Strike Sense

Authors:

Timothy Azinord	Landon Wright	Shem Fils-Aime	Benjamin Bolton
Computer Engineering	*Computer Engineering*	*Electrical Engineering*	*Electrical Engineering*

Review Committee:

Dr. Hao Zheng	ECE	*Associate Professor*
Dr. Mark Maddox	ECE	*Associate Professor*
Dr. Enxia Zhang	ECE	*Associate Professor*

Table of Contents

Chapter 1 - Executive Summary

Improving as a martial artist requires more than just knowing how many punches were thrown—it requires understanding *how* those strikes flow together, what patterns emerge, and where adjustments can optimize performance. Existing punch trackers, such as Hykso, focus on raw counts and basic differentiation between straights and power punches. While useful, this narrow scope provides limited coaching value, particularly when hooks, uppercuts, and combinations play central roles in real training and competition. Camera-based solutions offer broader motion analysis but often fail under occlusion and lack the precision necessary for rapid, close-range striking.

The MUGI: Smart Strike Performance Analyzer is designed to fill this gap by delivering actionable insights into striking dynamics, not just surface-level statistics. By leveraging two 9-DoF IMUs with quaternion-based orientation tracking and integrated heart rate monitoring, MUGI provides a richer and more connected picture of performance. In addition to accurately classifying straights, hooks, and uppercuts from both sides, the system will:

- Identify strike sequences and recommend optimized combinations by analyzing the frequency and order of punches within a training session.

- Detect rotational movement and power contribution, offering feedback on form efficiency and whether a strike is being thrown with proper mechanics.

- Integrate physiological feedback by correlating strike sequences and heart rate patterns, helping athletes track fatigue and endurance across combinations.

- Highlight best/worst combos or tendencies, allowing fighters to see which sequences produce the highest efficiency, power, or technical execution.

With low-latency feedback (<150 ms) streamed directly to an iOS app, MUGI goes beyond being a punch counter. It becomes a virtual coach, recognizing not only the strikes themselves but also the *relationships between them*—whether a right cross consistently follows a jab, or whether hooks drop in frequency as fatigue sets in. These deeper insights enable tailored training recommendations, from fine-tuning individual strikes to refining entire fight strategies.

By combining biomechanics, physiology, and intelligent sequence analysis, MUGI establishes a new standard for martial arts performance tracking—transforming raw strike data into meaningful coaching guidance

Chapter 2 - Project Description

2.1 Project Background and Motivation

Martial arts, much like other disciplines of art, serve as a lens through which to interpret one's life. For those who dedicate enough of themselves to it, it even transforms into a way of life. One which starts as a warm embrace that gently guides you to increase confidence before turning into an inferno lit within your heart which pushes you towards self-actualization. It requires incredible amounts of **dedication, discipline, and repetition**. One of the ways that this takes form is in the thousands of intense heavy bag sessions. While doing this alone consistently may yield continued improvement, there is still space for efficiency in this improvement. This room is in **quantitative** and **qualitative** tracking of strikes. A fighter often relies on a pair of eyes to point out tendencies which they can then use to inform their training regimen. These eyes can come with their own set of biases and things they miss due to the limitations of human perception as time continues. We propose a **sensor based system with vision modularity** that can see past all these issues and towards a path of **calculated continuous improvement**.

While this may be an ambitious goal, we have worked to package it into a concise and deliverable Senior Design project. We considered the restrictions and purpose of a senior design project to do so. We understand a successful senior design project to be one where a problem is acutely identified then addressed then solved through well-defined methodologies and a success case that is quantifiable. We believe that this project fits that definition and by the end, we will hone current skills, acquire new ones, and solve our original problem.

2.2 Goals and Objectives

Our high-level goal is to provide a tool for martial artists to utilize to identify their natural striking dispositions as well as provide insightful advice into what direction to move towards for future training sessions. This tool will be composed of wearable sensors as well as central application components. We hope to accelerate a fighters' understanding of themselves as well as stimulate constant evolution.

Main Goals

- Real time counting of strikes thrown out of a defined set of strikes [Straight, Hook, Uppercut, Elbow, Body Shot]
    - Real Time heart rate monitoring
    - Post training session insight on strikes
    - Post Training information on heart rate in relation to strikes
- · Integration of computer vision in real time counting of strikes
- All data displayed in a clean UI on a local application

Main Objectives

- Create a system with a hardware component comprised of 2 IMU designs: one with 9 DoF, heart rate monitor, charging circuit, and MCU and the other with 9Dof, charging circuit, and MCU
    - Hardware component will be contained in housing and will be equipable with a strap on left and right wrist
    - Accelerometer, Gyroscope and Magnetometer output will be translated to quaternions
- ·Design deep neural network architecture which takes the input of quaternions from the IMU's and can identify strikes within that stream of data
    - This neural network architecture will have a strike identification percentage of >=80%
    - This architecture will be, at a high level, composed of a transformer and LSTM
- Create a quaternions dataset that is deep and broad enough to be used to train the neural network.
    - Neural network will run locally on machine where application is running
- · Deepen neural network architecture by adding the Alpha pose (CITE) pose estimation model + Transformer+LSTM+LLM to use spatiotemporal coordinates obtained through video to further support initial predictions provided by IMU data

Advanced Goals

- Post Training ‘pathway’ suggestions to move towards based on tendencies and specific strike occurrences observed

Advanced Objectives

- Add an LLM to the network architecture that can understand the strikes being identified as input and output coherent output which suggest 1-3 combos as well as general suggestions towards direction.

Stretch Goals

- Data displayed on an iPhone application
- Add two more IMU’s to system to extract data on kicks as well
- Real time skeleton reflection on Unity (digital twin)
- Add support for multiple users

Stretch Objectives

- Develop an iPhone application that runs locally on iPhones running IOS 16+. This app will not be on the app store.
- With addition to neural network architecture, machine learning components to be hosted on server to relieve tax on iPhone.
- Duplicate IMU design without heart rate monitor and use those to place on a right leg and left leg

2.2.1 Hardware Description

The sensor component’s primary function is to capture the absolute location of the appendage to which it is attached to. This is achieved by leveraging a 9 Degrees of Freedom (DoF) Inertial Measurement Unit (IMU), composed of an accelerometer, gyroscope, and magnetometer. Each of these sensors contributes uniquely to motion tracking: the accelerometer measures linear acceleration, the gyroscope measures angular velocity, and the magnetometer provides directional heading. When combined through sensor fusion algorithms such as the

Madgwick or Kalman filter, these data streams yield highly accurate representations of orientation in the form of quaternions or Euler angles.

The importance of the 9 DoF configuration lies in its ability to correct weaknesses present in lower-DoF setups. For example, a 6 DoF IMU (accelerometer + gyroscope) can track movement and rotation but suffers from drift and alignment issues over time. The addition of the magnetometer in a 9 DoF IMU anchors orientation to an external reference, reducing drift and providing greater long-term stability. This is especially critical in martial arts strike detection, where precise and repeatable motion data ensures that the system can differentiate between subtle strike variations (straight, hook, uppercut) and deliver reliable metrics for both training feedback and machine learning analysis.

From a hardware perspective, incorporating 9 DoF sensors ensures that strike tracking is not only accurate in the short term but also robust across extended sessions. The IMUs will be housed in custom enclosures with straps to secure them to the athlete's wrists, ensuring consistent alignment during motion. In addition, the hardware design includes a heart rate sensor, charging circuit, and microcontroller unit (MCU) to manage data acquisition and wireless transmission. The integration of these hardware elements provides a comprehensive data capture platform capable of supporting the project's broader objectives of real-time analysis, post-training insights, and machine learning–driven recommendations.

2.2.2 Software Description

The software component of this project serves as the 'intelligence' layer that interprets raw sensor data into meaningful insights. The primary function is to take quaternion outputs from the 9 DoF IMUs and process them in real time to identify specific strike types. This is accomplished by integrating sensor fusion algorithms such as the Madgwick filter, which converts accelerometer, gyroscope, and magnetometer data into stable quaternion representations. Quaternions are advantageous because they capture 3D orientation without the gimbal lock issues inherent in Euler angles, and they also provide smoother data streams well-suited for downstream machine learning models.

On top of this foundation, the software will employ a deep neural network architecture designed to classify strikes (e.g., straight, hook, uppercut) from the quaternion streams. The architecture is envisioned to include a transformer-based sequence model to recognize temporal strike patterns and a large language model (LLM) component to generate legible training suggestions for users. These layers allow not only strike classification but also contextual

recommendations, such as identifying common tendencies and proposing future strike combinations.

The software stack will also manage real-time strike counting, session logging, and heart rate monitoring integration. Data is displayed through a clean, user-friendly application interface, enabling athletes to view both real-time and post-session insights. Stretch objectives extend the software's capability to include computer vision modules like AlphaPose, which could add spatial-temporal coordinates from video to further support IMU predictions, and Unity-based visualization for a digital twin of the athlete.

Together, these software systems transform raw motion data into actionable feedback. By combining robust sensor fusion, advanced neural network models, and clear data visualization, the software ensures the project delivers reliable, interpretable, and performance-enhancing insights for martial artists.

2.3 Requirements and Specifications

System Requirements	
Parameter	Specification
Strike Classification Accuracy	≥ 80% correct identification of strikes
Heart Rate Monitoring	Continuous, ± 2 BPM accuracy
Size	≤ 50 mm × 40 mm × 15 mm (L × W × H)
Session Runtime	≥ 1 hour continuous operation
Wireless Data Transmission	≥ 95% packet success rate
Weight	≤ 60 g per wrist unit (including strap)
Total Cost	≤ $800 BOM

3 Demostationable:

1. **Strike Classification Accuracy:** The machine learning model demonstrates the ability to classify strikes with an accuracy exceeding 80% during testing trials.

2. **Heart Rate Monitoring:** The system's heart rate measurements are comparable to a commercial standard monitor, with a deviation of no more than ±2 bpm.

3. **Wireless Data Transmission:** Wireless data transfer is validated with a packet success rate of 95%, ensuring reliable communication of sensor outputs.

To fully reconstruct a signal per the Nyquist-Shannon theorem one must sample at least twice the highest frequency component of said signal. This directly relates to the output data rate (ODR) of our sensors, meaning in order to fully reconstruct our signal the ODR of each sensor must be >= twice the highest frequency component. We can approximate this using the relationship between time and frequency, that being f = 1/t, where time in this scenario represents impact duration(the brief period when the fist makes contact with the target). We focus on impact duration specifically as this is the critical phase where peak accelerations and forces occur.

Deriving the sample rate:
The average impact duration across various types of strikes last between 15 - 25ms. The shortest being 15ms, we will use this value for the calculations.

Minimum sample frequency : f = 1/t = 1/15ms ~= 66.67 Hz
Per Nyquist-Shannon Theorem the minimum ODR must be 2*66.67Hz ~= 133.333 data points per second

This aligns well with current IMU based performance tracking studies which recommend a sample rate between 200 - 400 Hz

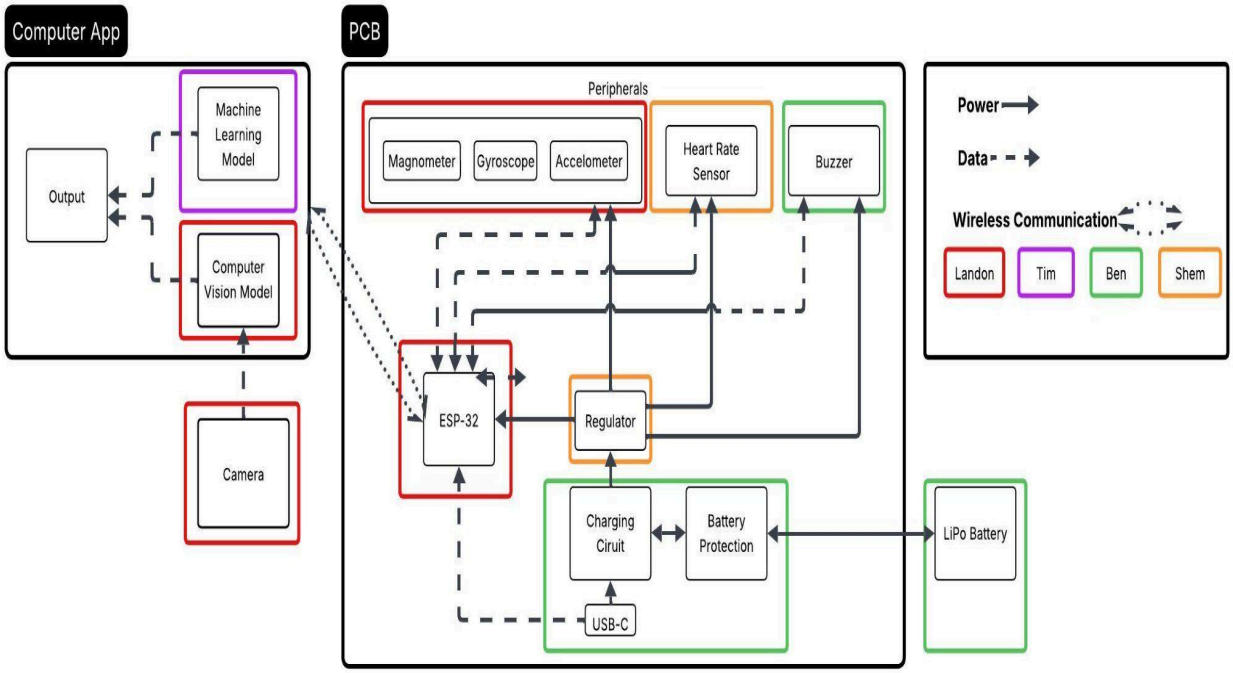
Practical Considerations:
In practice strike impacts can generate much higher frequencys components than previously mentioned because of abrupt acceleration changes during impact, within the range of 500 to 1000Hz. Per Nyquist-Shannon theorem that would mean a ODR of 1000 - 2000 points per second however, the computational load would be too great resulting in lower battery life and increased memory use. Especially considering there are diminishing returns on

the practical metrics we are interested in extrapolating in these higher frequency components.

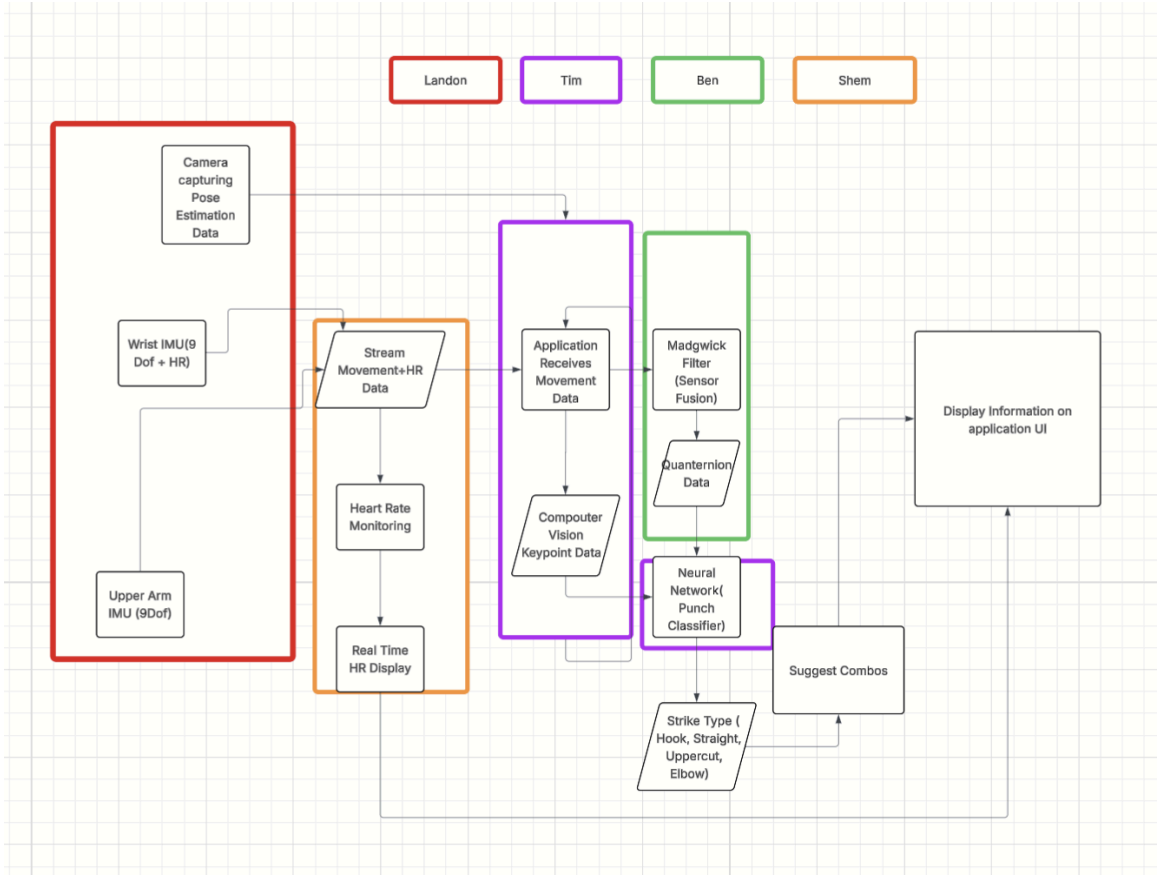
Component Level Specifications		
Component	Parameter	Specification
Accelerometer	Full Scale Range	± 16 g
Accelerometer	Output Data Rate	≥ 200 Hz
Gyroscope	Full Scale Range	± 2000 °/s
Gyroscope	Bias Stability	≤ 10 °/h
Magnetometer	Range	± 65 μT
Magnetometer	Output Data Rate	≥ 100 Hz
MCU	Processing Latency	≤ 10 ms per packet
MCU	Wireless Protocol	Bluetooth Low Energy (BLE)
Heart Rate Sensor	Accuracy	± 2 BPM vs reference
Heart Rate Sensor	Sampling Rate	≥ 25 Hz
Power System	Battery Life	≥ 1 hour
Power System	Charging Time	≤ 2 hours
Application (Software)	Strike Count Accuracy	≥ 80%
Application (Software)	Suggestion Generation	Human-readable training advice

