails to initialize or encounters an error ⁸
Device ID	6	2	The device ID is not properly configured ⁸

Table 13: Thetis error code table

⁸not currently implemented

5.2.8 User control

The on-board RGB LED can be controlled by the user using the strobe and colour commands. See Section 7.1.14 on page 25 and Section 7.1.15 on page 25 for more information.

5.3 Secondary Diagnostic LED

In addition to the main RGB LED, Thetis also has four other monochromatic diagnostic LEDs on the left and top edge of the board. From top to bottom, these LEDs are for battery charging, activity, GPS fix, and power. Their locations are highlighted in Figure 12

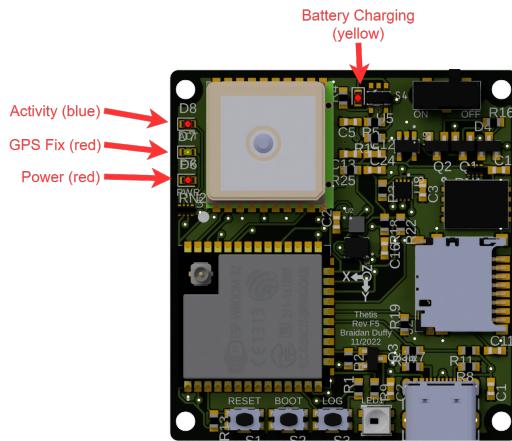


Figure 12: Additional monochromatic diagnostic LEDs present on Thetis

The battery charging LED illuminates orange when the battery is plugged in, Universal Serial Bus (USB) is connected, and the battery is not at 4.2V. The board does not need to be on to charge the battery.

The activity LED is blue and typically only illuminated when the device is logging. This behavior can be changed in the firmware.

The Global Positioning System (GPS) fix LED blinks red once per second when the GPS is acquiring a signal. Once the signal is acquired, it blinks quickly once every 15 seconds.

The power LED illuminates red when the 3V3 bus on the board is active; nominally when the battery or USB is plugged in and the switch turned on.

6 Data logger

Thetis can function as a stand-alone data logger by streaming real-time data to a file on the micro Secure Digital (microSD). Files created by the data logger use the .bin extension and can be downloaded from the device to be converted to Comma-Separated Values (CSV) files using the product software.

The data logger will create a new file in the root directory on the microSD each time the device boots. Files will never be overwritten or deleted by the data logger. If the microSD becomes full then the data logger will stop and Thetis will indicate an error.

6.1 Start and stop

The data logger is started or stopped by the “Log Enable” button (See Section 5.1 on page 16 for more information.). As soon as the button is held for half of a second, the device will immediately begin logging. When the operator wishes to stop logging, the “Log Enable” button can be held again for another half second or until the “Activity” LED and RGB LED reflect the “Ready” state.



Warning - The data file is not written to the microSD card filesystem until the device is commanded to stop logging. DO NOT reset or power off the device when logging, otherwise the data may not be saved or will be corrupted!

6.2 File name

The file name format is “log_CCC.bin” where “CCC” is a counter. The counter runs between “000” and “999”. On startup, Thetis scans the microSD card and increments the counter until the next available number is encountered. When it exceeds its maximum value, the system will throw a filesystem initialization error. Even if the device is powered on, but does not log, a new log file will be created. In testing environments, this may lead to multiple empty files that can obscure an actual desired data file.

6.3 File contents

The contents of the file is a byte stream as per the communication protocol described in Section 7.

7 Communication protocol

All communication interfaces use the same communication protocol. The byte stream is therefore identical for USB, User Datagram Protocol (UDP), and the files created by the data logger. The communication protocol consists of two message types:

- Command messages
- Data messages

All messages are terminated by a Line Feed (LF) control character. This termination byte will not appear anywhere else in a message and so can be used to divide a byte stream into individual messages. Table 14 describes the different ways that the control character LF may be referred to throughout this document.

Control character	Abbreviation	String	Hex	Decimal
Line Feed	LF	“\n”	0x0A	10

Table 14: Control characters LF representations

The first byte of a message indicates the message type. Command messages start with the character “{” (0x7B in hex, 123 in decimal). Data messages start with either an uppercase character or a byte value greater than 0x80 (128 in decimal) depending on the message.

7.1 Command messages

Command messages are sent to Thetis to read and write settings and execute commands. All command messages are a JavaScript Object Notation (JSON) object containing a single key/value pair, terminated by the control character LF. The control character LF must not appear anywhere else in a command message. Thetis will acknowledge each received command message by sending a command message with the same key to the host.

The key used by command messages sent to Thetis is **case sensitive** and must be exactly as stated in the API. For example, “serialNumber” will work, but “Serial Number” and “serial_number” are not valid keys for a command message to read the device serial number.

7.1.1 Read all settings command

The read all settings command is sent to Thetis to read all the setting values. The key is “readAll” and the value is null. See Section 10.1 on page 39 for a complete list of settings. Thetis will acknowledge a read all settings command by sending a write setting command to the host for each of its settings.

Example: {"readAll":null}\n

7.1.2 Read JSON file command

The read JSON file command is sent to Thetis to read out the settings file stored in its non-volatile memory. This file is a JSON file containing all the settings as key-value pairs. See Section 10.1 on page 39 for a complete list of settings. Thetis will acknowledge a read JSON file command by sending a stream reading out the entire contents of the settings file.

Example: {"readJSON":null}\n

7.1.3 Read setting command

The read setting command is sent to Thetis to read a setting value. The key is the setting key and the value is null. See Section 10.1 on page 39 for a complete list of settings. Thetis will acknowledge a read setting command by sending a write setting command to the host.

Example: {"serialNumber":null}\n

7.1.4 Write setting command

The write setting command is sent to Thetis to write a setting value, or sent from Thetis to the host in response to a read setting command. The key is the setting key and the value is the setting value. See Section 10.1 on page 39 for a complete list of settings. Thetis will acknowledge a write setting command by sending a setting write command back to the host, indicating the new settings value. Thetis will not apply new settings until two seconds after the most recent write setting command or default command was received.

Example: {"deviceName":"Thetis-000"}\n

7.1.5 Default command

The default command is sent to Thetis to set all settings to default values. The key is “default” and the value is null. Thetis will not apply new settings until two seconds after the most recent write setting command or default command was received.

Example: {"default":null}\n

7.1.6 Apply command

The apply command is sent to Thetis to apply all settings. The key is “apply” and the value is null. This command can be sent after a write setting or default command to apply settings immediately instead of after a two second delay.

Example: {"apply":null}\n

7.1.7 Save command

The save command is sent to Thetis to save all settings to Electrically Erasable Programmable Read-Only Memory (EEPROM). The key is “save” and the value is null. The command acknowledgement will not be sent until the save is complete. This may take up to 300 milliseconds. The save command is unnecessary in most applications because Thetis will automatically save all settings on shutdown.