2.4 System Design

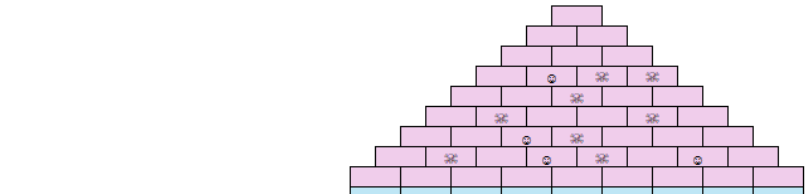
2.4.1 Hardware



2.4.2 Software



2.5 House of Quality



	Customer Requirements \ Design Requirements	Direction of Desirability	1 Strike Accuracy	2 HR Accuracy	3 Latency	4 Weight	5 Cost	6 Battery	7 UI Update	8 Wireless Data Transfer	9 Drop resistance
	Direction of Desirability		1	1	0	0	0	1	1	1	1
1	Accurate Strike Recognition	1	(+)		(+)				+	+	
2	Reliable heart rate monitoring	1		(+)	+						
3	Low system latency	1	(+)	+	(+)			-	(+)	+	
4	Comfortable to wear	1				(+)		-			+
5	Affordable device	1	+	+	+	+	(+)	-	+	+	+
6	Long battery life	1		+	-	-	-	(+)		-	
7	Easy-to-use app	1	+		+				(+)	+	
8	Reliable wireless connectivity	1	+	+	+			-	+	(+)	
9	Durable under use	1	+				+	+		+	(+)
10	Lightweight and stylish	1				(+)	+	-			+
	Product Targets		≥ 60% correct	± 2 BPM	≤ 10 ms per packet	≤ 60 g per unit	≤ $800 BOM	≥ 1 hour	≥ 10 Hz	≥ 95% packet success	≤ 4feet

Correlation:
Positive ☺
Negative ☹
Nocorrelation

Direction of desirability:
High 1
Low 0

Relationship:
Strongly Positive (x)
Weakly Positive +
No Relation
Weakly Negative -
Strongly Negative (-)

Chapter 3 - Hardware

3.1 Regulators

3.1.1 Introduction

A voltage regulator is one of the most critical components in modern electronics. Its role is to maintain a stable, constant output voltage regardless of fluctuations in input voltage or changes in load current. In battery-powered and portable systems—such as wearables—this stability is essential for maintaining reliable operation, preventing data loss, and protecting sensitive components such as microcontrollers and sensors [1][2].

Voltage regulators exist at every level of the electronic hierarchy—from high-voltage power supplies feeding industrial controllers to the miniature regulators embedded within smartwatches and wireless sensors. Without proper regulation, even small variations in supply voltage can cause significant functional errors, including analog drift, digital logic malfunction, or complete system reset [3].

Modern regulators not only stabilize voltage but also handle power conversion, noise suppression, and protection against transient disturbances. In wearable systems, this means providing clean rails for microcontrollers, analog sensors, wireless transceivers, and audio outputs—all of which demand stable operation at low voltages and limited power budgets.

3.1.2 Different Types

Voltage regulators are broadly divided into linear and switching types. Each uses a fundamentally different approach to maintaining constant output voltage:

Different Types of Regulators				
Regulator Type	**Method**	**Efficiency**	**Output Noise**	**Typical User**
Linear	Dissipates excess voltage as heat	40–70 %	Very low	Analog & sensor circuits
Switching	Stores and transfers energy using inductors/ switching transistors	80–95 %	Moderate	Digital & high-power loads

3.1.2.1 Linear Regulator

Linear regulators are favored for simplicity and low noise, whereas switching regulators are preferred when efficiency and heat management are critical. In practice, most systems—including wearable devices—use a combination of both: a switching regulator for the main supply rail and linear post-regulators for low-noise subsystems [4][5].

A linear regulator controls its output by continuously varying the resistance of a pass element (typically a BJT or MOSFET) in series or parallel with the load [6]. The feedback network compares the output to a precise reference voltage and adjusts the transistor's conduction to correct any deviation. This method yields fast response and extremely low noise, making it ideal for analog circuits or sensors where ripple cannot be tolerated [7].

However, because linear regulators dissipate the voltage difference between input and output as heat, their efficiency decreases rapidly with larger voltage drops or high output currents. The power loss is:

Ploss = (Vin − Vout) × Iout
$P_{loss} = (V_{in} - V_{out}) \times I_{out}$
Ploss = (Vin − Vout) × Iout

Shunt Regulators

A shunt regulator maintains voltage by routing excess current away from the load. It is typically connected in parallel with the load and uses a Zener diode or transistor to conduct when the voltage exceeds a set threshold [8].

Zener diodes are particularly common in small-signal applications. When reverse-biased beyond the breakdown voltage, they maintain a nearly constant voltage, allowing the circuit to stabilize variations in supply or load [9].

While simple and inexpensive, shunt regulators are inefficient because current always flows through the shunt path—even when the load current is low. This constant current makes them unsuitable for portable or high-power systems, though they are still used for voltage references or protection circuits.

Example Applications: voltage reference circuits, microcontroller reset lines, and low-power analog biasing [10].

Series Regulators

A series regulator connects its control transistor in series with the load. The transistor’s resistance changes dynamically to maintain a constant output voltage [11]. When the load voltage drops, the regulator increases conduction; when it rises, it reduces current.

Compared to shunt regulators, series designs are far more efficient because current only flows through the load path. They also offer better control accuracy, improved line and load regulation, and higher output currents. The downside is thermal dissipation, as excess voltage is dropped across the series transistor [12].

Series regulators form the basis for most integrated linear regulator ICs, including the classic 78XX and 79XX series, which offer fixed positive or negative voltages with currents up to 2 A [13].

Video Reference: The tutorial “Series Regulators” [14] demonstrates that series designs improve upon shunt regulators by reducing waste in the series resistor and decreasing current stress on the reference element.

Low Dropout(LDO) Regulators

A low-dropout (LDO) regulator is a refined series regulator capable of operating with very small input-to-output voltage differences (the dropout voltage). LDOs are particularly important in systems powered by batteries, where supply voltage can fall close to the desired output as the cell discharges [15][16].

In PMOS-based LDOs, the gate-to-source voltage is negative; increasing the input voltage increases the negative bias, lowering resistance and minimizing dropout. NMOS architectures, by contrast, require an additional bias or charge-pump circuit to drive the gate above the input voltage, achieving comparable dropout performance [17].

LDOs are widely used to power analog and RF subsystems because of their low noise, low ripple, and high power-supply rejection ratio (PSRR). The trade-off is lower efficiency at large voltage differences, and limited current capacity compared with switching designs [18].

Video Reference: Texas Instruments, “What is an LDO?”, demonstrates how LDOs maintain regulation even when Vin is only slightly above Vout [19].

3.1.2.2 Switching Regulator

Unlike linear designs, switching regulators control voltage by rapidly switching a transistor between on and off states and filtering the resulting pulsed waveform through inductors and capacitors. This switching action transfers energy in discrete packets, allowing extremely high efficiency (up to 95 %) [21].

The ratio of “on-time” to total cycle time—known as the duty cycle—determines the average output voltage. By modulating this duty cycle through feedback control, the regulator maintains a steady output despite varying load or input conditions [22].

Switching regulators are classified into buck, boost, and buck-boost topologies.

Buck Converter (Step Down)

A buck converter produces a lower output voltage than its input. When the transistor switch is on, current flows through the inductor, storing magnetic

energy. When the switch turns off, the inductor releases energy to the load via a diode, maintaining current flow [23].

Buck converters are common in portable and embedded devices because they efficiently convert battery voltage to logic-level rails such as 3.3 V or 5 V. Proper inductor and capacitor selection is critical to minimize ripple and electromagnetic interference (EMI) [24].

Video Reference: “Buck Converter Explained” [25] shows how adjusting the duty cycle controls output voltage, demonstrating the energy transfer process and filtering requirements.

Boost Converter (Step-Up)

A boost converter raises a lower input voltage to a higher output. Energy is stored in the inductor when the transistor is on, and released when the switch is off, effectively adding the stored energy to the input [26].

Boost converters are used when a device must maintain constant operation even as its supply voltage drops—such as in LED drivers, sensor networks, and battery-powered communication devices [27].

Video Reference: “Boost Converter Basics” [28] (Texas Instruments) illustrates the operation and efficiency considerations of this topology.

Buck-Boost Converter

The buck-boost converter can either increase or decrease voltage depending on the input conditions. This topology is particularly valuable in systems powered by single-cell lithium-polymer batteries, where the supply varies from 4.2 V fully charged to 3.0 V discharged [29].

By automatically stepping up or down as required, the converter maintains a stable voltage output—such as the 3.3 V required by digital logic—even as the battery voltage crosses that threshold [30].

Although buck-boost converters introduce additional ripple and control complexity, they provide unmatched flexibility and are widely used in portable, battery-dependent applications such as wearables and IoT nodes [31].

3.1.3 Performance Metrics

Performance Metrics		
Metric	Definition	Typical Target
Line Regulation	Change in output voltage per change in input voltage	< 0.05 %/V
Load Regulation	Change in output voltage per change in load current	< 0.1 %
Dropout Voltage	Minimum Vin − Vout for proper regulation	100–300 mV (LDO)
PSRR	Ability to reject ripple/noise from input	> 60 dB for sensitive circuits
Thermal Stability	Drift of output with temperature	< 0.01 %/°C

Line regulation measures how well the output resists input voltage fluctuations, while load regulation measures its ability to maintain output when current draw varies. Both metrics are essential for ensuring reliable operation in dynamic environments [20].

3.1.4 Comparison

Comparison					
Topology	Function	Efficiency	Complexity	Output Ripple	Use Case
Linear (Shunt/Series)	Step-down	40–70 %	Low	Very low	Low-power analog
LDO	Step-down (small ΔV)	60–85 %	Low	Very low	Battery sensors
Buck	Step-down	85–95 %	Moderate	Moderate	Logic rails
Boost	Step-up	80–90 %	Moderate	Moderate	LED drivers
Buck-Boost	Step-up/down	75–90 %	High	Higher	Battery systems

3.1.5 Applications

In wearable systems, the challenge lies in balancing efficiency, size, and noise performance. The power source—a 3.7 V nominal Li-Po cell—must provide two main rails:

- 3.3 V for digital logic and communication, and
- 1.8 V for low-voltage analog and sensor cores.

Because the battery voltage both exceeds and drops below 3.3 V during discharge, a buck-boost topology is ideal for the primary rail. The secondary rail can be derived from the 3.3 V output using a low-noise LDO, providing stable operation for the IMU and optical sensors without additional switching ripple.

Peak current demands (≈ 300 mA for microcontroller and wireless transmission bursts) dictate regulator sizing, while component placement and decoupling are crucial for minimizing conducted and radiated noise [32][33].

This dual-stage regulation architecture—buck-boost plus LDO—provides a balance of efficiency, reliability, and noise isolation, ensuring consistent operation across the entire battery voltage range.

3.1.6 Summary

Voltage regulators form the foundation of any reliable electronic design. Linear regulators offer simplicity and low noise; switching regulators provide efficiency and flexibility. In wearable systems where both battery efficiency and sensor precision are essential, combining these technologies achieves optimal performance. The theoretical understanding of regulator operation, performance metrics, and design considerations serves as the basis for detailed hardware implementation in later sections.

3.2 Level Shifters

3.2.1 What Are They?

A level shifter (a.k.a. logic-level translator) converts digital signals between voltage domains—for example, from a 1.8 V sensor to a 3.3 V microcontroller—so that logic-high/low thresholds are met and pins are not overstressed [34]. Modern systems commonly mix 1.8 V, 3.3 V, and 5 V logic; direct connection can fail because a “high” in one domain may not satisfy VIH in another, or worse, 5 V could damage a 3.3 V-only input [35].

Level shifters translate logic, not power; unlike regulators, they provide signal compatibility without sourcing substantial current to the load [36], [37].

3.2.2 Types

Common categories include [34]:

- Uni-directional: Fixed direction (e.g., 5 V → 3.3 V).
- Bi-directional with dedicated ports: Each domain has defined inputs/outputs.
- Bi-directional with direction pin: Direction controlled by a dedicated signal.
- Bi-directional auto-sensing: Automatically senses direction (popular for open-drain buses like I²C).

3.2.3 Implementations

- Resistor Divider (uni-directional): Simple attenuation (e.g., 5 V → 3.3 V). Suitable for slow inputs or unidirectional lines like some GPIO/UART RX. Not ideal for high-speed edges due to RC delay [35].
- Diode/Clamp: Schottky diode clamps to protect inputs; crude, not precise for logic thresholds.
- MOSFET-Based Bi-Directional Shifter: A small N-MOSFET (e.g., BSS138) with pull-ups on each side supports open-drain buses (I²C) with automatic directionality; widely used on hobby breakouts [35].
- Dedicated Translator ICs: Devices like PCA9306 (I²C level translator) or TXS0108E (8-bit auto-direction) provide controlled edges, ESD protection, and higher speed, and are better for push-pull interfaces and wider buses [33], [35].

3.2.4 Communication, Speed, and Signal Integrity

- I²C (open-drain): Use MOSFET-based or I²C-specific translators; ensure separate pull-ups in each domain and check total bus capacitance vs clock rate [35].
- SPI (push-pull): Prefer uni-directional translators or buffer ICs with defined direction and adequate bandwidth; MOSFET auto-sensing types can distort edges.

- UART: Often uni-directional per line (TX, RX). Resistor dividers can work on RX into a 3.3 V device; verify the 3.3 V output meets the VIH of the higher-voltage side, or use a translator/buffer [36], [37].

3.2.5 Performance

- Logic Thresholds: Ensure translated VOH/VOL meet receiving device VIH/VIL across PVT (process/voltage/temperature).

- Edge Rate and Bandwidth: Dividers and MOSFET-style shifters add capacitance and can slow edges; high-speed push-pull buses need buffer translators with specified Mbps ratings [35].

- Pull-Ups and Capacitance: On I²C, each side requires its own pull-up to its domain. Total bus capacitance limits rise time and maximum clock speed.

- Fail-Safe and ESD: Prefer translators with fail-safe I/O and integrated ESD diodes when interfacing off-board connectors.

- Direction Control: Where possible, using fixed-direction devices reduces uncertainty and improves timing.

Level Shifter Options				
Method	Direction	Pros	Cons	Best Use
Resistor Divider	Uni	Ultra simple, cheap	Slow edges, uni-direction only	UART RX, slow GPIO
Diode Clamp	Uni	Protects inputs	Poor logic thresholds	Over-voltage protection
MOSFET (BSS138)	Bi, auto	Great for I²C, simple	Slow at high speed, pull-ups required	I²C up to moderate kHz
PCA9306 / TXS0108E	Bi (I²C) / Bi (push-pull)	Specified speed, ESD, robust	Cost, package size	SPI/UART/GPIO at MHz, mixed buses

3.2.6 Layout, Best Practices, and Comparison

Place translators close to the receiving device to minimize stub length. Keep separate pull-ups per domain (for open-drain buses). Route short, clean traces with a continuous reference plane to maintain signal integrity. For connectors or long cables, add series resistors (22–100 Ω) to damp ringing and consider ESD protection diodes. Validate timing (setup/hold) after translation at your target data rate [33], [35].

3.3 IMU

3.3.1 Introduction

An Inertia Measurement Unit (IMU) is an integrated sensor module, or the combination of an accelerometer, gyroscope and optional magnetometer, designed to measure linear acceleration, angular velocity, and magnetic field strength across three axes (X, Y, Z). By combining these measurements an IMU can provide real-time estimates of an object's orientation, position, and motion.

There are two primary types of IMUs:

- 6 Degrees of Freedom (6DoF): Combines a 3 axis accelerometer and a 3-axis gyroscope. This configuration can determine orientation and motion but is prone to drift over time due to accumulated integration errors.
- 9 Degrees of Freedom (9DoF): Similar to the 6DoF configuration but added a 3-axis magnetometer. This magnetometer can reference Earth’s magnetic field, providing an absolute heading that can correct drift issues that are prone in the 6DoF design.

For the Strike Sense system, a 9DoF IMU provided the best orientation stability and synchronization across multiple sensor nodes. This will prove to be critical for accurate motion classification.

3.3.2 Accelerometer

A MEMS accelerometer measures acceleration by detecting displacement of a micro mass suspended by springs. Variations in the displacement change the

capacitance of interdigitated plates, this capacitance is then converted to a proportional electrical signal.