Example: {"save":null}\n

7.1.8 Read time command

The read time command is sent to Thetis to read the date and time of the Real-Time Clock (RTC). The key is “time” and the value is null. Thetis will acknowledge a read time command by sending a write time command to the host.

Example: {"time":null}\n

7.1.9 Write time command

The write time command is sent to Thetis to write the date and time of the RTC, or sent from Thetis to the host in response to a read time command. The key is “time” and the value is a string expressing the date and time in the format “YYYY-MM-DD hh:mm:ss” where each delimiter can be any non-numerical character. Thetis will acknowledge a write time command by sending a write time command back to the host, indicating the new date and time.

Example: {"time":"2020-01-01 00:00:00"}\n

7.1.10 Ping command

The ping command is sent to Thetis to trigger a ping response. The key is “ping” and the value is null. Thetis will acknowledge a ping command by sending a ping response to the host.

Example: {"ping":null}\n

7.1.11 Ping response

The ping response is sent from Thetis to the host in response to the ping command. The key is “ping” and the value is a JSON object containing three key/value pairs indicating the communication interface, device name, and device serial number. The keys are “interface”, “deviceName”, and “serialNumber”, respectively and all values are string types.

Example*: {
 "ping": {
 "interface": "USB",
 "deviceName": "x-IMU3",
 "serialNumber": "0123-4567-89AB-CDEF"
 }
}\n

* The actual JSON will not include any whitespace.

7.1.12 Reset command

The reset command is sent to Thetis to reset Thetis. The key is “reset” and the value is null. A reset is equivalent to switching Thetis off and then on again. Thetis will reset two seconds after receiving this command.

Example: {"reset":null}\n

7.1.13 Shutdown command

The shutdown command is sent to Thetis to switch Thetis off. The key is “shutdown” and the value is null. Thetis will shutdown two seconds after receiving this command.

Example: {"shutdown":null}\n

7.1.14 Strobe command

The strobe command is sent to Thetis to strobe the LED bright white for 5 seconds. The key is “strobe” and the value is null. This command can be used to quickly locate a specific x-IMU3 when using multiple x-IMU3s.

Example: {"strobe":null}\n

7.1.15 Colour command

The colour command is sent to Thetis to set the LED colour. The key is “colour” or “color” and the value is either a RGB hex triplet expressed as a string, or null. Setting the colour will override the normal LED behaviour. A value of null will restore the normal behaviour.

Example: {"colour":"FFFFFF"}\n

7.1.16 Initialise AHRS command

The initialise AHRS command is sent to Thetis to reinitialise the AHRS algorithm. The key is “initialise” or “initialize” and the value is null. This command is unnecessary for normal operation and typically only used during calibration and testing.

Example: {"initialise":null}\n

7.1.17 Heading command

The heading command is sent to Thetis to set the heading of the orientation measurement provided by the AHRS algorithm. The key is “heading” and the value is a number equal to the heading in degrees. The heading command can only be used if the magnetometer is ignored in the AHRS settings.

Example: {"heading":0}\n

7.1.18 Serial accessory command

The serial accessory command is sent to Thetis to transmit data to a serial accessory when the serial interface is in serial accessory mode. The key is “accessory” and the value is the data expressed as a string of up to 256 characters. Longer strings will be truncated to the maximum size. The string escape sequence “\u” can be used to express any byte value as per the JSON specification.

Example: {"accessory":"hello \u0077\u006F\u0072\u006C\u0064"}\n

7.1.19 Note command

The note command is sent to Thetis to generate a timestamped notification message containing a user-defined string. The key is “note” and the value is the string of up to 127 characters. Longer strings

will be truncated to the maximum size. This command can be used to create timestamped notes of events during data logging.

Example: {"note":"Something happened."}\n

7.1.20 Format command

The format command is sent to Thetis to format the microSD. The key is “format” and the value is null. The command acknowledgement will not be sent until the format is complete. This will take approximately 3 seconds for an 8 GB microSD. Larger SD microSDs will take longer to format. Formatting will erase all data on the SD card.

Example: {"format":null}\n

7.1.21 Self-test command

The self-test command is sent to Thetis to perform a self-test. The key is “test” and the value is null. Thetis will acknowledge a self-test command by sending a self-test response to the host once the self-test is complete. This may take up to 5 seconds. Thetis must be stationary during the self-test.

Example: {"test":null}\n

7.1.22 Self-test response

The self-test response is sent from Thetis to the host in response to the self-test command. The key is “test” and the value is a JSON object containing multiple key/value pairs. Each key/value pair indicates the result of a test. Each key is the test name and the value is the string “Passed” if the test was passed.

Example*: {
 "test": {
 "EEPROM": "Passed",
 "RTC": "Passed",
 "Inertial": "Passed",
 "Magnetometer": "Passed",
 "High-g Accelerometer": "Passed",
 "Battery": "Passed",
 "SD Card": "Passed",
 "Wireless": "Passed",
 }
}\n

* The actual JSON will not include any whitespace.

7.1.23 Bootloader command

The bootloader command is sent to Thetis to put Thetis in bootloader mode. The key is “bootloader” and the value is null. Thetis will enter bootloader mode two seconds after receiving this command.

Example: {"bootloader":null}\n

7.1.24 Factory command

The factory command is sent to Thetis to enable factory mode. The key is “factory” and the value is null. In factory mode, read-only settings can be written using the write setting command and the erase command will be enabled.

Example: {"factory":null}\n



Warning - Incorrect use of this command may permanently damage the device. Do not use this command unless instructed by customer support.

7.1.25 Erase command

The erase command is sent to Thetis to erase the EEPROM. The key is “erase” and the value is null. The command acknowledgement will not be sent until the erase is complete. This will take approximately 700 milliseconds. This command can only be used if factory mode is enabled.

Example: {"erase":null}\n



Warning - Incorrect use of this command may permanently damage the device. Do not use this command unless instructed by customer support.

7.2 Data messages

Data messages are sent from Thetis to the host to provide timestamped measurements, serial accessory data, notifications, and error messages. Data messages will be either American Standard Code for Information Interchange (ASCII) or binary, depending on the device settings.

ASCII data messages consist of multiple comma-separated values terminated by the control character character LF. The first value is a single uppercase character indicating the message type. The second value is the timestamp in microseconds. The remaining values are arguments specific to the message type.

Binary data messages are a sequence of bytes terminated by the control character LF. The first byte of the sequence indicates the message type. The value of this byte is equal to 0x80 plus the first character of the equivalent ASCII message. The next eight bytes are the timestamp in microseconds expressed as a 64-bit unsigned integer. The remaining bytes are arguments specific to the message type. Numerical types use little-endian ordering. Byte stuffing is used to remove all occurrences of the control character LF prior to the termination byte.

7.2.1 Byte stuffing

Byte stuffing ensures that the termination byte value, 0x0A, only occurs at the end of a binary data message. This is achieved by replacing all occurrences of the termination byte prior to termination with an escape sequence. This process is identical to Serial Line Internet Protocol (SLIP) except that the “END” byte value is defined as 0x0A. Table 15 lists the values used by the byte stuffing process.

Hex	Decimal	Name	Description
0x0A	10	END	Message termination
0xDB	219	ESC	Message escape
0xDC	220	ESC-END	Transposed message termination
0xDD	221	ESC-ESC	Transposed message escape

Table 15: Values used by the byte stuffing process

Byte stuffing is achieved by the following:

- Replace each occurrence of END in the original message with the two byte sequence: ESC, ESC-END.
- Replace each occurrence of ESC in the original message with the two byte sequence: ESC, ESC-ESC.

The byte stuffing process will not modify the END that terminates the message. Table 16 demonstrates byte stuffing for example byte sequences terminated as binary data messages.

Example	Before byte stuffing	After byte stuffing
1	45 58 41 4D 50 4C 45 OA	45 58 41 4D 50 4C 45 OA
2	45 OA 41 4D 50 4C 45 OA	45 DB DC 41 4D 50 4C 45 OA
3	45 58 DB 4D 50 4C 45 OA	45 58 DB DD 4D 50 4C 45 OA
4	45 58 41 4D 50 DB OA OA	45 58 41 4D 50 DB DD DB DC OA

Table 16: Byte stuffing examples

7.2.2 Inertial message

The inertial message provides timestamped gyroscope and accelerometer measurements. Inertial messages are sent continuously at the message rate configured in the device settings. The first value of an ASCII message is the character “I” and the arguments are six numerical values expressed to four decimal places. The first byte of a binary message is 0xC9 (equal to 0x80 + “I”) and the arguments are six contiguous 32-bit floats. The message arguments are described in Table 17.

Argument	Description
1	Gyroscope X axis in degrees per second
2	Gyroscope Y axis in degrees per second
3	Gyroscope Z axis in degrees per second
4	Accelerometer X axis in g
5	Accelerometer Y axis in g
6	Accelerometer Z axis in g

Table 17: Inertial message arguments

The following message examples are for a timestamp of 1 second (1,000,000 microseconds) and argument values of:

1. Gyroscope X axis = 0
2. Gyroscope Y axis = 0
3. Gyroscope Z axis = 0
4. Accelerometer X axis = 0
5. Accelerometer Y axis = 0
6. Accelerometer Z axis = 1

ASCII example: I,1000000,0.0000,0.0000,0.0000,0.0000,0.0000,1.0000\n

Binary example: C9 40 42 0F 00 0A

7.2.3 Magnetometer message

The magnetometer message provides timestamped magnetometer measurements. Magnetometer messages are sent continuously at the message rate configured in the device settings. The first value of an ASCII message is the character “M” and the arguments are three numerical values expressed to four decimal places. The first byte of a binary message is 0xCD (equal to 0x80 + “M”) and the arguments are three contiguous 32-bit floats. The message arguments are described in Table 18.

Argument	Description
1	Magnetometer X axis in a.u.
2	Magnetometer Y axis in a.u.
3	Magnetometer Z axis in a.u.

Table 18: Magnetometer message arguments

The following message examples are for a timestamp of 1 second (1,000,000 microseconds) and argument values of:

1. Magnetometer X axis = 1
2. Magnetometer Y axis = 0
3. Magnetometer Z axis = 0

ASCII example: M,1000000,1.0000,0.0000,0.0000\n

Binary example: CD 40 42 0F 00 0A

7.2.4 Position Message

The position message provides timestamped GPS data from the on-board radio. The message are sent once per second and will only be valid or populated when the GPS radio has a valid fix. The first value of an ASCII message is the character “P” and the arguments are numerical values. The first byte of a binary message is 0xD0 (equal to 0x80 + “P”) and the arguments are several 32-bit longs and a couple 8-bit characters. The message argument are described in Table 19

Argument	Description
1	GPS fix valid
2	Number of satellites
3	HDOP
4	Latitude in microdegrees
5	Longitude in microdegrees
6	Speed in meters per second
7	Course in millidegrees

Table 19: Position message arguments

The following message examples are for a timestamp of 1 second (1,000,000 microseconds) and argument values of:

1. GPS fix valid = 1
2. Number of satellites = 12

3. HDOP = 7
4. Latitude = 280652369
5. Longitude = -806229767
6. Speed = 0
7. Course = 0

ASCII example: P,1000000,1,12,7,280652369,-806229767,0,0\n

Binary example: 50 40 42 0F 00 00 00 00 00 01 0C 07 10 BA 6A 51 30 0E 17 07 00 00 00 00 0A

7.2.5 Quaternion message

The quaternion message provides timestamped measurements of the orientation of Thetis relative to the Earth. Quaternion messages are sent continuously at the message rate configured in the device settings. The first value of an ASCII message is the character “Q” and the arguments are four numerical values expressed to four decimal places. The first byte of a binary message is 0xD1 (equal to 0x80 + “Q”) and the arguments are four contiguous 32-bit floats. The message arguments are described in Table 20.

Argument	Description
1	Quaternion W element
2	Quaternion X element
3	Quaternion Y element
4	Quaternion Z element

Table 20: Quaternion message arguments

The following message examples are for a timestamp of 1 second (1,000,000 microseconds) and argument values of:

1. Quaternion W element = 1
2. Quaternion X element = 0
3. Quaternion Y element = 0
4. Quaternion Z element = 0

ASCII example: Q,1000000,1.0000,0.0000,0.0000,0.0000\n

Binary example: D1 40 42 0F 00 00 00 00 00 00 00 00 80 3F 00 00 00 00 00 00 00 00 00 00 00 00 0A

7.2.6 Rotation matrix message

The rotation matrix message provides timestamped measurements of the orientation of Thetis relative to the Earth. Rotation matrix messages are sent continuously at the message rate configured in the device settings. The first value of an ASCII message is the character “R” and the arguments are nine numerical values expressed to four decimal places. The first byte of a binary message is 0xD2 (equal to 0x80 + “R”) and the arguments are nine contiguous 32-bit floats. The message arguments are described in Table 21 on the next page.

Argument	Description
1	Rotation matrix XX element
2	Rotation matrix XY element
3	Rotation matrix XZ element
4	Rotation matrix YX element
5	Rotation matrix YY element
6	Rotation matrix YZ element
7	Rotation matrix ZX element
8	Rotation matrix ZY element
9	Rotation matrix ZZ element

Table 21: Rotation matrix message arguments

The following message examples are for a timestamp of 1 second (1,000,000 microseconds) and argument values of:

1. Rotation matrix XX element = 1
 2. Rotation matrix XY element = 0
 3. Rotation matrix XZ element = 0
 4. Rotation matrix YX element = 0
 5. Rotation matrix YY element = 1
 6. Rotation matrix YZ element = 0
 7. Rotation matrix ZX element = 0
 8. Rotation matrix ZY element = 0
 9. Rotation matrix ZZ element = 1

ASCII example: R,1000000,1.0000,0.0000,0.0000,0.0000,1.0000,0.0000,0.0000,0.0000,0.0000,1.0000\n

7.2.7 Euler angles message

The Euler angles message provides timestamped measurements of the orientation of Thetis relative to the Earth. Euler angles messages are sent continuously at the message rate configured in the device settings. The first value of an ASCII message is the character “A” and the arguments are three numerical values expressed to four decimal places. The first byte of a binary message is 0xC1 (equal to 0x80 + “A”) and the arguments are three contiguous 32-bit floats. The message arguments are described in Table 22.