Key Specification

- Range(±g): The maximum measurable acceleration before sensor saturation
- Sensitivity: Defines how much the digital output changes per unit of acceleration
- Noise Density: The RMS noise level; lower values will improve small-signal resolution
- Bandwidth/ODR: Output data rate defines the frequency response and update rate
- Bias and Drift: Offsets variation acquired overtime, requiring calibration
- Shock Survivability: Determines the robustness against impact forces

Accelerometer Specification		
Parameter	**Typical Value**	**Impact**
Range	±2 to ±16 g	Defines dynamic range
Noise Density	100–500 μg/√Hz	Affects resolution
ODR	100–3200 Hz	Captures motion fidelity
Bias Stability	Few mg	Influences static accuracy
Shock	Up to 10,000 g	Durability under strikes

Integration and Filtering

For high-impact activities (punching), accelerometers must capture fast events without clipping. A high ODR (500-1000 Hz) ensures accurate capture, while low-pass filtering reduces noise. Calibration will remove bias and temperature drift.

3.3.3 Gyroscope

A MEMS gyroscope measures angular velocity by detecting the Coriolis effect on a vibrating mass. As the device rotates, the mass experiences a force proportional to its rotational speed. This force is detected using capacitance and then translated into an electrical signal representing the angular rate on the X, Y, and Z axis.

Key Specifications

- Range (°/s): Typical ranges include ±250, ±500, ±1000, and ±2000 °/s.
- Noise Density: Determines stability of angular rate readings; low noise will improve short-term accuracy
- Bias Drift: The slow variation of zero-rate output overtime or temperature (ie reading 0.001 on any axis for a stationary device)
- Bandwidth/ODR: Defines how fast angular data is updated; 200-1000Hz is common for motion analysis

Applications

Gyroscope provides short-term rotational accuracy but suffers from drift over time. When fused with accelerometer and magnetometer data using an AHRS algorithm, they can yield precise 3D orientation tracking.

3.3.4 Magnetometer

A magnetometer measures the strength and direction of magnetic fields. It converts the density of magnetic flux into electrical signals. This ensures the IMU can determine its heading in relation to the Earth's magnetic field, providing absolute orientation. The magnetometer will help compensate for gyroscope drift, therefore maintaining long-term accuracy of the system.

Types of Magnetometers

- Hall-Effect: Simple and low-cost, measures voltage changes induced by a magnetic field.
- Fluxgate: Uses ferromagnets for high-sensitivity magnetic field measurements
- Magnetoresistive: Detects resistance changes due to magnetic field variations, compact, low-power and precise.

Magnetometer Comparison			
Parameter	**MMC5983MA (Memsic)**	**LIS3MDL (STMicro)**	**BMM150 (Bosch)**
Range	±8 G	±4/±8/±12/±16 G	±13/±25 G
Resolution	0.25 mG	0.14 mG	0.3 μT (~3 mG)
RMS Noise	0.4 mG	1.5–2 mG	6 mG
Max ODR	1000 Hz	1000 Hz	300 Hz
Voltage	1.7–3.6 V	1.9–3.6 V	1.62–3.6 V
Current	20 μA	100 μA	170 μA
Special Features	SET/RESET Coil	Temp Sensor	Low Power
Suitability (High-Speed)	Excellent	Moderate	Limited

Conclusion
The MMC5983mA offers great precision and stability. However a 6+3 DoF system is not preferred and will introduce additional synchronization errors.

3.3.5 IMU Comparision

IMU Comparision						
IMU	**DoF**	**Max ODR**	**Range (Accel / Gyro / Mag)**	**Power**	**Interface**	**Special Features**

IMU Comparision						
ICM-20948	9	1125 Hz	±16 g / ±2000 °/s / ±4900 µT	2.5 mA	I²C / SPI	DMP, sensor fusion engine
MPU-9250	9	1000 Hz	±16 g / ±2000 °/s / ±4800 µT	3.5 mA	I²C / SPI	Older generation, higher drift
Dual-IC Solution (ICM-20649 + MMC5983)	6+3	1000 + 1000 Hz	±30 g / ±4000 °/s / ±8 G	3.2 mA (combined)	I²C	Modular, but less synchronized

Selection and Justification –ICM-20948
After evaluating, the TDK InvenSense ICM-20948 was the best fit for the chosen role. This IC integrates a 3-axis accelerometer, 3-axis gyroscope, and a 3-axis magnetometer forming a 9DoF sensor; this immediately stood out from all 6+3 DoF solutions for the ease of integration and synchronization, simplifying the PCB and software design. Among the 9DoF solutions only the ICM-20948 features an integrated digital motion processor (DMP) which can execute, on chip, AHRS filters. Other key advantages is the low power consumption and the configurable ODR making it ideal for a wearable device balancing precision with power efficiency.

3.4 MCU

3.4.1 Intro

The MCU is the heart of our hardware design and plays a key role in bridging the hardware and software. The project requirements demand some minimum basic requirements; low power design, native wireless communication support, small form factor, SPI/I2C peripheral support, and adequate processing power/memory depending on our selected computational load.

The MCU is the heart of our hardware design and plays a key role in bridging the hardware and software. The project requirements demand some minimum basic requirements; low power design, native wireless communication support, small

form factor, SPI/I2C peripheral support, and adequate processing power/memory depending on our selected computational load.

3.4.2 Computation Load Tradeoffs

This project presents two avenues, and the choice of which greatly impacts our computational load. The Inertial Measurement Unit (IMU) generates a raw data stream including; accelerometers-capturing linear acceleration of multiple axis, gyroscopes-capturing angular velocity and a magnetometer-measuring the earth's magnetic field. All sampling at a high rate (200-400Hz). This data is crucial to the machine learning model to identify strikes, but feeding this model raw data versus applying pre-processing(on-chip) e.g. using sensor fusion algorithms to derive quaternions, creates tradeoff especially relating to accuracy, latency, power consumption and computational load. Computational load is measured by MCU cycles, memory usage, and execution time.

Advantages of Raw Data approach:

- This approach greatly reduces the computation load as no sensor fusion algorithms run on the MCU. This shifts this computation complexity to the host computer running a powerful deep learning model
- Lower Latency, direct streaming avoids delays from popular sensor fusion algorithms, approximately saving 1-3ms per iteration.
- Power savings

Disadvantages:

- Raw data is prone to errors resulting in reduced accuracy
- Increased noise resulting in reduced accuracy
- Larger data size, increasing transmission bandwidth

Advantages of Pre-Processing:

- Improved Machine learning efficiency
- Noise Reduction

Disadvantages:

- Increased MCU load
- Increases latency

3.4.3 Comparing MCU’s

3.4.3.1 Espressif vs Nordic Semiconductor vs Raspberry Pi

When evaluating various MCU families for embedded sensor fusion plus wireless data streaming, the decision will depend on trade-offs among connectivity, processing power and power efficiency

MCU Comparisons			
	Espressif (ESP32 series)	Nordic Semiconductors (nRF52/53 Series)	Raspberry Pi (RP2040)
Architecture	Xtensa LX6/LX7 or RISC-V (dual-core and single core options)	Arm Cortex-M4F (nRF52) or dual-core Cortex-M33 (nRF53)	Dual-core Arm Cortex
Memory	320-512KB SRAM + PSRAM/Flash up to 16MB	512 KB-1 MB Flash, 256-512 KB RAM	264 KB SRAM, external Flash up to 16 MB
Wireless Connectivity	Wi-Fi 4/6 and Bluetooth 5 LE	Bluetooth 5 LE	None requires external radio
Power Efficiency	Optimized deep sleep ~5-10uA but higher active draw ~100-300mA	Excellent deep sleep ~3-6 uA BLE optimized	Moderate heavily depends on external radio
Peripherals	Up to 45 GPIO; ADC (12-bit); I²C ×2; SPI ×4; UART ×3; USB-OTG; PWM; I²S audio support	48 GPIO; ADC (12-bit); I²C/SPI/UART; PWM; QDEC; PDM microphone interface	30 GPIO; ADC (12-bit); I²C/SPI/UART; 2 × PIO (custom state machines); PWM
Ecosystems	ESP-IDF (C/C++), Arduino Core, MicroPython support	nRF Connect SDK (Zephyr RTOS)	Pico SDK(C/C++), CircuitPython, MicroPython
Price	$6-$15	$10-$20	$4-$8

3.4.3.2 SoCs vs Modules

Espresif sells both SoCs and Modules. It is important to understand this difference. SoCs (System-on-Chip) is just the silicon chip itself, this includes the CPU cores, Wi-Fi/Bluetooth radios, peripherals and memory controllers. However various external components are required such as; flash memory, antenna, power management and crystal oscillators. This option will require extensive PCB design, the benefit being control over this design. Modules are ready-to-use packages that include the ESP32 SoC plus the required external parts mentioned above. These designs are pre-certified for regulatory compliance.

3.4.3.3 ESP-32 Chip Series

The line of ESP-32 chips from Espressif is divided into four series, each optimized for different needs.

S-Series
Optimized for general-purpose and AI-enhanced applications. Key models include the; ESP32-S2, ESP3- S3.

S-Series Breakdown				
Processor	Memory	Connectivity	Peripherals	Power
S2: Xtensa LX7 single-core 240 MHz S3: dual-core 240 MHz, Optional low-power RISC-V co-processor	S2: 320 KB SRAM, 128 KB ROM, external Flash (up to 16 MB), no standard PSRAM; S3: 512 KB SRAM + 16 KB RTC, 384 KB ROM, external	2.4 GHz Wi-Fi 4 Bluetooth 5 LE	GPIO: 43(S2)/45(S3) USB OTG ADC (12 bit) I2C(2 Instances) SPI(4 Instances) UART (2-3 Instances)	Deep Sleep: ~7-10uA Max Use: S2: 130-200 mA S3: 240-340 mA

S-Series Breakdown				
	Flash (up to 32 MB), PSRAM (2-16 MB)			

C-Series
Optimized to be a cost effective and low-power IoT solution. Key models include ESP32- C2, C3, C5, C6.

C- Series Breakdown				
Processor	Memory	Connectivity	Peripherals	Power
RISC-V single core C2: 120 MHz C3: 160 MHz C5: 240 MHz C6: 160 MHz Optional low-power co-processor (C5,C6)	C2: 272 KB SRAM (16 KB cache), 576 KB ROM, external Flash/PSRAM (up to 8 MB each); C3: 400 KB SRAM, 384 KB ROM, external Flash/PSRAM; C5: 384 KB HP + 16 KB LP SRAM, 320 KB ROM, external Flash (up to 4 MB); C6: 320-512 KB SRAM + 16 KB LP, 256-512 KB ROM, external Flash/PSRAM (up to 8 MB)	C2/C3: 2.4 GHz Wi-Fi 4 C5: Dual-band 2.4 / 5 GHz Wi-Fi 6 C6: 2.4 GHz Wi-Fi 4 All include Bluetooth 5 LE	GPIO C2/C3: 22 pins C5: 29 pins C6: 31 pins 6-8 ADC channels (12 bit) I2C(2 Instances) SPI(4 Instances) UART(2-3 Instances)	Deep Sleep: ~5-7 uA Max Use: C2: 80-150 mA C3: 100-180 mA C5: 150-250 mA C6: 120-200 mA

P-Series
Optimized for high-performance human-machine interfaces (HMI). Currently only the ESP32-P4

P Series				
Processor	Memory	Connectivity	Peripherals	Power
RISC-V High performance dual-core + low performance single core	128 KB HP ROM 16 KB LP ROM, 768 KB HP L2MEM 32 KB LP SRAM 8 KB SPM	2.4 / 5 GHz Wi-Fi 6 Bluetooth 5 LE	GPIO: 55 pins; CSI/DSI;USB 2.0 OTG; SDIO 3.0; I3C, SPI, I2S, I2C; PWM; ADC; UART	Deep Sleep: ~5uA Max Use: 300-500mA

H-Series
Optimized for low-power wireless protocols without Wi-Fi. Currently only the ESP32-H2

H Series				
Processor	Memory	Connectivity	Peripherals	Power
RISC-V single-core (6MHZ)	320 KB SRAM 4 KB LP 128 KB ROM External Flash (2-4MB) No PSRAM	Bluetooth 5 LE (long range mesh) Zigbee 3.0 Thread 1.3	GPIO: 19-26 ADC (12-bit); I2C (2) SPI (3) UART (3) LED PWM;	Deep Sleep: ~5 uA Max Use: 50-100mA

3.4.3.4 WROOM vs MINI vs PICO vs SOLO vs PICO-MINI

The ESP32 family comes in a variety of module variants, balancing factors such as size, ease of integration, and performance for target use cases. These variants are built around the various chip series mentioned above. Some points of differences include form factor, antenna integration, integrated components, GPIO exposure and power efficiency. WROOM and SOLO are larger plug-and-play modules (SOLO is essentially a variant of WROOM modules based on a single core SoC). MINI is a smaller version of WROOM modules. PICO are ultra compact SoCs (require extensive PCB design). PICO-MINI is an ultra-compact SiP (System-in-Package).

Form Factor	
WROOM/SOLO	18mm x 25.5-31.4mm
MINI	13.2-15.4mm x 12.5-21.3mm
PICO	7mm x 7mm
PICO-MINI	13.2mm x 11.2 - 16mm

Antenna Options	
WROOM/SOLO/MINI/PICO-MINI	Integrated PCB antenna (standard) u.FL/IPEX connection (U variants)
PICO	Requires External

3.4.4 Memory

ROM (Read-Only Memory): Internal, non-volatile memory programmed during manufacturing. Can't be changed by the user. This memory holds essential low-level code like, bootloader, secure boot feature, core firmware libraries.

SRAM (Static Random Access Memory): Main internal volatile memory, used for runtime operations, contains; variables, stacks, heaps and buffering data. Fast and directly accessible by the CPU

PSRAM (Pseudo-Static RAM): Act as additional external SRAM. Useful in memory intensive tasks. PSRAM is connected via an SPI bus as Flash.

Flash (SPI Flash or QSPI Flash): Typically external non-volatile memory, used for storing program code. Externally connected via an SPI interface, or faster variants like QSPI. Flash is slower than SRAM, so code is often loaded into SRAM at runtime. Some modules have implemented internal Flash.

3.5 Heart Rate Monitor

3.5.1 Why It's Needed

With one of the main goals of StrikeSense being a quantitative and qualitative view of training, providing insight into how the body responds would align with that goal. The system chosen to focus on was cardiovascular. Heart rate is the simplest window into how hard the body is working. It rises with effort, falls with recovery, and tells us how the cardiovascular system responds to stress. By watching heart rate during sessions, StrikeSense can show when an athlete is training in the right zone for the day.

Resting heart rate and trends across the week help flag recovery status. A higher than usual resting rate can point to poor sleep, dehydration, or illness. Seeing

that change before hard work begins lets the athlete pull back or modify the plan. Pairing heart rate with perceived effort makes the picture even clearer and helps athletes learn what “easy,” “steady,” and “hard” actually feel like.

Heart rate variability adds another layer. It reflects how the nervous system balances stress and recovery. Lower variability over several days can warn that the body needs lighter work. Higher variability often signals readiness to push. Together, these signals guide smarter workloads, reduce injury risk, and build consistency. For a combat athlete who needs power, speed, and repeatability, cardiovascular insight turns raw effort into informed training.