Argument	Description
1	Roll angle in degrees
2	Pitch angle in degrees
3	Yaw angle in degrees

Table 22: Euler angles message arguments

The following message examples are for a timestamp of 1 second (1,000,000 microseconds) and argument values of:

1. Roll angle = 0
 2. Pitch angle = 0
 3. Yaw angle = 0

ASCII example: A,1000000,0.0000,0.0000,0.0000\n

7.2.8 Linear acceleration message

The linear acceleration message provides timestamped measurements of linear acceleration and the orientation of Thetis relative to the Earth. Linear acceleration messages are sent continuously at the message rate configured in the device settings. The first value of an ASCII message is the character “L” and the arguments are seven numerical values expressed to four decimal places. The first byte of a binary message is 0xCC (equal to 0x80 + “L”) and the arguments are seven contiguous 32-bit floats. The message arguments are described in Table 23.

Argument	Description
1	Quaternion W element
2	Quaternion X element
3	Quaternion Y element
4	Quaternion Z element
5	Linear acceleration X axis in g
6	Linear acceleration Y axis in g
7	Linear acceleration Z axis in g

Table 23: Linear acceleration message arguments

The following message examples are for a timestamp of 1 second (1,000,000 microseconds) and argument values of:

1. Quaternion W element = 1
 2. Quaternion X element = 0
 3. Quaternion Y element = 0
 4. Quaternion Z element = 0
 5. Linear acceleration X axis = 0
 6. Linear acceleration Y axis = 0
 7. Linear acceleration Z axis = 0

ASCII example: L,1000000,1.0000,0.0000,0.0000,0.0000,0.0000,0.0000,0.0000,0.0000\n

7.2.9 Earth acceleration message

The Earth acceleration message provides timestamped measurements of Earth acceleration and the orientation of Thetis relative to the Earth. Earth acceleration messages are sent continuously at the message rate configured in the device settings. The first value of an ASCII message is the character "E" and the arguments are seven numerical values expressed to four decimal places. The first byte of a binary message is 0xC5 (equal to 0x80 + "E") and the arguments are seven contiguous 32-bit floats. The message arguments are described in Table 24 on the following page.

Argument	Description
1	Quaternion W element
2	Quaternion X element
3	Quaternion Y element
4	Quaternion Z element
5	Earth acceleration X axis in g
6	Earth acceleration Y axis in g
7	Earth acceleration Z axis in g

Table 24: Earth acceleration message arguments

The following message examples are for a timestamp of 1 second (1,000,000 microseconds) and argument values of:

1. Quaternion W element = **1**
2. Quaternion X element = **0**
3. Quaternion Y element = **0**
4. Quaternion Z element = **0**
5. Earth acceleration X axis = **0**
6. Earth acceleration Y axis = **0**
7. Earth acceleration Z axis = **0**

ASCII example: E,1000000,**1.0000,0.0000,0.0000,0.0000,0.0000,0.0000,0.0000\n**

Binary example: C5 40 42 0F 00 0A

7.2.10 AHRS status message

The AHRS status message provides timestamped indications of the AHRS status flags. An AHRS status message is sent each time a status flag changes. AHRS status messages are only sent if AHRS messages are enabled in the device settings. The first value of an ASCII message is the character "U" and the arguments are four numerical values expressed to four decimal places. The first byte of a binary message is 0xD5 (equal to 0x80 + "U") and the arguments are four contiguous 32-bit floats. The message arguments are described in Table 25.

Argument	Description
1	Initialising
2	Angular rate recovery
3	Acceleration recovery
4	Magnetic recovery

Table 25: AHRS status message arguments (all arguments are Boolean, see Table 26 on the next page)

Value	Boolean
0	False
!0	True

Table 26: Boolean argument values

The following message examples are for a timestamp of 1 second (1,000,000 microseconds) and argument values of:

1. Initialising = 1
2. Angular rate recovery = 0
3. Acceleration recovery = 0
4. Magnetic recovery = 0

ASCII example: U,1000000,1.0000,0.0000,0.0000,0.0000\n

Binary example: D5 40 42 0F 00 0A

7.2.11 High-g accelerometer message

The High-g accelerometer message provides timestamped high-g accelerometer measurements. High-g accelerometer messages are sent continuously at the message rate configured in the device settings. The first value of an ASCII message is the character "H" and the arguments are three numerical values expressed to four decimal places. The first byte of a binary message is 0xC8 (equal to 0x80 + "H") and the arguments are three contiguous 32-bit floats. The message arguments are described in Table 27.

Argument	Description
1	High-g accelerometer X axis in g
2	High-g accelerometer Y axis in g
3	High-g accelerometer Z axis in g

Table 27: High-g accelerometer message arguments

The following message examples are for a timestamp of 1 second (1,000,000 microseconds) and argument values of:

1. High-g accelerometer X axis = 0
2. High-g accelerometer Y axis = 0
3. High-g accelerometer Z axis = 1

ASCII example: H,1000000,0.0000,0.0000,1.0000\n

Binary example: C8 40 42 0F 00 0A

7.2.12 Temperature message

The temperature message provides timestamped temperature measurements. Temperature messages are sent continuously at the message rate configured in the device settings. The first value of an ASCII message is the character "T" and the argument is a numerical value expressed to four decimal places. The first byte of a binary message is 0xD4 (equal to 0x80 + "T") and the argument is a 32-bit float. The message arguments are described in Table 28 on the next page.

Argument	Description
1	Temperature in degrees Celsius

Table 28: Temperature message arguments

The following message examples are for a timestamp of 1 second (1,000,000 microseconds) and argument values of:

1. Temperature = 25

ASCII example: T,1000000,**25.0000**\n

Binary example: D4 40 42 0F 00 00 00 00 00 00 00 00 00 00 41 C8 0A

7.2.13 Battery message

The battery message message provides timestamped measurements of the battery level and charger status. Battery message messages are sent continuously at the message rate configured in the device settings. The first value of an ASCII message is the character "B" and the arguments are four numerical values expressed to four decimal places. The first byte of a binary message is 0xC2 (equal to 0x80 + "B") and the arguments are four contiguous 32-bit floats. The message arguments are described in Table 29.

Argument	Description
1	Battery percentage
2	Battery voltage in volts
3	Charging status (See Table 30)

Table 29: Battery message arguments

Charging status	Description
0	Not connected
1	Charging
2	Charging complete

Table 30: Charging status enumeration

The following message examples are for a timestamp of 1 second (1,000,000 microseconds) and argument values of:

1. Percentage = 100
2. Voltage = 4.2
3. Charging status = 2

ASCII example: B,1000000,**100.0000**,**4.2000**,**2.0000**\n

Binary example: C2 40 42 0F 00 00 00 00 00 00 00 00 00 00 00 00 42 66 66 86 40 00 00 00 40 0A

7.2.14 RSSI message

The Received Signal Strength Indicator (RSSI) message provides timestamped Wi-Fi RSSI measurements. RSSI messages are sent continuously at the message rate configured in the device settings. RSSI messages will only be sent if Thetis is connected in Wi-Fi client mode. The first value of an ASCII message is the character "W" and the arguments are two numerical values expressed to four decimal places. The first byte of a binary message is 0xD7 (equal to 0x80 + "W") and the arguments are two contiguous 32-bit floats. The message arguments are described in Table 31.

Argument	Description
1	RSSI percentage
2	RSSI power in dBm

Table 31: RSSI message arguments

The following message examples are for a timestamp of 1 second (1,000,000 microseconds) and argument values of:

1. RSSI percentage = 100
2. RSSI power = -50

ASCII example: W,1000000,100.000,-50.000\n

Binary example: D7 40 42 0F 00 00 00 00 00 00 00 C8 42 00 00 48 C2 0A

7.2.15 Serial accessory message

The serial accessory message provides timestamped received serial accessory data. Serial accessory messages are sent as serial accessory data is received as configured in the device settings. The first value of an ASCII message is the character "S" and the argument is the received data. Received byte values less than 0x20 or greater than 0x7E will be replaced with the character "?" so that the argument is a string of printable characters. The string is not null-terminated. The first byte of a binary message is 0xD3 (equal to 0x80 + "S") and the argument is the unmodified received data. The message arguments are described in Table 32.

Argument	Description
1	Received serial accessory data

Table 32: Serial accessory message arguments

The following message examples are for a timestamp of 1 second (1,000,000 microseconds) and argument values of:

1. Data = 0x61 0x62 0x63 0x31 0x32 0x33 0xF1 0xF2 0xF3

ASCII example: S,1000000,abc123???\n

Binary example: D3 40 42 0F 00 00 00 00 00 61 62 63 31 32 33 F1 F2 F3 0A

7.2.16 Notification message

The notification message provides timestamped notifications of system events. Notification messages may be sent by Thetis at any time and cannot be disabled. The first value of an ASCII message is the character "N". The first byte of a binary message is 0xCE (equal to 0x80 + "N"). The argument of both ASCII and binary

messages is a string of printable characters. The string is not null-terminated. The message arguments are described in Table 33.

Argument	Description
1	Notification string

Table 33: Notification message arguments

The following message examples are for a timestamp of 1 second (1,000,000 microseconds) and argument values of:

1. String = [This is a notification message.](#)

ASCII example: N,1000000,[This is a notification message.\n](#)

Binary example: CE 40 42 0F 00 00 00 00 00 54 68 69 73 20 69 73 20 61 20 6E 6F 74 69 66 69 63 61 74 69 6F 6E 20 6D 65 73 73 61 67 65 2E 0A

7.2.17 Error message

The error message provides timestamped notifications of errors. Error messages may be sent by Thetis at any time and cannot be disabled. The first value of an ASCII message is the character “F”. The first byte of a binary message is 0xC6 (equal to 0x80 + “F”). The argument of both ASCII and binary messages is a string of printable characters. The string is not null-terminated. The message arguments are described in Table 34.

Argument	Description
1	Error string

Table 34: Notification message arguments

The following message examples are for a timestamp of 1 second (1,000,000 microseconds) and argument values of:

1. String = [This is an error message.](#)

ASCII example: F,1000000,[This is an error message.\n](#)

Binary example: C6 40 42 0F 00 00 00 00 00 54 68 69 73 20 69 73 20 61 6E 20 65 72 72 6F 72 20 6D 65 73 73 61 67 65 2E 0A

8 Sample rates, message rates, and timestamps

This section describes the relationship between sample rates and message rates, and the role of timestamps in synchronisation.

8.1 Sample rates

The sample rate is the rate at which measurements are sampled by a measurement source. For example, an Analog-to-Digital Converter (ADC). All sample rates are fixed and cannot be adjusted by the user. Each measurement source has an independent sample clock. The sample rate and associated data messages for each measurement source are listed in Table 35 on the following page.

Measurement source	Sample rate	Sample rate error	Data messages
Inertial sensor	400 Hz	±0.3%	Inertial, Quaternion, Rotation matrix Euler angles, Linear acceleration, Earth acceleration, Temperature
Magnetometer	20 Hz	±8%	Magnetometer
High-g accelerometer	1600 Hz	±5%	High-g accelerometer
Battery voltage	5 Hz	-	Battery
RSSI	1 Hz	-	RSSI

Table 35: Sample rate and associated data messages for each measurement source

8.2 Message rates

The message rate is the rate at which a data message is sent. The message rate of each data message type is configured by a separate message rate divisor in the device settings. A message rate divisor is a positive integer that a fixed sample rate is divided by to obtain the message rate. For example, if the inertial message rate divisor is 4 then the inertial message rate will be 100 messages per second. A message rate divisor of 0 will disable the sending of that data message type. See Section 10.1.78 on page 52 to Section 10.1.84 on page 53 for a detailed description and examples for each message rate divisor setting.

8.3 Sample averaging

If a message rate divisor is greater than 1 then the measurements in each data message will be the average of the most recent n samples where n equal to the message rate divisor. The timestamp of the data message will be that of the most recent sample. For example, if the inertial message rate divisor is 8 then the measurements in each inertial message will be the average of 8 samples and the timestamp of the message will be that of the 8th sample.

8.4 Timestamps

The timestamp of a data message indicates the time at which a measurement was obtained. For example, when an ADC conversion completes. Timestamps are therefore not affected by the sample rate error or the latency of a communication interface. Applications that involve time-dependent calculations such as numerical integration or interpolation should not infer timing from the nominal sample rate and should instead use the timestamp of each measurement. A timestamp is the number of microseconds since Thetis was switched on, with a resolution of one microsecond.

8.5 Synchronisation

Multiple x-IMU3s operating on the same Wi-Fi network will automatically synchronise so that the timestamps from all x-IMU3s is the number of microseconds since the first x-IMU3 was switched on. The sample clocks of synchronised x-IMU3s will remain asynchronous. If an application requires synchronous sampling then this must be achieved in post-processing through interpolation and resampling.