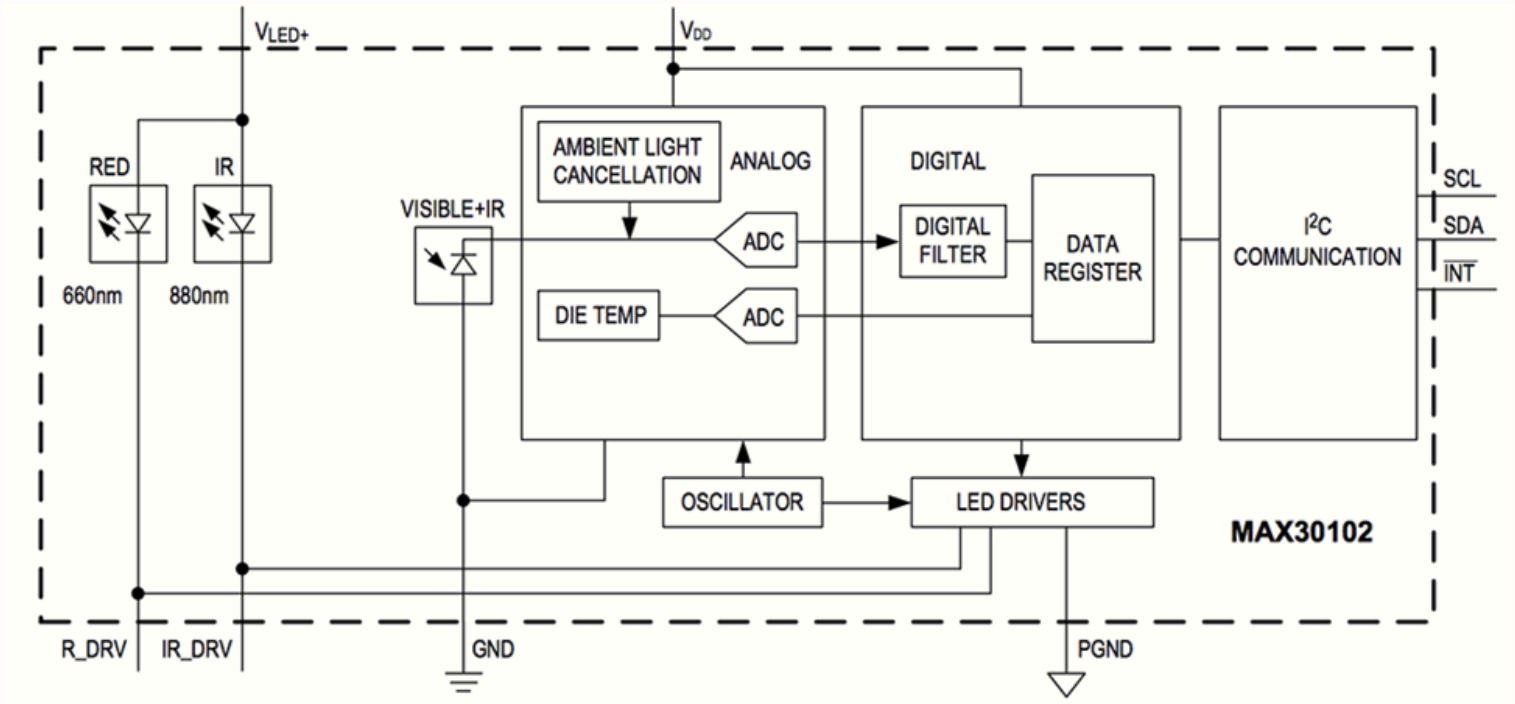
3.5.2 How Does it Work

In order to reap the benefits of heart rate monitoring, MAX30102 was selected as the module to carry out the task. The MAX30102 makes sense for StrikeSense because it delivers clean optical heart signals in a tiny, power-friendly package that keeps our board simple. It integrates the red and IR LEDs, a photodiode, a low-noise analog front end, an on-chip ADC, a FIFO, and I²C control, so we avoid a pile of discrete analog parts and the tuning that comes with them. The FIFO and an interrupt pin let us read samples in bursts and let the microcontroller sleep between reads to lower average current. On the firmware side we can set sample rate, pulse width, and LED current at runtime, timestamp each frame, and line it up with IMU events so beat detection is deterministic.

During an intense training session on the bag, sweat, movement, and impact from punches get introduced. These variables can impact contact pressure which in turn could bury any signal we are getting in noise. The MAX30102 provides us with enough tools to navigate this issue. Ambient light cancellation helps when a gap opens at the edges, and programmable LED current lets us trade power for signal quality when the session heats up. We can bump LED current during a fast combination to preserve signal-to-noise, then dial it back once we detect recovery to save battery. The on-die temperature readout supports slow baseline correction as the sensor warms up on skin. Mechanically, the compact package works behind a clear window with a thin foam standoff or a gentle strap bias, which helps maintain firm contact without discomfort over longer sessions. Our approach is to construct based on nodes on a band. With the MAX30102, as long as the module is on the node closest to the arteries, we will have access to the data we need.

With just a few components, the MAX30102 can provide great insight into the cardiovascular system. The device is both a pulse oximeter and a heart-rate monitor. It shines red and infrared light into the skin and measures the returning light with a photodetector to produce a photoplethysmogram. When the heart pumps, arterial volume changes and so does the reflected light, which forms a waveform from which we detect beats and compute heart rate. Because oxygenation changes the relative absorption of red and IR wavelengths, the same hardware also supports estimating blood oxygen saturation.

The MAX30102 communicates over I²C through the SDA and SCL data lines. There is a built-in FIFO buffer for 32 to 256 data samples. Once the limit is reached, new data is brought in to overwrite the old data. This allows us not to have to service every frame in real life. Interrupts can be used to signal important events such as fresh data being ready or power on status. That combination makes our firmware loop simple: configure the LED mode, sample rate, and pulse width, sleep until the interrupt fires, burst-read the FIFO, and push frames into our time-aligned pipeline with the IMU data. In the figure below, you can see a diagram of the MAX30102.



Bringing these pieces together lets us keep the microcontroller focused on sensor fusion and strike classification while the sensor handles optical acquisition. We get a stable heart-rate stream for training load, recovery, and

more well-rounded session summaries. The overall approach matches how MAX30102 modules are intended to be used and aligns with common guidance for PPG projects.

3.6 Power Supply Unit

3.6.1 Battery Technologies

In our project, our goal is to create a lightweight sensor that can identify a fighter's strikes. While also alerting the user to adjustments that they can make to improve their fighting technique. The importance of selecting a specific battery is to ensure long-term use during each training session, so the user doesn't have to constantly change batteries or recharge the system. Additionally, we aim to design a system that does not interfere with the fighter's experience and is comfortable to use. Some of the safety requirements that need to be considered include temperature behavior and overcharging to prevent any injuries to users. We researched a range of batteries that can be used to power the system below. We will discuss the various battery conditions and will end up with one that works best for our system.

3.6.1.1 Lithium Ion (Li-ion)

Li-ion batteries are a commonly used technology for many different electronic products. Some typical chemistries are NCA, NMC, and $LiMn_2O_4$. These are best used in electronics that require a stable voltage and long service life. The cathode and anode use aluminum and copper current collectors, and they move between the electrolyte during charging. To prevent contact while allowing ion transport, it utilizes a microporous polyolefin separator. This helps seal their pores at abnormal temperatures during any short. Li-Ion cells use additives, like flame retardants and other forming agents and stabilizers, to improve safety. Even though Li-Ion chemistry offers the best power-to-weight ratio, which is ideal for compact electronics, due to its high degradation at high temperatures when repeated operation is above 60 degrees C, which is an important consideration to have when operating in an environment where heat is often generated. This combination of heat can cause swelling in the battery. Overall, Li-ion cells are a

great choice for high energy in a small footprint and a strong charge retention for long cycle life.

3.6.1.2 Nickel-Metal Hydride (NiMH)

NiMH batteries have been around for decades, which is much longer than Li-ion batteries. These batteries provide a much higher energy density. They utilize a NiOOH cathode and hydrogen hydrogen-absorbing metal that supports ion movement during both charge and discharge. Compared to Li-ion, they have batteries that have lower energy density and a higher self-discharge rate. Manufacturing for these batteries is often half the price of Li-ion batteries. Although it has a reliable service life with about hundreds of recharge cycles under low temperatures, the battery degrades. For this project, although it could meet cost targets, a chemistry with higher energy density would ensure a longer runtime and be smaller and better for lighter wearables.

3.6.1.3 Nickel-Cadmium (NiCd)

NiCd has a very similar composition; they use a cathode and a cadmium anode, which is similar to NiMH batteries. NiCd cells are durable, and they have a long cycle life and a flat discharge. They are known for their high durability and flat discharge rate. This means that they can deliver voltage at a consistent rate under a certain load. However, the drawbacks are a lot higher due to manufacturing cost and severe environmental risk. Since NiCd contains toxic cadmium, this can cause a potential environmental and disposal issue. For this project, NiCd would not be beneficial because we don't seem to need high power consumption or need to be extremely durable. For our project, NiCd requires extra cost and weight, which defeats the purpose of creating a safe, lightweight, safe and inexpensive option. And since our power demand is low, the additional cost would not make sense for us to invest in

3.6.1.4 Alkaline (AA/AAA)

Alkaline cells are inexpensive and widely available primary batteries with long shelf life. This battery has a high internal resistance due to voltage sag under pulse currents. These are bulky and can leak over time. Unlike the previous batteries, it is unable to recharge for a daily training cycle, so it would constantly have to be replaced. In comparison to the thin single-cell LiPo battery, the LiPo

would provide higher energy density and better overall performance for our wearable device.

3.6.1.5 Lithium Polymer(LiPo)

Some typical applications of the LiPo can include wearables and medical devices, and portable DIY electronics. The LiPo charging circuit utilizes a lithium-ion polymer battery, a subtype of lithium-ion battery that employs a polymer instead of an organic solvent. The characteristics of the polymer are usually a gelled or solid polymer matrix that holds lithium salt. The first distinction between the two options of batteries is their density and weight. However, they have similar gravimetric energy density. LiPo typically offers better volumetric use of space due to the flat pouch. Some characteristics that are pertinent to our project are the packaging and form factor. The reason why we didn't select the Li-ion is due to its rigid cylindrical shape and battery size. The Lipo comes in a soft, laminated pouch, lighter and thinner, allowing for a custom shape. Our goal with our project is to create a lightweight sensor that can fit comfortably on fighters, so this LiPo is a feasible option.

3.6.1.6 Comparison

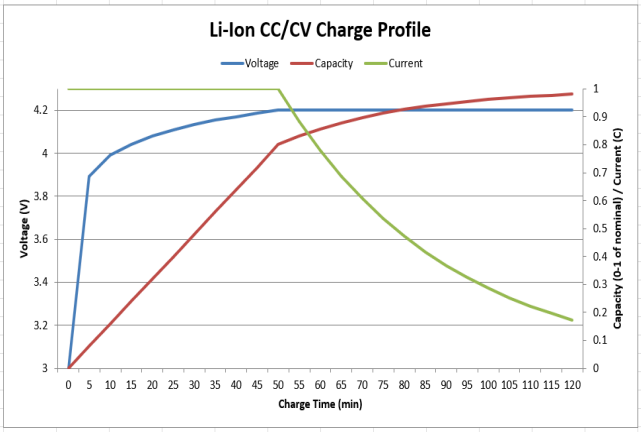
Battery Comparisons		
Battery Options	Positive	Negative
LiPO	● Very high energy density ● High discharge rate ● Lightweight	● Requires strict charging control ● Risk of swelling
Li-ION	● High energy density ● Long cycle life ● Low self-discharge	● Sensitive to overcharge ● High cost
NiMH	● Recyclable ● Good for high-drain devices	● Higher self-discharge than Li-ion ● Heavy ● Low energy density

Battery Comparisons		
NICD	● Robust ● Handles deep discharge ● High cycle life	● Low energy density ● Toxic cadmium ● Memory effects
AIKALINE	● Inexpensive ● Last a long time ● Very available	● Not rechargeable ● Heavier ● Must be replaced.

3.6.2 LiPo Charging and Voltage Regulation

LiPo batteries operate within three critical voltage ranges nominal, minimum, and maximum. The nominal is the LiPos normal charging operations. LiPo cells must stay between 3.0 V and 4.2 V per cell. Ensuring an understanding of the relationship of the charging design can ensure battery performance will be at an all-time high. A proper charging algorithm starts with pre-charge. This stage occurs when the LiPo is at a point where it's below about 3.0V, and the charger is at about 0.02 to .1 amp hour capacity of the cell. During this time, the input is used to raise the voltage into a normal operating range while keeping temperatures at a low. Once it enters back into 3.0V, the algorithm follows a constant current/constant voltage (CC/CV). The fixed current in this range is approximately between .5 and 1 amp per hour. And charges till it reaches 4.2V, where it will remain at its maximum. Once that range is completed, it enters the constant voltage phase when it is at its maximum, and tapers at this range to refrain from overshooting the voltage limit. The entire charge process ends when the current falls to a preset threshold or when a maximum time limit is met. The reason why it matters is because the CC/CV allows for longevity, maximum capacity for the cell, and constant speed., This method maximizes cycle life and minimizes common issues in cells, such as heat generation, swelling, and thermal runaway.

Time_min	Voltage_V	Current_C	Current_scaled	Capacity_fraction
0	3	0.5	1	0
5	3.893936	0.5	1	0.08
10	3.990309	0.5	1	0.16
15	4.043136	0.5	1	0.24
20	4.080618	0.5	1	0.32
25	4.109691	0.5	1	0.4
30	4.133445	0.5	1	0.48
35	4.153529	0.5	1	0.56
40	4.170927	0.5	1	0.64
45	4.186273	0.5	1	0.72
50	4.2	0.5	1	0.8
55	4.2	0.441248	0.882496903	0.830703655
60	4.2	0.3894	0.778800783	0.856693738
65	4.2	0.343645	0.687289279	0.878693868
70	4.2	0.303265	0.60653066	0.897316576
75	4.2	0.267631	0.535261429	0.913080358
80	4.2	0.236183	0.472366553	0.926424112
85	4.2	0.208431	0.41686202	0.937719355
90	4.2	0.18394	0.367879441	0.947280572
95	4.2	0.162326	0.324652467	0.955373968
100	4.2	0.143252	0.286504797	0.962224879
105	4.2	0.12642	0.252839596	0.968024051
110	4.2	0.111565	0.22313016	0.972932943
115	4.2	0.098456	0.196911675	0.977088231
120	4.2	0.086887	0.173773943	0.980605606



Cell Chemistry and behavior on batteries are strongly influenced by the external and internal temperatures. Temperatures exceeding the same operating limit can lead to complete degradation of the battery. At high temperatures, chemical reactions cause an increased rate of gas generation and swelling. Thermal feedback is required using an external thermistor to suspend charging outside optimal ranges of “−30 °C to 55 °C or −20 °C to 60 °C”. These considerations require proper PCB design to include thermal paths and copper areas to evenly dissipate heat that is generated by power components. Protection circuitry for LiPo are necessary electronic safeguards that keep each cell within safe parameters.

This circuitry is necessary because the tendency of these batteries to swell is critical to the charging algorithm. One of the main considerations that the protection circuit provides is preventing overvoltage. The circuit has the ability to turn off the charging once the battery reaches about 4.2V by comparing the cell voltage to precise reference levels at a millivolt precision. On the other hand, it also prevents undervoltage where the discharging cell only provides below the minimum voltage of 3.0. The protection monitors this and prevents it from reaching any lower, which will cause irreparable damage to the battery

Multi-cell balancing is essential because small differences in capacity can cause current leakage from cells that drift apart during state changes over time. If one battery reaches a maximum capacity before the others, the charge must be stopped immediately. There are two approaches that exist for multi-cell balancing. The first is passive balancing, which uses transistors to switch across each cell. When a cell reaches a maximum voltage, the circuit reduces the current and allows lower cells to catch up. Although it is simple and inexpensive, it utilizes too much energy.

Active balancing moves charge from the higher cells to the lower ones using converters. Although this improves efficiency, it is super complex and expensive. A robust LiPo design combines these electronic protections and redundant testing to detect faults. For our project, on the contrary, it would not be the best idea to use Multi-cell because not only does it add costs, but it also adds weight. This is an issue we are trying to avoid because we don't want anything too heavy, which could inevitably affect fighters' form. Not only that it could potentially cause higher voltage hazards. The best course of action for our device is to use one LIPO cell and pair it with a buck regulator. Not only will it meet our standards, it will be a lighter and simpler, and safer overall option.

Power path management and regulation enable a device to operate from an external source while simultaneously charging the LiPo battery. Some typical front-end elements for input sources include polarity diodes and surge protection. The power path manager looks at the available input voltage and uses that as a reference to limit current to meet the USB adapter ratings so that the source is never experiencing overvoltage or is overloaded.

3.6.3 Switching vs Linear

There are two versions of switching regulation: traditional linear chargers drop excess voltage across a pass transistor and regulate current. The benefits of this are low noise; however, it is highly inefficient, and since the voltage is higher, it inevitably wastes energy. Investing in a switching regulator is a better option because it resolves all these issues. We looked into a Buck-boost converter, which would convert excess voltage into added-on current, which can increase efficiency. The switching converter feeds an intermediate node that both powers the system and supplies the battery, and this keeps power dissipation low and maintains fast charging.

3.6.4 Electromagnetic Inference and Signal Integrity

Creating a battery charging PCB requires attention to detail regarding layout techniques. Some key considerations we examined are signal integrity, noise reduction, and electromagnetic interference (EMI). The EMI is a crucial factor

when it comes to sensitive analog sensors that utilize IMUs and magnetometers. In the sensor, the EMI will come from the chagrin circuits and the digital communications between lines. In our device, both the LiPO charger and the buck regulator are in the hundreds of kilohertz to megahertz range. These frequencies generate switching currents that radiate through traces; these emissions, coupled with the IMU, can cause bias errors and distortion. Our device needs accurate quaternion updates so that we can have precise strike identification, and any small discrepancies can be detrimental to the orientation accuracy.

We devised a plan to reduce any EMI near sensitive components, and the first step is to keep in mind the PCB layout. Components like the inductor, diode, and transistor must be as compact as possible. Having shorter traces with these loops can reduce the electromagnetic field to a smaller size. Placing our orientation IC on the opposite side of the charger and the inductor is a key factor in preventing magnetic coupling and preserving the integrity of lower signals.

Considering signal integrity is a major part of this charging design, poor signals lead to inconsistent charging outcomes. Or a reduction in the quality or longevity of the battery. Our goal is to keep sensitive components like feedback resistors and control ICs away from any section of the PCB that draws any high current and from any of the noisier areas. Placing these sensitive components near the ground plane can be the solution when designing this charging circuit.

A Pi-filter can be used to smooth the voltage rail and stop ripples that can disrupt the sensors. Utilizing decoupling capacitors can be a great option for stabilizing power delivery and reducing noise. Placing these next to each of our sensitive components, like the ESP32 IMU and magnetometer, will prevent voltage dips due to the capacitors' ability to act as a local energy reservoir and provide current during switching currents. Implementing a ferrite bead is another option to suppress EMI. These components can be used to block noise that travels between circuit sections. Electromagnetic compatibility is important for wearable devices because space is limited. Short traces and well-placed filters are necessary to maintain robust grounding and filtering to maintain clean voltage references.