9 Network announcement message

The network announcement message is used by a host to discover and connect to x-IMU3s on the same network. The message is continuously broadcast by the x-IMU3 on UDP port 10000 at a fixed rate of one message per second. Each message provides the device name, serial number, Wi-Fi and battery status, as well as the device settings required for a host to establish a Transmission Control Protocol (TCP) or UDP connection. The message is a single JSON object. The key/value pairs are described in Table 36 on the next page.

Key	Value type	Description
“sync”	number	Used for synchronisation
“name”	string	Device name
“sn”	string	Device serial number
“ip”	string	x-IMU3 IP address
“port”	number	TCP port
“send”	number	UDP send port (x-IMU3 sends to this port)
“receive”	number	UDP receive port (x-IMU3 receives on this port)
“rssI”	number	RSSI percentage (-1 in Wi-Fi AP mode)
“battery”	number	Battery percentage
“status”	number	Charging status (See Table 30 on page 35)

Table 36: Network announcement message key/value pairs

Example*:

```
{  
    "sync": 0,  
    "name": "x-IMU3",  
    "sn": "0123-4567-89AB-CDEF",  
    "ip": "192.168.1.1",  
    "port": 7000,  
    "send": 8000,  
    "receive": 9000,  
    "rssI": 100,  
    "battery": 100,  
    "status": 2  
}
```

* The actual JSON will not include any whitespace.

10 Device Settings

10.1 Individual Settings

10.1.1 Calibration date (read-only)

Description: Calibration date.
JSON key: "calibrationDate"
JSON value type: string
Default value: “Unknown”

10.1.2 System clock calibration (read-only)

Description: System clock calibration.
JSON key: "systemClockCalibration"
JSON value type: number
Default value: 1.0

10.1.3 RTC calibration (read-only)

Description: RTC calibration.

JSON key: "rtcCalibration"

JSON value type: number

Default value: 1.0

10.1.4 Battery voltmeter sensitivity (read-only)

Description: Battery voltmeter sensitivity.

JSON key: "batteryVoltmeterSensitivity"

JSON value type: number

Default value: 1.0

10.1.5 Gyroscope misalignment (read-only)

Description: Gyroscope misalignment.

JSON key: "gyroscopeMisalignment"

JSON value type: array of 9 numbers

Default value: [1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0]

10.1.6 Gyroscope sensitivity (read-only)

Description: Gyroscope sensitivity.

JSON key: "gyroscopeSensitivity"

JSON value type: array of 3 numbers

Default value: [1.0, 1.0, 1.0]

10.1.7 Gyroscope offset (read-only)

Description: Gyroscope offset.

JSON key: "gyroscopeOffset"

JSON value type: array of 3 numbers

Default value: [0.0, 0.0, 0.0]

10.1.8 Accelerometer misalignment (read-only)

Description: Accelerometer misalignment.

JSON key: "accelerometerMisalignment"

JSON value type: array of 9 numbers

Default value: [1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0]

10.1.9 Accelerometer sensitivity (read-only)

Description: Accelerometer sensitivity.
JSON key: "accelerometerSensitivity"
JSON value type: array of 3 numbers
Default value: [1.0, 1.0, 1.0]

10.1.10 Accelerometer offset (read-only)

Description: Accelerometer offset.
JSON key: "accelerometerOffset"
JSON value type: array of 3 numbers
Default value: [0.0, 0.0, 0.0]

10.1.11 Soft iron matrix (read-only)

Description: Soft iron matrix.
JSON key: "softIronMatrix"
JSON value type: array of 9 numbers
Default value: [1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0]

10.1.12 Hard iron offset (read-only)

Description: Hard iron offset.
JSON key: "hardIronOffset"
JSON value type: array of 3 numbers
Default value: [0.0, 0.0, 0.0]

10.1.13 High-g accelerometer misalignment (read-only)

Description: High-g accelerometer misalignment.
JSON key: "highGAccelerometerMisalignment"
JSON value type: array of 9 numbers
Default value: [1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0]

10.1.14 High-g accelerometer sensitivity (read-only)

Description: High-g accelerometer sensitivity.
JSON key: "highGAccelerometerSensitivity"
JSON value type: array of 3 numbers
Default value: [1.0, 1.0, 1.0]

10.1.15 High-g accelerometer offset (read-only)

Description: High-g accelerometer offset.
JSON key: "highGAccelerometerOffset"
JSON value type: array of 3 numbers
Default value: [0.0, 0.0, 0.0]

10.1.16 Device name

Description: Device name.
JSON key: "deviceName"
JSON value type: string
Default value: "x-IMU3"

10.1.17 Serial number (read-only)

Description: Serial number.
JSON key: "serialNumber"
JSON value type: string
Default value: "Unknown"

10.1.18 Firmware version (read-only)

Description: Firmware version.
JSON key: "firmwareVersion"
JSON value type: string
Default value: "Unknown"

10.1.19 Bootloader version (read-only)

Description: Bootloader version.
JSON key: "bootloaderVersion"
JSON value type: string
Default value: "Unknown"

10.1.20 Hardware version (read-only)

Description: Hardware version.
JSON key: "hardwareVersion"
JSON value type: string
Default value: "Unknown"

10.1.21 Serial mode

Description: Serial mode.
JSON key: "serialMode"
JSON value type: number
Default value: 0

10.1.22 Serial baud rate

Description: Serial baud rate.
JSON key: "serialBaudRate"
JSON value type: number
Default value: 115200

10.1.23 Serial RTS/CTS enabled

Description: Serial Request To Send (RTS)/Clear To Send (CTS) enabled.
JSON key: "serialRtsCtsEnabled"
JSON value type: true or false
Default value: false

10.1.24 Serial accessory number of bytes

Description: Serial accessory number of bytes.
JSON key: "serialAccessoryNumberOfBytes"
JSON value type: number
Default value: 1024

10.1.25 Serial accessory termination byte

Description: Serial accessory termination byte.
JSON key: "serialAccessoryTerminationByte"
JSON value type: number
Default value: 10

10.1.26 Serial accessory timeout

Description: Serial accessory timeout.
JSON key: "serialAccessoryTimeout"
JSON value type: number
Default value: 100

10.1.27 Wireless mode

Description: Wireless mode.

JSON key: "wirelessMode"

JSON value type: number

Default value: 2

10.1.28 Wireless firmware version (read-only)

Description: Wireless firmware version.

JSON key: "wirelessFirmwareVersion"

JSON value type: string

Default value: "Unknown"

10.1.29 External antennae enabled

Description: External antennae enabled.

JSON key: "externalAntennaeEnabled"

JSON value type: true or false

Default value: false

10.1.30 Wi-Fi region

Description: Wi-Fi region.

JSON key: "wiFiRegion"

JSON value type: number

Default value: 2

10.1.31 Wi-Fi MAC address (read-only)

Description: Wi-Fi Media Access Control (MAC) address.

JSON key: "wiFiMacAddress"

JSON value type: string

Default value: 0

10.1.32 Wi-Fi IP address (read-only)

Description: Wi-Fi Internet Protocol (IP) address.