3.6.5 Lipo Charging Circuit

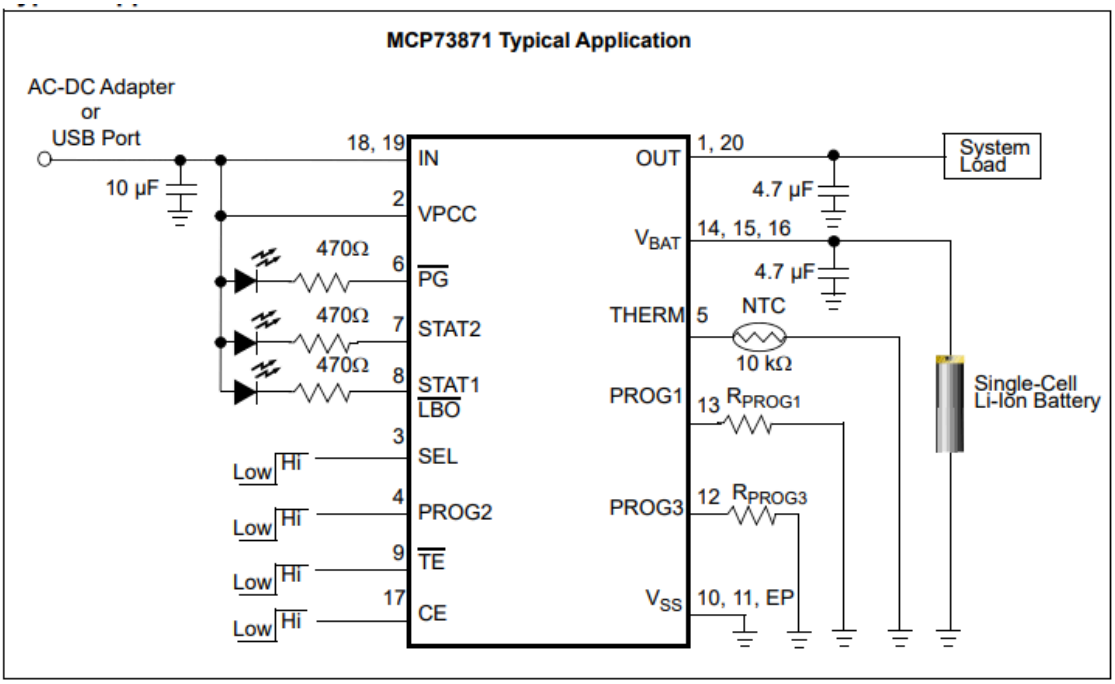
We considered why the charging design matters for the circuit. Inefficient and improper charging can correlate directly to potential safety concerns, cycle life, and usable energy. A charging design also determines how efficiently the system operates. A proper power path charger can supply both the ESP32 IMU and the magnetometer while also providing the necessary charge for the battery. When this is in place, we can ensure brownouts and current spikes are all prevented. We decided between three charging circuit options with the main goal of picking the one that is most efficient with our system. In short, its importance is more than just replenishing energy, but to inevitably stabilize the voltage rail and protect the Li-Po cell from damage.

The first option was the MCP73831, which is a Simple Linear charger. The reason why we considered this as an option is because we saw that it utilized minimal components. This is beneficial because this allows us to refrain from an extensive PCB design and makes routing and things like that more efficient. The cost and compactness of the circuit were another pro due to our need to minimize costs and reduce excess material on a fighter's arm. And provide reliable charging at up to 500 mA.. The reason it wasn't selected as our main option is because of its lack of power path management. While looking at the data sheet, we noticed that the system could not run while charging. Another point that came to mind is the need for USB only input. This is an issue because we require more flexibility since we are considering using USB-C.

Another option we had was the BQ24075, which is a switch-mode charger. The first positive we saw in this is that it is high-efficiency charging. This drew our attention because in our project, we needed to consider good thermal performance. After all, fighters will constantly draw heat while fighting, and so will the device. This charging circuit has an integrated power path so that it can run while charging. However, the issues that arose are that the IC is much more complex when it comes to design. Because of this, it comes at a much higher price than normal, with a larger footprint. Even though this is a good option, it is essentially not applicable and too much for a low-power wearable.

The final option for the circuit is the MCP73871, which is a power path linear charger. This combines USB/DC input and will automatically choose which one to

use without the need for a user to make changes. This comes with programmable charge and input limit currents with only simple resistors. In addition to that, it comes in a small package with a low amount of external parts and has excellent reliability for wearables. The reason why we selected this IC and not the other is that it provides the best balance of simplicity, safety, and flexibility. This makes it the ideal option for our strike sensor.



3.6.6 MCP73871 Battery management system

The BMS in the IC integrates the necessary protection and charging controls that are needed for a LIPo battery. It combines charging regulations, safety supervision, and power path control in one IC. This removes the need for a separate external BMS and can reduce the need for excess circuit design. It utilises a three-phase changing algorithm as mentioned beforehand: the preconditioning phase, constant current, and the constant voltage phase. This ensures that the charging is safe across all the battery states. Another characteristic that the IC has is that it intelligently decides whether the system is powered from the battery, USB, or the DC input. If there is a new external power source, the IC will make sure that the system supplies the load and charges the

battery. This happens automatically through the system power path management that makes sure that it is powered even when the cable is removed.

In addition to that, the BMS monitors the system safety parameters. The NTC input reads the battery's temperature via a thermistor, which limits instances where charging reaches any crazy numbers. These limits suspend operations when a system is in either overheating or freezing conditions, and thermal regulation that essentially reduces the charge current to regulate the temperature. Both are essential to prevent any issues with the user. In pins STA1 and STA2, they provide real-time updates on the charging conditions. It states whether the charging is in progress, complete, or a fault. In the IC, it uses a PG, which is an indicator that the source is inputting a valid supply voltage. Having both of these indicators will help the system diagnose any issues with charges and rectify the behaviours accordingly. Lastly, the IC has recharge functionality, which forces a restart when the battery voltage drops well below the safety threshold. This function is essential because it prevents long-term energy stability. The BMS inside our chosen IC provides reliable energy management and has multiple safeguards that can ensure that our devices are suitable for a fighter.

3.7 Communication Protocols

3.7.1 Serial Communication Protocols

3.7.1.1 I²C (Inter-Integrated Circuit

The I²C protocol is a two-wire serial communication interface designed for short-distance communication between multiple integrated circuits. It uses a bidirectional data line (SDA) and a clock line (SCL) to transmit data, allowing multiple slave devices to communicate with a single master over shared lines. I²C is easy to implement, requires minimal wiring, and is ideal for sensor communication in compact embedded systems such as wearables.
Common data rates include Standard Mode (100 kbps), Fast Mode (400 kbps), Fast Mode Plus (1 Mbps), and High-Speed Mode (3.4 Mbps), depending on bus capacitance and device capability.

3.7.1.2 SPI (Serial Peripheral Interface)

The SPI protocol is a full-duplex, four-wire communication interface designed for higher-speed data transfer. It uses separate lines for MOSI (Master Out Slave In), MISO (Master In Slave Out), SCLK (Clock), and SS (Slave Select). Unlike I²C, SPI allows simultaneous two-way data exchange and supports much higher throughput—often reaching tens of megabits per second.

SPI is advantageous for high-speed peripherals such as displays, memory chips, or IMUs that require continuous data streaming, though it consumes more GPIO pins and lacks native addressing capability.

3.7.2.3 Comparison and Selection

Communication Protocol Comparisons		
Feature	I²C (Inter-Integrated Circuit)	SPI (Serial Peripheral Interface)
Wires Required	2 (SCL, SDA)	4 (MOSI, MISO, SCLK, SS)
Speed Range	100 kbps – 5 Mbps	Up to 50 Mbps or higher
Communication	Half-duplex (shared bus)	Full-duplex (simultaneous TX/RX)
Addressing	Built-in (multiple slaves via unique addresses)	Requires individual chip-select lines
Complexity	Simple and cost-efficient	Slightly more complex, more pins required
Bus Length	Limited (<1 m typical)	Short to moderate (<1.5 m typical)
Noise Immunity	Moderate	High, differential clocking possible
Best For	Sensor networks, low-speed peripherals	High-speed data transfer (e.g., IMU, flash memory)

For this project, I²C was selected as the primary communication protocol for sensor interfacing (IMU and heart-rate module). The decision was driven by its simplicity, low pin count, and multi-device support, which reduce board complexity and routing constraints—key advantages for compact wearable hardware.
Although SPI provides superior speed, the performance of I²C in Fast Mode (400 kbps) is sufficient for real-time sensor data acquisition while maintaining low power consumption and minimal interconnects.
SPI remains available for future high-bandwidth peripherals such as displays or flash storage modules, but I²C offers the best balance of efficiency, compatibility, and ease of implementation for the current sensor suite.

3.7.2 Wireless Communication Protocols

Wireless communication enables devices to exchange data without physical connections, primarly using radio waves.These systems eliminate the need for wired infrastructure, providing greater flexibility and mobility. Modern wireless technologies support a wide range of applications from short-range peripheral links to high-throughput data streaming.

3.7.2.1 Bluetooth vs Wi-Fi

The link between our sensor modules and our host system is a crucial design consideration. The link should deliver the 9-axis sensor data with; low latency, reliably all with the lowest power consumption possible. Our chosen MCU presents us with two wireless communication options, bluetooth and Wi-Fi. We can compare these communication technologies in terms of throughput/latency, range/reliability and power/battery life.

3.7.2.1.1 Throughput and Latency

Example Packet: This calculation is based on raw sensor data, the worst case scenario, ideally our final design will include fused data in the form of quaternions, after applying a sensor fusion algorithm. In theory this will decrease the wireless payload, therefore decreasing throughput. However it is important to prove in the worst case scenario our chosen standard won't fail.

From our chosen sensor IC(ICM-20948) we find; The Gyroscope outputs a 16-bit signed number per axis (x,y,z) regardless of the full-scale range selection. With the digital low-pass filter (DLPF) enabled the Gyroscope has a selectable bandwidth of 5.7-197Hz. By Nyquist rule we must select an ODR >= 2*BW to

prevent aliasing. Since we know a human strike can generate much higher frequencies we must choose the highest selectable BW to capture the most data, therefore we can conclude, ODR>= 2*200Hz >= 400Hz. In keeping true to our examination of the worst case scenario, from the data ICM-20948 datasheet, the maximum ODR of the gyroscope is approximately 1kHz in Low-Noise Mode. We then can then estimate at 16 Bits per sample per axis and a 1kHz sample rate, the gyroscope produces

$$16 \text{ bits} \times 3 \text{ axes} \times 1000 \text{ samples/s} = 48{,}000 \text{ bits/s} \approx 6 \text{ kB/s}.$$

Similarly the accelerometer ,from our chosen sensor IC, outputs a 16-bit signed number per axis (x,y,z) regardless of the selected full-scale range. With the accelerometers digital low-pass filter (DLPF) enabled the output has a selectable BW between 5.7-246Hz. For simplicity we will choose a BW to match our Gyroscope data (200Hz). For the same reasons as above we will choose the maximum ODR our sensor has to offer in Low-Noise Mode (≈ 1kHz). Therefore we can estimate at 16 Bits per sample per axis and a 1kHz sample rate, the accelerometer produces

$$16 \text{ bits} \times 3 \text{ axes} \times 1000 \text{ samples/s} = 48{,}000 \text{ bits/s} \approx 6 \text{ kB/s}.$$

Given from the ICM-20948 datasheet the magnetometer has an output resolution of 16 bits with only one selectable FSR. Furthermore, the magnetometer only supports an ODR of 100Hz as magnetic fields change much slower. Therefore, we can estimate at 16 bits per sample per axis and a 100Hz sample rate, the magnetometer produces 16

$$16 \text{ bits} \times 3 \text{ axes} \times 100 \text{ samples/s} = 4{,}800 \text{ bits/s} \approx 600 \text{ B/s}.$$

Adding metadata like timestamps, sequence numbers and synchronizations flags is a standard practice in sensor networks, especially when streaming high frequency data. We include this data because while transmitting IMU samples wirelessly packets can:

- Arrive out of order or occasionally completely be dropped
- Be generated by multiple sensors (e.q the three sensor modules shoulder, elbow and wrist) that need to be time alignment for machine learning
- Merged with other systems, such as video frames

Metadata will ensure that packets can be identified, ordered, and aligned in time. Below is a table breaking down common metadata fields that pertain to our project:

Metadata Breakdown			
Field	Purpose	Typical Size	Example Implementation
Timestamp	Marks the time the sample was captured (relative or absolute)	4 bytes (uint32)	milliseconds since boot or epoch time
Sequence number	Lets receiver detect dropped or out-of-order packets	2 bytes (uint16)	increments per sample or per packet
Sensor ID / Node ID	Identifies which IMU the packet came from (for multi-sensor setups)	1 byte	e.g., LeftHand = 0x01, RightHand = 0x02
Status / Flags	Encodes mode bits: error states, calibration flags, DLPF setting, etc.	1 byte	bitfield (e.g., bit0=gyroCal, bit1=magActive)

From the above we can safely estimate our metadata will result in no more than an additional 10 bytes per sample per sensor. Therefor our final packet size calculations are as listed bellow

Packet Calculations				
Souce	Payload (Bytes)	ODR(Hz)	Data rate B/s	Data rate kB/s
Gyroscope	6 + 10 (meta)	1000	16,000	16
Accelerometer	6 + 10(meta)	1000	16,000	16
Magnetometer	6 + 10(meta)	100	1,600	1.6

Packet Calculations				
Total	–	–	33,600	33.6

Bluetooth Low Energy (BLE) and Wi-Fi both use the 2.4GHz radio band, but they each are designed with a different goal in mind. Wi-Fi is built for high-speed data transfer, such as video streaming or file downloads, while BLE is optimized for short-range, low-power communications between small devices.

In Bluetooth 5, data is transmitted over a physical layer (PHY), this defines how fast bits are sent over the air. Two common options exist:

- **1 M PHY:** transfers data at 1 megabit per second (Mb/s)
- **2 M PHY:** doubles that speed to 2 Mb/s

These are raw radio bit rates, thus after accounting for protocol overhead, BLE delivers roughly 75-100 kilobytes per second (kB/s) or real application throughput.

Wi-Fi on the other hand, can move data at a much faster speed (100x or even 1000x) than that of BLE. Our worst-case IMU payload (≈ 33.6 kB/s) easily fits within the real-world capacity of BLE, leaving enough headroom for retransmission or additional metadata. Wi-Fi greatly exceeds our needed bandwidth with potential downsides in reliability and power/battery life see more below.

3.7.2.1.2 Range and Reliability

Bluetooth Low Energy (BLE) has been designed for short-ranges, operating reliably over distances of 10-100 meters indoors, depending on obstacles such as walls, human body and interference. One the reliability point, BLE, benefits from simple radio and protocol stacks due to fewer network layers and less congestion, in small-scale point-to-point links. This helps maintain timely and ordered delivery of sensor data packets.
On the other hand Wi-Fi offers significantly longer ranges and higher device density. In favorable conditions Wi-Fi can cover 10s to 100s of meters, supporting a plethora of devices, switches and routers. However the higher complexity of Wi-Fi resulting from a richer protocol, background traffic and connections with multiple devices leads to variable latency.

For our use case, a short-range sensor-to-hub application, the extra range of Wi-Fi is unnecessary while the added latency creates a less than desirable communication protocol for our real-time use case.

3.7.2.1.3 Power and Battery Life

From the start BLE was designed for very low power consumption making it ideal for battery-powered devices. A BLE radio can remain idle for long periods of time while “waking” very briefly to transmit small packets; this keeps the average current draw very low. By contrast, Wi-Fi radios are designed for higher sustained throughput and often require the radio and network stack to stay active continuously which significantly increases power consumption. For example, one study by the University of Groningen[10] concluded that a phone using BLE communication lasted ~16hrs 38mins versus ~14hrs 46mins using WiFi. This equates to roughly a 30% more battery efficiency for BLE. In practice for our use case this means for the same size battery we can expect a longer battery life using BLE vs Wi-Fi.

We can therefore conclude BLE is best fit for our use case.

3.8 Machine Learning and Neural Networks

3.8.1 Brief History

With the term artificial intelligence proliferating throughout both corporate spaces and pop culture, it’s easy to believe that it is a novel revolution that jumpstarted a couple years ago. This couldn’t be farther from the case. Artificial intelligence has its roots in the mid-20th century, growing out of an early work in statistics and cognitive science. It was around this time that the immediate subset, machine learning, was being defined as well. The term was popularized by Arthur Samuel in the 1950’s who developed a program that could “learn” to play checkers as it built up “experience”. At about the same time, Alan Turing had already begun his investigation into whether machines could learn. This investigation-built momentum into the 1960s which lasted until about the 1980s. During this time, machine learning evolved through rule-based systems and symbolic approaches. Unfortunately, a “dark age” of sorts came between 1985 to about the late 1990’s. This was brought about because of limited computational power and small datasets. Despite these obstacles, groundwork that is still being used today was laid out. This includes decision trees, k-nearest neighbors, and linear regression.

As the new millennium began, there was a shift towards statistical learning

which emphasized probability, optimization and rigorous mathematical modeling. It was around this time that algorithms such as support vector machines and ensemble methods were introduced. With better ways to store data came an explosion of digital data sets and the birth of datasets being built that we wouldn't see until much later. These datasets provided fertile ground for training models. However, while the methods used thus far paired with these datasets performed well, they still struggled with complex tasks such as image recognition and language processing. The commonality between these complex tasks was their highly non-linear and multi-layered nature.

As was done before when a challenge arose, researchers rose to the challenge. In this case instead of taking advantage of advancements, they went back to the fundamentals in the form of neural networks. Neural networks were concepts inspired by models of the human brain in the 1940's. Early neural networks like the perception showed promise but were limited by theory and hardware constraints. By the 2010s, advances in parallel computing (particularly GPUs), larger datasets, and improved training techniques such as backpropagation and deep architectures revived the field. This "deep learning revolution" transformed neural networks into the driving force behind today's breakthroughs in computer vision, natural language processing, and reinforcement learning. Neural networks are now at the center of machine learning, representing the convergence of decades of progress in both algorithms and computational power.

3.8.2 Neural Networks and How They Work

Neural networks are a class of machine learning models designed to recognize complex patterns in data. They are inspired by the structure of the human brain which contains interconnected neurons that when activated, transmit signals to process information. Artificial neurons serve as the fundamental building block to artificial neural networks. These neurons work by receiving input values, apply a weighted sum, and pass the results through an activation function. The next level would be organizing these neurons into layers. There are different types of layers within networks that help accomplish different goals. An input layer receives data, hidden layers transform that data through successive computations, and an output layer generates predictions. Adjusting the weights that connect the neurons aids in the network learning intricate relationships that are often difficult to capture with traditional statistical methods.

The actual process of learning can be described through forward and backward propagation. In a forward pass, data moves from the input layer into the hidden layers where it is processed and then a prediction is given through an output layer. The prediction is then assessed by a loss function which quantifies the error relative to the correct output. This error is then propagated backwards through the network where it is optimized with algorithms such as gradient descent to adjust the weights. As this process persists across iterations, the network improves its internal representation of the data, enabling it to generalize to new examples. *

Neural networks show their strength in dealing with temporal and spatial data. Images or output from sensors often contain local patterns and hierarchical structures that reveal context behind the numbers. In the pursuit of dealing with this kind of data, Convolutional neural networks were born. CNN's use convolutional layers that apply filters upon the input with the purpose of extracting features. When stacked, features from edges, textures, or shapes to abstract representations can be learned through enough training. The ability for abstract understanding makes it suitable for tasks such as image recognition, medical imaging , and even modeling physical systems.

Understanding temporal data requires heavy emphasis on sequence and context. Some of the data that requires this includes language, time series signals, and video frames. Recurrent neural networks (RNNs) were made to process and understand this data by introducing a sort of 'memory' through feedback connections. These connections allow information to persist across time steps. This approach makes it possible to learn dependencies across a given sequence. The one downside is that traditionally, RNN's would begin to struggle the longer the range of dependencies got. This problem gave birth to more advanced architecture like Long-Short-Term Memory networks and Gated Recurrent Units. These architectures use gating mechanisms to control the follow of information and retain relevant information over long time sequences. Recently, through methodologies applied in the paradigm shifting "Attention is All You Need" paper, Transformer architectures have surpassed RNN's in many applications. This has been accomplished by leveraging self-attention mechanisms to capture temporal dependencies with none of the bottleneck induced by sequences.

Neural networks provide a flexible and robust framework for modeling data with spatial and temporal structure through the use of the architectures working in unison. Video analysis can employ the following approach: CNNs to extract spatial features from individual frames and transformers to model how those

features evolve across time. As researchers explored the domains that neural networks can be applied to, ideas of image compression went from theory to reality. This evolved to become known as computer vision which is focused on getting computers to understand what they see on a deeper level. This evolution and subsequent niche serve as a cornerstone for our project as it serves as one half of our source of data.

3.8.3 Computer Vision and Pose Estimation

Computer Vision is one of the most impactful errors where neural networks have demonstrated their potential. At its core, computer vision seeks to give machines the ability to mimic the way humans perceive visual data. The combination of large image datasets, powerful CNNs, and high performance computing has made it possible to get pretty close to human performance. Unlike its preceding traditional algorithms, modern computer vision pipelines are built to automatically learn hierarchical representations of images by extracting low level features before progressing to higher level patterns. These low level features include edges and textures while high level patterns reveal meaning about the visual data. This shift has made computer vision a cornerstone of engineering applications, ranging from autonomous vehicles to medical imaging diagnostics.

With such a general definition, computer vision gets applied in many facets however the one being focused on in this project is pose estimation. Pose estimation refers to the task of detecting and tracking the positions of joints (known as key points) or landmarks from images or video sequences. It begins with the expansion of the principles found in spatial feature extraction by using it to localize joints in two or three dimensions. It does this while also integrating temporal modeling to track movement across frames. The first few approaches relied on template matching or handcrafted key point descriptors but as deep learning has evolved, we are now able to produce highly accurate, real time pose estimation. The backbone model of our project, Alphapose, is an amazing example of this. Alphapose employs multi stage CNN’s that predict joint heatmaps and refine them through iterative processing. More recent methods leverage transformer based models which improve the ability to capture long-range dependencies between body parts.

Pose estimation has significant implications in engineering and beyond. In biomechanics, it enables detailed analysis of human motion without the need for intrusive markers or sensors. In sports engineering, it supports performance

analysis by quantifying technique, posture, and movement efficiency. In robotics and human-computer interaction, pose estimation allows machines to interpret gestures, enabling more natural and adaptive interfaces. Furthermore, healthcare applications such as physical therapy monitoring and fall detection are increasingly adopting pose estimation technologies to enhance safety and improve patient outcomes. These examples highlight how advances in neural networks and computer vision not only enable pose estimation but also open new avenues for applied research in fields where temporal and spatial understanding of human movement is critical.

3.8.4 Alphapose : What it is and How it Fits

The requirements for StrikeSense necessitate a robust pose estimation model to capture datapoints. The following were listed to be the most important: ability to stream data in real time, ability to maintain contexts for all datapoints across long sequences, and the ability to identify multiple entities as well maintain accurate separation of entities. After reading through multiple papers and multiple approaches, Alphapose stood out head and shoulders above the rest because of it's accuracy and having all the elements that were needed.

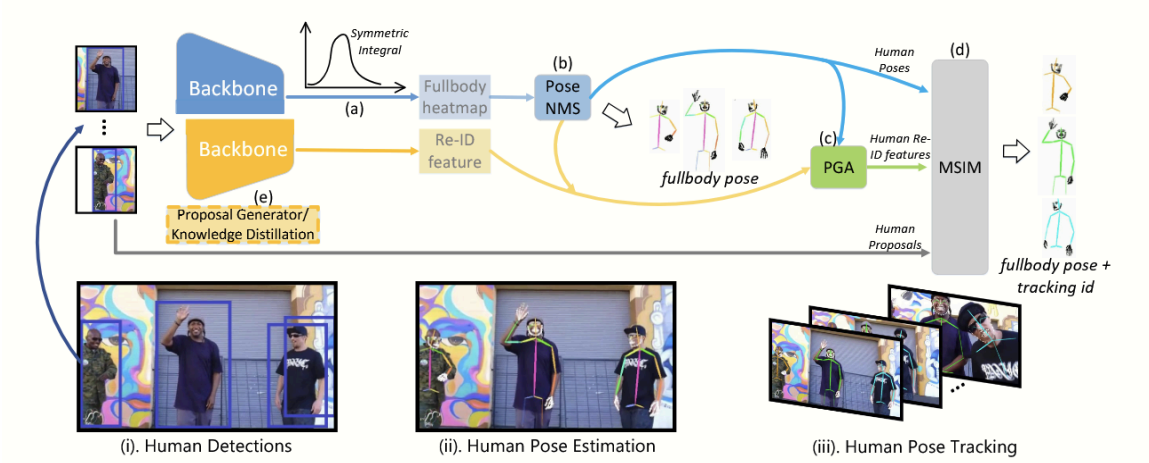
Alphapose is a pipeline that provides real time, whole body pose estimation, and person tracking. It was introduced in the paper [AlphaPose: Whole-Body Regional Multi-PersonPose Estimation and Tracking in Real-Time]**.** At its core, AlphaPose uses a top-down framework that first detects human bounding boxes and then estimates detailed keypoints within each one. What makes it stand out is its ability to handle entire body structures, including the face, hands, feet, and torso, while still running in real time. This level of precision across all regions of the body was previously difficult to achieve without sacrificing speed.

One of the key innovations that makes this possible is its use of *Symmetric Integral Keypoint Regression* (SIKR). In earlier pose estimation systems, keypoints were represented as points on a heatmap. Because heatmaps are discrete, this approach often introduces small but important localization errors known as quantization errors. These errors were especially noticeable when working with fine details, such as finger or facial movements. AlphaPose overcomes this by using SIKR, which allows the model to estimate joint positions continuously rather than snapping them to the nearest pixel. This leads to smoother and more precise keypoint predictions, especially in regions that require fine-level accuracy.

Another major contribution of AlphaPose is its *Parametric Pose Non-Maximum Suppression* (P-NMS). When detecting multiple people in the same frame, models often produce overlapping or duplicate poses. P-NMS solves this by learning how to compare and remove redundant detections based on how similar two poses are, rather than relying only on bounding box overlap. This results in cleaner, more reliable outputs even in crowded or fast-moving scenes.

AlphaPose also incorporates *Pose-Aware Identity Embedding*, which allows it to track people consistently across video frames. This means it can recognize and maintain the identity of individuals as they move, turn, or temporarily leave the frame. This feature is particularly useful in applications such as sports analytics, motion capture, and surveillance, where accurate tracking over time is essential.

To further improve generalization, AlphaPose uses a combination of *multi-domain knowledge distillation* and a *Part-Guided Proposal Generator (PGPG)*. These components help the model learn from diverse datasets that focus on different body parts, allowing it to adapt to a wide variety of poses and conditions. The result is a system that performs well even in real-world environments where lighting, camera angles, and body positions vary greatly.



A Breakdown of Alphapose

Overall, AlphaPose represents a balance between speed and precision that few systems have achieved. It not only detects and tracks human motion accurately

but also does so at a rate suitable for live applications. For StrikeSense, this translates into dependable, real-time body tracking that can interpret fast and complex movements without losing accuracy. In doing so, AlphaPose provides a strong foundation for capturing motion data that is both high in quality and consistent over time, supporting deeper analysis of performance, technique, and biomechanics.

Chapter 4 - Standards and Design Constraints

4.1 Battery Standards

There are various domains when it comes to battery standards. Some important considerations for some are cell and pack design, safety, transport environment, and regulatory compliance. The purpose behind these standards is to help document and justify design choices and prepare for robust real-world applications

4.1.1 IEC 62133-2 – Rechargeable Lithium System Safety

One of the first battery standards we came across was the IEC 62133-2, which addresses the LiPo chemistry. This is issued by the International Electrotechnical Commission (IEC), which applies to rechargeable cells and batteries that are for portable devices. It lays out the minimum design requirements and performance and test criteria to make sure that the battery is operating in both normal and abusive conditions.

The reason this standard directly impacts our project is that the sensor system utilizes a 3.7 V single-cell LiPo battery. Making sure that we comply with IEC 62133-2 will allow all the selected cells and charging circuitry to meet the environmental, mechanical, and electrical safety requirements for consumers.

Some of the key requirements and tests include an external short circuit, a free-fall or drop test. These tests are for the safety of the battery when reasonably foreseeable misuse occurs. These are important because they check to see how a battery handles a dangerous condition and help to look at how it will respond. For the pack design, we must ensure that the cell has certification for both the form factor and the usage environment.

Some of the key provisions IEC 62133-2 has are that it ensures multiple critical tests to make sure the LiPo circuit is reliable with projections for overcharge, short circuit, forced discharge, and mechanical shock. These test shows that the battery does not ignite or explode when under extreme pressure or stress. In our design, we are utilizing the MCP73871 charge controller, which has a battery management system that regulates charge current and terminates when the voltage reaches its maximum output. The charging circuit utilizes a MOSFET and

an NTC thermistor to provide a cut-off for any excess thermal activity. The design integration for our system must include a tested Li-po cell and have an enclosure that isolates the pouch from the mechanical compressor, and be able to calibrate

4.1.2 UL 2054 Household and Commercial Battery Pack

The Underwriters Laboratories standard defines safety requirements for battery packs. The difference between the IEC 62133 and the UL 2054 is that UL governs the complete battery assembly. It makes sure that the packs went through temperature cycling, dielectric strength, and drop testing. For our strike sensor, the polymer pouch structure will be reinforced by a 3D printed housing that stops tearing under load. The standard also requires labeling and marking conventions, such as nominal voltage and capacity. By ensuring all these are met, the project will refrain from all electrical hazards from extended use. Some things it would be able to detect would be polarity indication and improper charging, and disposal

4.1.3 Transport and Handling Requirements

This is the United Nations manual of tests and criteria. The scope of this standard is to ensure that the batteries are safe for transportation under typical transport conditions. The relevance of Li-po batteries is that these packs will be transported under various altitudes internationally, and because they tend to catch on fire and potentially even explode, we need to ensure that they meet minimum standards. For instance, it must undergo Altitude thermal and vibration testing, which will verify battery venting, place prototype temperature rangers, and potential swelling. For our project, this allowed us to keep in mind to order a Li-po pack that has an UN 38.3 test summary to ensure these safeties

UL 1642 is the U.S Cell-Level Safety Standard, and it gives the necessary electrical and mechanical safety tests for individual lithium cells. This standard is widely accepted and often required for consumer electronics. Some of the needs that the Lipo battery must have to consider passing the standard are to first have a casing that is rigid enough to resist any abusive behavior that could happen without starting fires or things of that nature. Another criterion is that it must be hard enough to prevent flexing

4.2 Battery Design Constraints

4.2.1 Electrical and Power Constraints

For the subsystems to be efficient, they must incorporate a balance between energy density, safety, and weight. Because the project requires a constant sensor out and reading the voltage must be at a stable 3.7 V power output with minimal ripples. This increases the need for over-discharge protection, which will

prevent deep cycling. The need to limit current draw to manage load monitoring is also a requirement so that we can comply with thermal limits.

4.2.2 Thermal Constraints

The need for thermal constraints is imperative to our LiPO system. During instances of charge and discharge, the resistance will generate a large amount of heat, which can cause both swelling and runaway. For our project, since fighters are constantly in motion, we know that even more heat will be generated on the device. This requires us to keep our sensor below 45 degrees Celsius to refrain from those issues. We will incorporate a temperature feedback loop using the NTC to do so.

4.2.3 Mechanical Constraints

When looking at the mechanical constraints, we noticed that we would need to have a design that can withstand puncture, bending, and vibration. We will use a 3D casing to isolate the battery from the PCB.

4.2.4 Sustainability Constraints

Overall, our goal is to ensure that fighters can be comfortable while training with these devices. We want to refrain from having them constantly charge their device and interrupt their focus sessions. We want or focus on long battery life span and recyclable materials, and responsible energy use. We must limit charge voltage to approximately 4.1 V, which will extend cycle life to about 500 cycles. Selecting the correct materials for the battery enclosure can help to increase sustainability as well.

4.3 Bluetooth

4.3.1 Introduction

4.3.2 BLE Protocol Stack Overview

A typical BLE communication consists of two devices: The **peripheral**, a small low powered device e.g the sensor module in our design and a **central**, the controller or host. First a connection must be made. A BLE peripheral advertised its presence by broadcasting small packets on a specific radio frequency. The host scans these advertisement channels and chooses a device to connect to,

establishing a data link. The devices then communicate by propagating data through the BLE protocol layers seen below.

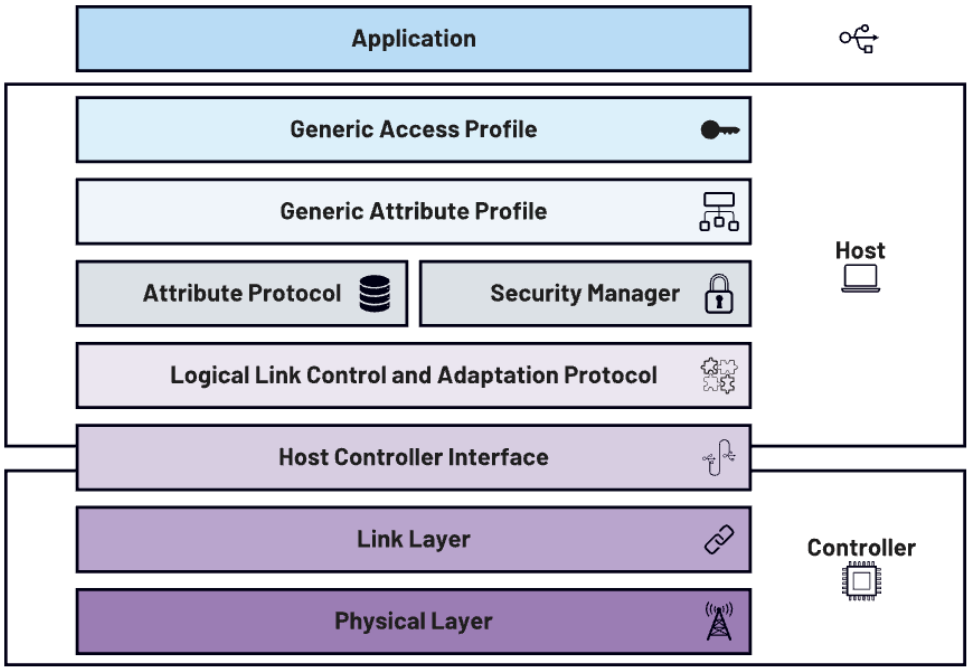


Figure 1. A BLE stack architecture. Adapted from Analog Devices [7]

The BLE stack above describes the complete communication framework between two BLE-enabled devices. The BLE stack is organized into three main layers (each with sub layers), those layers are **Application**, **Host** and **Controller** layers.