JSON key: "wiFiIPAddress"

JSON value type: string

Default value: 0

10.1.33 Wi-Fi client SSID

Description: Wi-Fi client Service Set Identifier (SSID).
JSON key: "wifiClientSsid"
JSON value type: string
Default value: ""

10.1.34 Wi-Fi client key

Description: Wi-Fi client key.
JSON key: "wifiClientKey"
JSON value type: string
Default value: ""

10.1.35 Wi-Fi client channel

Description: Wi-Fi client channel.
JSON key: "wifiClientChannel"
JSON value type: number
Default value: 0

10.1.36 Wi-Fi client DHCP enabled

Description: Wi-Fi client Dynamic Host Configuration Protocol (DHCP) enabled.
JSON key: "wifiClientDhcpEnabled"
JSON value type: true or false
Default value: true

10.1.37 Wi-Fi client IP address

Description: Wi-Fi client IP address.
JSON key: "wifiClientIpAddress"
JSON value type: string
Default value: "192.168.1.2"

10.1.38 Wi-Fi client netmask

Description: Wi-Fi client netmask.
JSON key: "wifiClientNetmask"
JSON value type: string
Default value: "255.255.255.0"

10.1.39 Wi-Fi client gateway

Description: Wi-Fi client gateway.
JSON key: "wifiClientGateway"
JSON value type: string
Default value: "192.168.1.1"

10.1.40 Wi-Fi AP SSID

Description: Wi-Fi Access Point (AP) SSID.
JSON key: "wifiAPSsid"
JSON value type: string
Default value: ""

10.1.41 Wi-Fi AP key

Description: Wi-Fi AP key.
JSON key: "wifiAPKey"
JSON value type: string
Default value: ""

10.1.42 Wi-Fi AP channel

Description: Wi-Fi AP channel.
JSON key: "wifiAPChannel"
JSON value type: number
Default value: 36

10.1.43 Wi-Fi AP IP address

Description: Wi-Fi AP IP address.
JSON key: "wifiAPIPAddress"
JSON value type: string
Default value: 0x0101A9C0

10.1.44 TCP port

Description: TCP port.
JSON key: "tcpPort"
JSON value type: number
Default value: 7000

10.1.45 UDP IP address

Description: UDP IP address.
JSON key: "udpIPAddress"
JSON value type: string
Default value: 0

10.1.46 UDP send port

Description: UDP send port.
JSON key: "udpSendPort"
JSON value type: number
Default value: 0

10.1.47 UDP receive port

Description: UDP receive port.
JSON key: "udpReceivePort"
JSON value type: number
Default value: 9000

10.1.48 UDP low latency

Description: UDP low latency.
JSON key: "udpLowLatency"
JSON value type: true or false
Default value: false

10.1.49 Synchronisation enabled

Description: Synchronisation enabled.
JSON key: "synchronisationEnabled"
JSON value type: true or false
Default value: true

10.1.50 Synchronisation network latency

Description: Synchronisation network latency.
JSON key: "synchronisationNetworkLatency"
JSON value type: number
Default value: 1500

10.1.51 Bluetooth address (read-only)

Description: Bluetooth address.
JSON key: "bluetoothAddress"
JSON value type: number
Default value: 0

10.1.52 Bluetooth name

Description: Bluetooth name.
JSON key: "bluetoothName"
JSON value type: string
Default value: ""

10.1.53 Bluetooth pin code

Description: Bluetooth pin code.
JSON key: "bluetoothPinCode"
JSON value type: string
Default value: ""

10.1.54 Bluetooth discovery mode

Description: Bluetooth discovery mode.
JSON key: "bluetoothDiscoveryMode"
JSON value type: number
Default value: 2

10.1.55 Bluetooth paired address (read-only)

Description: Bluetooth paired address.
JSON key: "bluetoothPairedAddress"
JSON value type: number
Default value: 0

10.1.56 Bluetooth paired link key (read-only)

Description: Bluetooth paired link key.
JSON key: "bluetoothPairedLinkKey"
JSON value type: number
Default value: 0

10.1.57 Data logger enabled

Description: Data logger enabled.
JSON key: "dataLoggerEnabled"
JSON value type: true or false
Default value: false

10.1.58 Data logger file name prefix

Description: Data logger file name prefix.
JSON key: "dataLoggerFileNamePrefix"
JSON value type: string
Default value: ""

10.1.59 Data logger file name time enabled

Description: Data logger file name time enabled.
JSON key: "dataLoggerFileNameTimeEnabled"
JSON value type: true or false
Default value: true

10.1.60 Data logger file name counter enabled

Description: Data logger file name counter enabled.
JSON key: "dataLoggerFileNameCounterEnabled"
JSON value type: true or false
Default value: false

10.1.61 Data logger max file size

Description: Data logger max file size.
JSON key: "dataLoggerMaxFileSize"
JSON value type: number
Default value: 0

10.1.62 Data logger max file period

Description: Data logger max file period.
JSON key: "dataLoggerMaxFilePeriod"
JSON value type: number
Default value: 0

10.1.63 Axes alignment

Description: Axes alignment.

JSON key: "axesAlignment"

JSON value type: number

Default value: 0

10.1.64 Gyroscope offset correction enabled

Description: Gyroscope offset correction enabled.

JSON key: "gyroscopeOffsetCorrectionEnabled"

JSON value type: true or false

Default value: true

10.1.65 AHRS axes convention

Description: AHRS axes convention.

JSON key: "ahrsAxesConvention"

JSON value type: number

Default value: 0

10.1.66 AHRS gain

Description: AHRS gain.

JSON key: "ahrsGain"

JSON value type: number

Default value: 0.5

10.1.67 AHRS ignore magnetometer

Description: AHRS ignore magnetometer.

JSON key: "ahrsIgnoreMagnetometer"

JSON value type: true or false

Default value: false

10.1.68 AHRS acceleration rejection enabled

Description: AHRS acceleration rejection enabled.

JSON key: "ahrsAccelerationRejectionEnabled"

JSON value type: true or false

Default value: true

10.1.69 AHRS magnetic rejection enabled

Description: AHRS magnetic rejection enabled.
JSON key: "ahrsMagneticRejectionEnabled"
JSON value type: true or false
Default value: true

10.1.70 Binary mode enabled

Description: Binary mode enabled.
JSON key: "binaryModeEnabled"
JSON value type: true or false
Default value: true

10.1.71 USB data messages enabled

Description: USB data messages enabled.
JSON key: "usbDataMessagesEnabled"
JSON value type: true or false
Default value: true

10.1.72 Serial data messages enabled

Description: Serial data messages enabled.
JSON key: "serialDataMessagesEnabled"
JSON value type: true or false
Default value: true

10.1.73 TCP data messages enabled

Description: TCP data messages enabled.
JSON key: "tcpDataMessagesEnabled"
JSON value type: true or false
Default value: true

10.1.74 UDP data messages enabled

Description: UDP data messages enabled.
JSON key: "udpDataMessagesEnabled"
JSON value type: true or false
Default value: true

10.1.75 Bluetooth data messages enabled

Description: Bluetooth data messages enabled.
JSON key: "bluetoothDataMessagesEnabled"
JSON value type: true or false
Default value: true

10.1.76 Data logger data messages enabled

Description: Data logger data messages enabled.
JSON key: "dataLoggerDataMessagesEnabled"
JSON value type: true or false
Default value: true

10.1.77 AHRS message type

Description: AHRS message type.
JSON key: "ahrsMessageType"
JSON value type: number
Default value: 0

10.1.78 Inertial message rate divisor

Description: Inertial message rate divisor.
JSON key: "inertialMessageRateDivisor"
JSON value type: number
Default value: 8

10.1.79 Magnetometer message rate divisor

Description: Magnetometer message rate divisor.
JSON key: "magnetometerMessageRateDivisor"
JSON value type: number
Default value: 1

10.1.80 AHRS message rate divisor

Description: AHRS message rate divisor.
JSON key: "ahrsMessageRateDivisor"
JSON value type: number
Default value: 8

10.1.81 High-g accelerometer message rate divisor

Description: High-g accelerometer message rate divisor.
JSON key: "highGAccelerometerMessageRateDivisor"
JSON value type: number
Default value: 32

10.1.82 Temperature message rate divisor

Description: Temperature message rate divisor.
JSON key: "temperatureMessageRateDivisor"
JSON value type: number
Default value: 5

10.1.83 Battery message rate divisor

Description: Battery message rate divisor.
JSON key: "batteryMessageRateDivisor"
JSON value type: number
Default value: 5

10.1.84 RSSI message rate divisor

Description: RSSI message rate divisor.
JSON key: "rssimessageRateDivisor"
JSON value type: number
Default value: 1

10.1.85 FTP! enabled

Description: **FTP!** (**FTP!**) enabled.
JSON key: "ftpEnabled"
JSON value type: true or false
Default value: false

10.1.86 FTP! username

Description: **FTP!** username.
JSON key: "ftpUsername"
JSON value type: string
Default value: ""

10.1.87 Log print level

Description: Log print level.

JSON key: "logPrintLevel"

JSON value type: number

Default value: 0

10.1.88 Log file level

Description: Log file level.

JSON key: "logFileLevel"

JSON value type: number

Default value: 2

10.1.89 GPS RTC sync enabled

Description: GPS RTC sync enabled.

JSON key: "gpsRtcSyncEnabled"

JSON value type: true or false

Default value: false

10.1.90 Accelerometer range

Description: Accelerometer range.

JSON key: "accelerometerRange"

JSON value type: number

Default value: 2

10.1.91 Gyroscope range

Description: Gyroscope range.

JSON key: "gyroscopicRange"

JSON value type: number

Default value: 0

10.1.92 IMU data rate

Description: IMU data rate.

JSON key: "imuDataRate"

JSON value type: number

Default value: 5

10.1.93 Magnetometer performance mode

Description: Magnetometer performance mode.
JSON key: "magnetometerPerformanceMode"
JSON value type: number
Default value: 1

10.1.94 Magnetometer operation mode

Description: Magnetometer operation mode.
JSON key: "magnetometerOperationMode"
JSON value type: number
Default value: 0

10.1.95 Magnetometer data rate

Description: Magnetometer data rate.
JSON key: "magnetometerDataRate"
JSON value type: number
Default value: 1

10.1.96 Magnetometer range

Description: Magnetometer range.
JSON key: "magnetometerRange"
JSON value type: number
Default value: 0

10.1.97 Gauge reset voltage

Description: Gauge reset voltage.
JSON key: "gaugeResetVoltage"
JSON value type: number
Default value: 2.5

10.1.98 Gauge activity threshold

Description: Gauge activity threshold.
JSON key: "gaugeActivityThreshold"
JSON value type: number
Default value: 0.15

10.1.99 Gauge hibernation threshold

Description: Gauge hibernation threshold.
JSON key: "gaugeHibernationThreshold"
JSON value type: number
Default value: 0.1

10.1.100 Gauge alert minimum voltage

Description: Gauge alert minimum voltage.
JSON key: "gaugeAlertMinimumVoltage"
JSON value type: number
Default value: 3.0

10.1.101 Gauge alert maximum voltage

Description: Gauge alert maximum voltage.
JSON key: "gaugeAlertMaximumVoltage"
JSON value type: number
Default value: 4.2

10.1.102 Fusion update rate

Description: Fusion update rate.
JSON key: "fusionUpdateRate"
JSON value type: number
Default value: 20

Glossary

a.u. arbitrary units	14
ADC Analog-to-Digital Converter	37
AHRS Attitude Heading Reference System	14
AP Access Point	46
API Application Programming Interface	
ASCII American Standard Code for Information Interchange	27
CDC Communications Device Class	
CS Chip Select	
CSV Comma-Separated Values	21
CTS Clear To Send	43
DOF Degree of Freedom	7
DHCP Dynamic Host Configuration Protocol	45
EEPROM Electrically Erasable Programmable Read-Only Memory	23
GPIO General Purpose Input Output	
GPS Global Positioning System	21
GUI Graphical User Interface	
HDOP Horizontal Dillution of Precision	
I2C Inter-Integrated Circuit	
IC Integrated Circuit	
IMU Inertial Measurement Unit	7
IP Internet Protocol	44

IP67 Ingress Protection 67	11
JSON JavaScript Object Notation	22
LED Light-Emitting Diode	17
LF Line Feed	22
MAC Media Access Control	44
MISO Master In Slave Out	
MOSI Master Out Slave In	
RGB Red Green Blue	17
RMS Root Mean Square	15
RSSI Received Signal Strength Indicator	36
RTC Real-Time Clock	24
RTS Request To Send	43
RX Receive	
microSD micro Secure Digital	21
SCK Signal Clock (SPI)	
SCL Signal Clock (I2C)	
SDA Signal Data	
SLIP Serial Line Internet Protocol	27
SPI Serial Peripheral Interface	
SSID Service Set Identifier	45
TCP Transmission Control Protocol	38
TX Transmit	

UDP User Datagram Protocol	22
UART Universal Asynchronous Receiver-Transmitter	
USB Universal Serial Bus	21

Document version history

Version	Date	Changes
v1.0	October 31, 2023	<ul style="list-style-type: none">Converted original x-IMU3 user manual (v1.5) to Thetis user manual and released

Disclaimer

The information in this document pertains to information related to x-io Technologies products. This information is provided as a service to our customers, and may be used for information purposes only.

x-io Technologies assumes no liabilities or responsibilities for errors or omissions in this document. This document may be changed at any time at x-io Technologies' sole discretion without any prior notice to anyone. x-io Technologies is not committed to updating this document in the future.

Copyright © 2023 x-io Technologies. All rights reserved.