Application Layer

This layer contains the user applications logic for our project. This is where the punch-tracking algorithms, sensor data acquisition and transmission logic reside. The responsibility of this layer defines what the BLE device does. The application communicates with the host layer through Generic Attribute Profile (GATT). GATT defines how the data is structured and shared through designing services and characteristics. Simply this layer customizes the BLE stack to meet specific device/application specific requirements.

Host Layer

This layer provides high-level BLE protocol functionality and contains the remaining upper layers of the BLE stack.

Host Layer Breakdown	
Sub-Layer	Description
Generic Access Profile (GAP)	Controls how devices discover each other, advertise and from connections. Defines roles (peripheral, central, broadcaster, observer)
Generic Attribute Profile (GATT)	Defines how data is structured and accessed
Attribute Protocol (ATT)	Defines rules for accessing data/attributes. Handles reading and writing of attributes between client and server
Security Manager (SMP)	Handles device authentication and encryption usings keys and passcodes
Logical Link Control and Adaptation Protocol (L2CAP)	Segments, resembles, and multiplexes data packers from the upper layers. Organizes data between multiple data channels

Host Controller Interface (HCI)
We mention this sub-layer separately as it acts as a bridge between Host (software) and Controller (hardware) layers. This layer allows communication with an antenna over serial interfaces such as URAT, SPI, or USB. The responsibility here is to keep the BLE logic independently from the physical radio hardware.

Controller Layer
The controller contains two sub layers, link and physical layer. This layer functionality serves as a physical bluetooth link and handles all real-time radio operations. This hardware-driven layer ensures timing accuracy and low power consumption.

Controller Layer	
Sub-Layer	Description
Link Layer (LL)	Manages packet scheduling, frequency hopping, advertising, scanning and maintaining connections.
Physical Layer (PHY)	Converts digital data into 2.4GHz RF signals using Gaussian Frequency Shift Keying (GFSK). Defines the data rate (1 Mbps, 2 Mbps or a coded long range PHY)

4.4 Wearables

4.4.1 Introduction

StrikeSense is meant to have direct contact with the body, pull heart data optically, and collect motion data during training sessions. Because of the intensive environment as well as the duration of time it will be in that environment, it is important to implement a plan of safety to adhere as close as possible to preexisting standards concerning wearable technology. The main standards being aligned are ISO 10993-1, ISO 10993-5, and ISO 10993-10. This is due to the constant skin contact and risk for exposure to high impact. These standards are widely used with medical devices to make sure that any materials that interact with the body do not irritate skin, poison the wearer, and do not trigger allergic reactions.

4.4.2 ISO-10993: What Is It?

ISO 10993 is a family of standards that grew out of a simple problem in the medical device world. People were putting plastics, adhesives, paints, coatings,

metals, rubbers, and electronics directly on or even inside the body, and there was no universal playbook for answering a basic question: is this safe for human tissue over the time we plan to use it. ISO 10993 was created to be that playbook. It gives a structured way to evaluate how a device and its materials interact with the body, and it does that through a risk managed process instead of guesswork. The core document, ISO 10993 1, lays out the idea that safety is not just one lab test. You first identify what touches the body and for how long, you gather all existing data on those materials, and then you only run the biological tests that make sense for that real contact scenario. The goal is to protect people from toxic reactions, irritation, allergic response, or other harm while also avoiding pointless testing, especially heavy animal testing, if in vitro data and chemical data can answer the same safety question. ISO 10993 also recognizes that a medical device can be anything from a simple single material patch to a complicated multi material instrument, and that one material can be harmless in one situation and risky in another. Because of that, the standard groups devices by how they contact the body and for how long, then recommend the kinds of biological data that are relevant for each group, instead of pretending that one checklist fits everyone.

4.4.3 ISO-10993-1

ISO 10993-1 provides instructions on how to evaluate whether a product meant to maintain contact with the body is biologically safe. Rather than just throwing the device on a willing participant and looking at what happens, you should make a biological evaluation as part of a larger risk management process. In order to adhere to this while still working within the limited scope of our project, we will create a short document that document what the device is made of, how long it will contact skin, what kinds of fluids it might be exposed to, what chemicals might leach out, and what known risks come with similar materials. At this point in time, we are still working with an evolving design. Therefore, we will keep this standard in mind when doing so and document once we settle on the exact materials.

Another aspect that is stressed in ISO 10933-1 is that once assembled, different materials and coatings have the potential to behave differently than when isolated. The fabric selected for a strap may be fine however once it comes into contact with sweat, body heat, adhesives and coloring, that strap can create a rash. The standard requires us to consider the final picture of the project, meaning what gets put together during assembly, the residues that may still be there, and what variables come into play during normal use cases. That includes

additives like dyes and stabilizers, processing leftovers like solvents, and any breakdown products that could show up after long wear.

For us to operate under ISO 10993-1 moving forward, we are expected to do three big things

1. Characterize materials

2. Categorize the type of body contact

3. Run tests or refer to tests that are relevant to the material and type of body contact

We will undergo these actions within the scope of our project to adhere to the standard moving forward.

4.4.4 ISO-10993-5

ISO 10993-5 goes into detail into checking if a material kills or is at risk of damaging mammalian cells. If the material is toxic to cells in a dish, then it is strongly advised that you don't have it pressed onto the human skin for hours on end.

There are three types of test conduct. The first one is the extract test. You soak the device material in a liquid that mimics body fluid, then you expose cultured cells to that liquid and see if the cells get damaged or die. The second one is a direct contact test. In this test, you place the material itself on top of a cell layer and watch what happens to the cells under and around it. The last kind of test that can be used is the indirect contact test. This is where you physically separate the material from the cells with a barrier like agar, so that only diffusing chemicals reach the cells.

For StrikeSense, Cytoxicity matters most for the parts that will be tightly pressed against skin and risk trapping sweat. Some components that risk this include the optical sensor window and housing, the interior of the strap surface and any adhesive used to connect nodes from our design to the straps. For our use case, we will take a use case representative sample of those materials. This is important as ISO 109993-5 expects us to consider the final device state, including anything that could be on the surface after its final design. If any of those tests show meaningful cytotoxicity, we will treat that as a design blocker. We will either swap materials, change coatings, add barrier layers, or modify cleaning and finishing steps until the contact surfaces do not damage cells.

4.4.4 ISO-10993-10

A product can pass through all the test described in 10993-5 yet still cause a serious reaction on a user’s skin. This is why ISO 10993-10 is focused on skin sensitization. sensitization is different from simple irritation. Irritation is like temporary redness because something rubbed your skin too hard. Sensitization is an immune reaction. The immune system learns to see a chemical as an allergen and then every later exposure triggers a stronger reaction. This can show up as itching, redness, swelling, or blistering. The standard is written to catch that risk before people wear the device long term.

The standard emphasizes that we should understand chemical composition up front and review any existing data already available on those chemicals before testing. Anything that sits against sweat soaked skin during high friction training is a realistic sensitization risk, especially if it contains colorants, softeners, antimicrobial treatments, or nickel bearing fasteners. The plan is to review the full bill of materials for known sensitizers before we ever lock the design. That includes elastomers, TPU blends, or silicone rubbers.

we will treat any sign of sensitization as a serious hazard in our risk file. Unlike a one-time irritation that goes away when you take the device off, sensitization is a gift that keeps on giving. Once a user is sensitized, even a tiny later exposure can trigger the reaction again. Because of that, any StrikeSense material that shows sensitization potential will be redesigned finalizing our project design

Chapter 5 - Application of Chat GPT

5.1 Introduction

During the duration of this project, it is up to us to use all tools at our disposal in meaningful and enriching ways. One of the most indispensable tools at our disposal has been GenAI. In a world that is changing at such a rapid rate, AI serves as another variable that pushes us towards advancements. Despite this, it is not without it's pitfalls. From hallucinations to creating knowledge gaps, it is up to the user to navigate through these effectively. Each interaction we've had has had purpose and supplemented our projects development as well as our personal growth.

5.2 Short History

In 2015, OpenAI was originally founded under the guise of being a research company to develop safe and beneficial AI. The architecture at the core of it was the GPT, otherwise known as the Generative Pre-Trained Transformer language model. GPT was introduced to the general public in 2018. When first introduced, it served as a proof of concept that transformers could learn language patterns. As time progressed, there was a consistent and steady growth with capabilities which led to the introduction of ChatGPT in 2022. This represents the incarnation of GPT that has led to the explosion of AI use cases in all domains. From 'revolutionary' services shipped with a shiny coat of wrapper obstructing hundreds requests to ChatGPT to academia reaching an impasse as to what to do about rampant cheating, ChatGPT had arrived and was here to stay. Currently, we are on the 5th incarnation of ChatGPT which boasts features such as broader reasoning , coding, data analysis, and third party tool use. All of these features were used when appropriate for our project.

5.3 LLM Comparison

While utilizing ChatGPT as a source for information, it's imperative to consider that it is not perfect, and it is important to compare its output to that of other Large Language Models. In this section, we will do a comparison between ChatGPT responses and Gemini responses to further evaluate any discrepancies between the two Artificial Intelligences. To start with, we will ask a prompt and review responses

-**Prompt:** *Describe how a magnetometer can be used in a wearable strike sensor to maintain heading accuracy during rapid arm movement. Include how magnetic*

field data contributes to quaternion updates. Provide specific Information as well as a general overview.

ChatGPT Response:

“A magnetometer in a wearable strike sensor helps maintain accurate heading by providing a fixed reference to the Earth’s magnetic field. When the sensor measures rapid arm movements, the gyroscope and accelerometer alone can track orientation, but over time, they accumulate small integration errors that cause yaw drift. The magnetometer compensates for this by continuously measuring the local magnetic field along three axes (Bx, By, Bz) and using that information to correct the system’s directional reference. This ensures that the sensor maintains a consistent heading relative to magnetic north, even after multiple rotations or quick strikes.In a fused sensor system, magnetic field data is combined with accelerometer and gyroscope outputs using a sensor fusion algorithm such as a Kalman, Madgwick, or Mahony filter. The algorithm converts all incoming sensor data into a quaternion, a mathematical model that represents 3D orientation. The magnetometer specifically influences the yaw component of the quaternion by providing a reference to the horizontal plane. When gyroscope drift occurs, the fusion algorithm compares the predicted heading to the measured magnetic vector and adjusts the quaternion accordingly. This correction keeps orientation tracking stable and reliable, ensuring that every recorded strike reflects accurate direction and motion.”

The information ChatGPT provides is not entirely accurate. I noticed that it said things like “predicted heading,” which is an inaccurate term. Filters predict the magnetic vector in the sensor frame heading refers more to the corrected value. And another issue was that quaternions don't have an isolated yaw; there is no way to directly adjust it.

This is an accurate example of an LLM having issues with providing accurate information to users. Models like these have a hard time verifying responses, which causes them to have objectively false information. Although it did get a bit of a few responses wrong, it still gave the overall concept of Magnetometers in a well-put-together manner. This is just a prime example that you can't just rely on LLMs without doing your own research, or you can potentially be at risk of getting the wrong answers.

Gemini Response:
“A magnetometer is crucial in a wearable strike sensor to provide an absolute heading reference, which counters the rapid accumulation of error, known as gyroscope drift, during dynamic arm movement. In a general sense, the magnetometer acts as a digital compass, constantly measuring the local magnetic field vector relative to the sensor's current orientation. During a rapid strike (like a punch), the sensor fusion algorithm temporarily relies heavily on the gyroscope's instantaneous angular velocity data to track the high-rate motion.

However, as soon as the movement slows or stops, the algorithm uses the stable magnetic field data (in conjunction with gravity data from the accelerometer) to identify which way is magnetic North. This external, non-drifting reference is then used to correct the accumulated yaw (heading) error from the gyroscope, ensuring the sensor's reported orientation remains aligned with the real world before the next strike. Specifically, the magnetometer's data contributes directly to the quaternion update by generating a correction signal. The sensor fusion filter calculates the difference between the magnetometer's measurement and a predicted magnetic vector, which is obtained by rotating a known World Reference Vector using the current orientation quaternion. The angular difference between the measured and predicted vectors—often calculated via a cross-product—creates an error vector. This error vector is then used as a feedback term to slightly adjust the rotational rate used in the quaternion update equation. By applying this correction, the filter slowly pulls the estimated heading back toward magnetic North, ensuring it accurately represents the strike sensor's orientation, even after high-velocity movement has corrupted the gyroscope data. Excerpt adapted from ChatGPT (OpenAI, 2025) [1] *”*

The explanation Gemini provided was mostly accurate. It provided a near-perfect engineering explanation. The only issue it has is that it does not explain some of it in its entirety; it omits any mention of magnetic disturbances, but it's understandable because it is not required for core explanation. Overall, Gemini provided an extremely more accurate answer to the prompt compared to ChatGPT. The issue is that sometimes it is difficult to trust the validity if no citation of where the data is given or comes from. Even though this may be the case, it still provided a well-rounded explanation to fully understand the prompt.

5.4 Use In Coding

Although not full proof, ChatGPT has become very good at programming. For StrikeSense, ChatGPT served as a consultation of sorts for frameworks, optimizations, and general spit balling. It has assisted us greatly up until this point and will most likely continue to do so in this respect. We’ve always interacted within certain parameters when it comes to using ChatGPT as well as other GenAI tools and will continue to do so.

The first use of ChatGPT came from the creation of the computer vision portion of the project. When first undertaking this task, ChatGPT was consulted with how to best approach the problem. Specifically, the conversation consisted of ideas already fully formed being explained at our end and then asking which path would be the best option given the notes we already had. Once it gave its response, we evaluated and made the ultimate decision based on the evolving needs of the project. After that, during actual coding, ChatGPT was consulted for

any syntax errors or for any libraries that could potentially make task easier that we may have not been aware of.

The second use of ChatGPT came when looking into frameworks for our application to be ran on. Similar to the first instance, the goal was to find out which framework would best suit our use case. The only difference is that there were no real prewritten notes, just a constriction on which frameworks to choose from. Once it was provided, we then used it to help guide the setting up of the environment.

Between both of these use cases, the commonality remains that ChatGPT was used to assist in choosing the appropriate plan of action. This is how we plan on interacting with ChatGPT moving forward. As an aide of sorts for decision making rather than creating for us. We still design the system, write the code, and verify that it actually works on hardware. ChatGPT mainly helps us narrow down options faster so we can spend our time building instead of getting stuck on guesswork. It gives us possible directions, but we are the ones responsible for testing those ideas, validating performance, and making sure they align with the goals of StrikeSense. In that way, the final version of our project stays ours.

5.5 Use in Senior Design 1

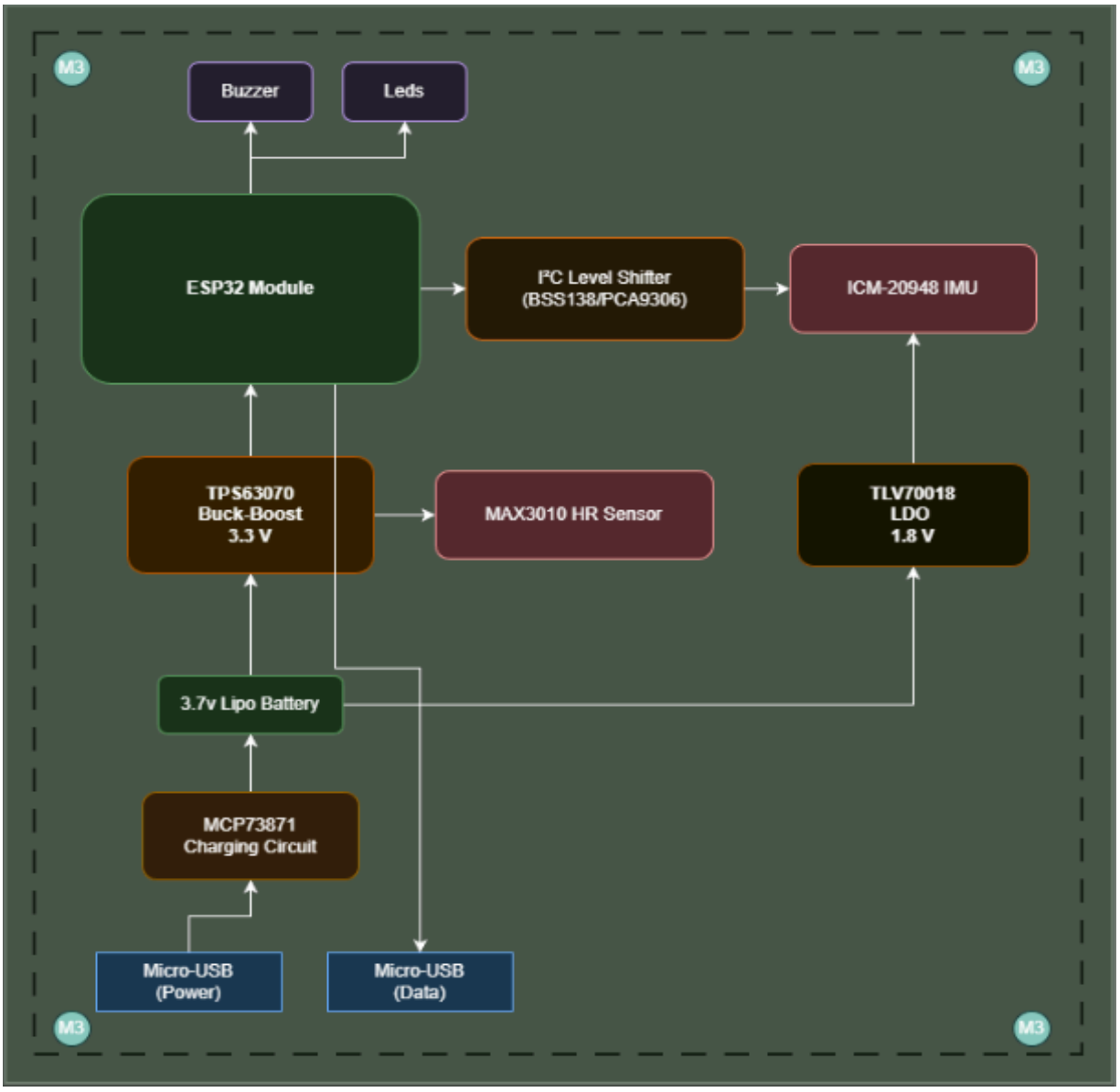
Throughout the design and documentation process, ChatGPT was used as a collaborative tool for research organization, writing refinement, and formatting assistance. Technical sources—such as datasheets, application notes, and TI design guides—were first identified and reviewed independently to ensure accuracy and understanding. After developing the initial technical content and analysis, ChatGPT was used to help refine explanations, improve clarity, ensure consistent formatting, and generate diagrams or structured outlines for readability. This approach allowed the writing to remain original and technically informed while leveraging ChatGPT for grammar, structure, and presentation enhancement rather than content generation.

Chapter 6 – Hardware Design

6.1 Overview

The wearable sensor module integrates multiple electronic subsystems—power regulation, data acquisition, wireless communication, and user feedback—into a compact, low-noise architecture. The central controller, an ESP32, coordinates sensor sampling, data processing, and wireless transmission while maintaining low power draw for extended battery life.

The system is powered by a 3.7 V Li-Po battery, stepped to regulated 3.3 V and 1.8 V rails. A TI TPS63070 buck-boost converter maintains a constant 3.3 V even as the cell discharges (3.0–4.2 V). A TI TLV700 LDO regulator provides a clean 1.8 V rail for analog sensors such as the ICM-20948 IMU and MAX30105 heart-rate sensor.



6.2 System Architecture

The hardware is partitioned into five subsystems:

1. Power Management – Converts and distributes regulated rails.
2. Microcontroller (ESP32) – Central processing and communication hub.
3. Sensor Array – Motion and physiological monitoring.
4. Signal Conditioning & Level Shifting – Voltage compatibility and noise filtering.
5. Peripheral Interface – LEDs and buzzer for feedback.

Primary Interfaces

- I²C for IMU and heart sensor
- GPIO for LEDs and buzzer
- UART for debugging

6.3 Power Regulation Subsystem

6.3.1 Main Regulation – Buck-Boost Converter (TI TPS63070)

The TPS63070 maintains a 3.3 V output with ~95 % efficiency and supports up to 2 A output [1]. It seamlessly transitions between buck and boost modes to handle battery discharge.

Breakdown	
Parameter	**Value / Spec**

Breakdown	
Input Range	3.0 – 4.2 V (Li-Po)
Output Voltage	3.3 V ± 2 %
Switching Freq.	2.4 MHz
Inductor	2.2 µH (≥ 1.5 A)
Efficiency	≈ 90 – 95 %

Design Notes

- Input/output capacitors (10 µF and 22 µF) placed within 2 mm of pins.
- Ground plane under device + thermal vias for heat spread.
- Feedback network R_1 = 1 MΩ, R_2 = 316 kΩ → 3.3 V output.

Comparison [web summary]:
Analog Devices LTC3531 (90 %) and MAX17291 (92 %) offer lower current (~600 mA) but reduced range; Murata OKI-78SR handles 1 A at ~95 % efficiency but costs ≈ $20. At ≈ $3.50, TPS63070 balances cost, current capacity, and efficiency.

6.3.2 Low-Noise Regulation – LDO (TI TLV70018)

The TLV70018 provides a quiet 1.8 V rail for analog sensors [2].
Dropout = 43 mV @ 50 mA; IQ ≈ 31 µA; PSRR ≈ 68 dB @ 1 kHz → excellent filtering.

Breakdown	
Spec	Value
Input	3.3 V
Output	1.8 V ± 1 %
Dropout	≈ 43 mV @ 50 mA
Quiescent Current	31 µA
Package	SOT-23-5

Why Chosen:
Low dropout extends battery runtime, and a compact package fits the board.
Alternatives like LT3045 offer higher PSRR (80 dB) but are larger and costlier.

6.4 Level Shifting and Signal Conditioning

The ESP32 (3.3 V) interfaces with 1.8 V sensors via a **bidirectional I²C level shifter**.
Discrete BSS138 MOSFET topology was selected for simplicity and low cost. Pull-ups (10 kΩ) are used on each side.

Alternatives:

- TI PCA9306 (1.2 – 5.5 V range) – excellent for low power I²C [3]
- Renesas SLG47525 – ultra-low voltage applications (< 2 V) [web summary]
- TXB0106 / TXS0108E – multi-channel logic translators for high-speed lines (not needed here).

Traces are short and shielded by a ground plane; a common ground reference is maintained for bus stability.

6.5 Power Distribution and Grounding

Line/Load Regulation

- Buck-Boost output variation < 1 % (10–300 mA).
- LDO output variation < 0.5 % under sensor load.

Grounding Plan

- Separate analog and digital grounds joined at a single star point [4].
- Ferrite beads between digital 3.3 V and analog 1.8 V rails.
- Wide power traces and short loops minimize IR drop and EMI.

6.6 Subsystem Hardware

6.6.1 Microcontroller – ESP32

- 3.3 V supply, dual-core CPU @ 240 MHz [5].
- I²C for sensor bus, GPIO for user interface, UART for debug.
- Includes crystal oscillator, boot resistors, and 0.1 µF decoupling at each VDD pin.

6.6.2 Sensor Array

ICM-20948 IMU (9-axis) [6] – 1.8 V operation, I²C bus, orthogonal mount.
MAX30105 Heart Sensor [7] – 1.8 V core, 3.3 V LED driver, optically shielded.
Sensors are placed at the PCB edge to minimize motion artifacts and light interference.

6.7 Battery and Charging Subsystem

A 3.7 V, 350 mAh Li-Po cell powers the system for ≈ 1–2 h.
Charging uses the **MCP73831** linear charger [8], driven by 5 V USB-C.
Charge current (100–500 mA) is set by a single resistor.
A status LED indicates charging state; the cell includes built-in over-charge and short-circuit protection.

6.8 PCB Layout and Connector Placement

Layout Guidelines

- Two-layer board (top signals, bottom ground plane).
- Short power traces; decoupling caps < 2 mm from IC pins.
- Thermal vias under buck-boost and LDO pads.
- Keep digital signals away from analog sensor lines to reduce capacitive and inductive coupling.
- Solid ground plane to absorb return currents and reduce EMI.

Connector Best Practices:

- Place connectors near board edges for accessibility and serviceability.
- Ensure mechanical tolerance and alignment for USB-C and sensor headers.

- Maintain separate ground pins for analog and digital sections when possible.

- Avoid placing connectors adjacent to heat-generating regulators to prevent thermal stress [9].

6.9 Prototype Validation and Summary

Prototyping Results	
Parameter	Result
3.3 V Rail	3.31 V ± 0.02 V (250 mA load)
1.8 V Rail	1.80 V ± 0.01 V (50 mA load)
Temp Rise	< 10 °C over ambient (30 min)

Validation used oscilloscope and multimeter readings to confirm load and line regulation.
Sensor outputs and wireless links were verified for stability up to 10 m.

Summary
The hardware achieved high efficiency, low noise, and robust performance.
Regulators maintain tight voltage tolerances; sensors perform accurately with minimal interference. PCB layout and connector placement adhere to mixed-signal best practices, resulting in a compact, stable, and ready-for-testing prototype.

6.10 ESP32

6.10.1 Description

The ESP32 requires high performance and is very low in power consumption. The microcontroller is one of the most used platforms for any sensor projects. It has the ability to integrate wifi and Bluetooth on a single chip, which makes it beneficial for engineers who want to build an advanced system. The one we selected is the ESP32-S3-DevKit N16R8. This has 16MB of flash and has a very good combination of computing power and a cost-effective package. In the rest of this section, we will be going over each of the necessary components involved in this microcontroller and why it was selected as an option for our strike sensor project.

6.10.2 Central Processing Unit

The CPU the ESP32 uses is the Xtensa LX7 dual-core 32-bit microprocessor, which has an operating speed of up to 240MHz. With each core being able to handle its own independent task, this allows for processing real-time data and communication protocols. Not only that it uses hardware acceleration for AI, which can improve the performance of digital signal processing or classification tasks. The CPU responds fast to sensor input and has efficient multitasking under FreeRTOS. For our project, it is imperative to have accurate time-sensitive data because we must be able to evaluate a fighter's technique. And to do so, we need deterministic timing and smooth data transmission with minimal lag occurring in our data.

6.10.3 Flash Memory

The ESP32 has a flash memory of 16 MB QSPI, this non-volatile memory is responsible for storing the device's firmware library and configurations. Unlike random access memory, information is retained even when power is on or off. This can also help the system to boot directly into the application. A larger flash capacity can integrate complex code bases and help with data logging for our project. This can include Bluetooth and Wi-Fi communication, which can all be done without exceeding memory.

6.10.4 PSRAM (Pseudo-Static RAM)

The PSRAM is 8MB and gives the ESP32 added memory for temporary data and dynamic memory. The reason this is beneficial for our project is due to the application's need to use real-time data buffering. The IMU and the

Magnetometer both need to be continuously sampled without data loss, and this is where the PSRAM comes in. Especially when wireless transmission fails briefly, there can still be a constant data connection.

6.10.5 Power Management and Voltage Regulation

The ESP32 has a 3.3V low-dropout regulator that converts the 5V from USB. It allows us to maintain a stable power source, preventing resets and or brownouts. This is essential for our system because voltage stability can prevent any noise in the analog data. Inconsistencies in voltage could lead to distorted sensor readings. The regulator in the ESP32 is another method for having consistent performance even under varying loads.

6.10.6 USB to UART Bridge

The onboard chip can convert a USB signal into a UART serial signal that the ESP32 can understand. This prevents the need for external tools and for simple firmware flashing and serial monitoring. The reason this is beneficial for our system is because it saves space and cost, and the bridge can streamline development.

6.10.7 USB to UART Bridge

A 40 MHz crystal oscillator is used for stable timing and provides a precise clock reference for the ESP32's processor. Having a stable clock can ensure that the WiFi and Bluetooth connectivity are reliable. Minor drifts can result in degradation of wireless performance, and the crystal guarantees consistent data. For the GPIO pins, the ESP has 40 different ones to choose from. With each of the pins being able to configure any of the communication protocols, such as SPI, I2C, and UART. This flexibility showcases the ability to use this microcontroller to design a wide range of sensors and actuators. For our project, it can be used variously. We decided on having the ADC read sensor voltage from the IMU and the UART transmit data through the wireless link.

6.10.8 Reset and Boot Buttons

The EN and BOOT are used to restart the microcontroller and place the chip in programming mode, respectively. These are necessary because they make testing and recovery simple. During the development process, these buttons provide hardware-level control for resetting all done within the board, which is quite useful for quick iteration cycles.

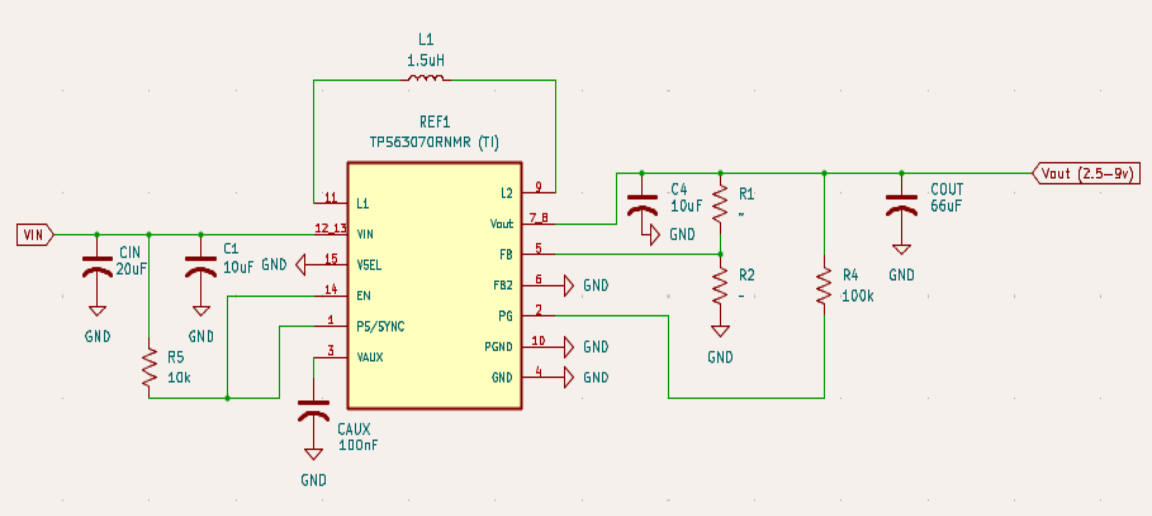
6.10.9 Decoupling and Filtering Components

The final part to mention about the ESP32 boards is the multiple ceramic capacitors and ferrite beads placed on the board. The purpose of these is to reduce the total amount of Electromagnetic interference. These components are placed alongside the power and analog pins, and these filtering components help with the signal integrity. Due to the magnetometers drawing a magnetic field, these filters are in place to prevent any magnetic disturbances and ensure accurate readings.

6.11 Schematics

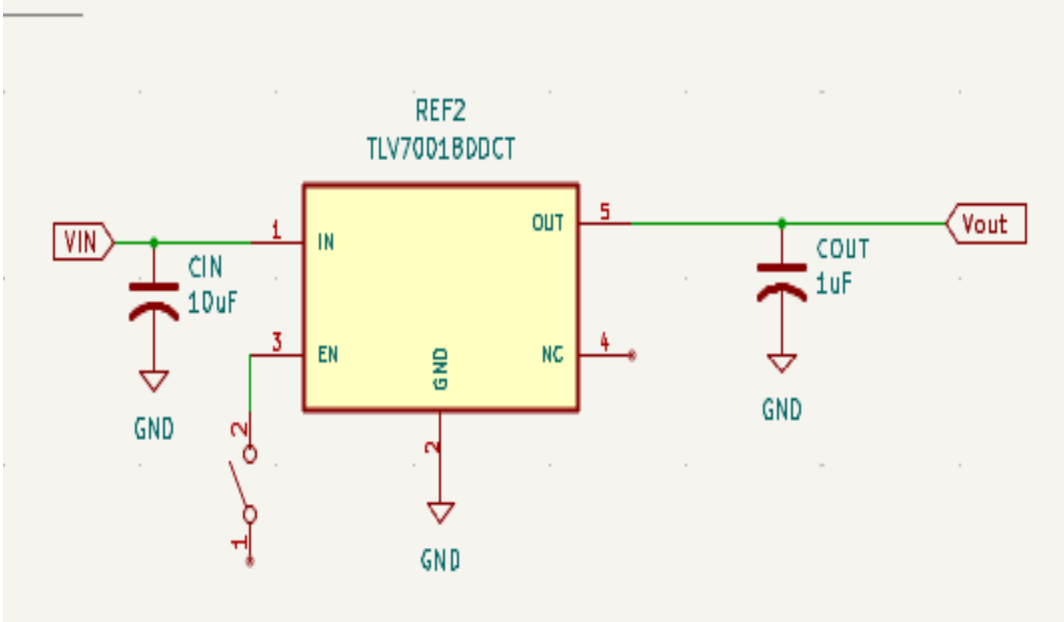
6.11.1 Buck Boost Regulator

TPS63070 application circuit provides a stable 3.3 V output from a Li-Po input (3.0 -- 4.2 V) using a single-inductor buck-boost topology. The inductor (L1) stores and transfers energy between input and output during switching cycles, while the input capacitors (CIN, CAUX) filter supply noise and reduce voltage ripple. The output capacitor (COUT) smooths the regulated voltage and stabilizes feedback. Together, these components enable efficient, low-noise power conversion



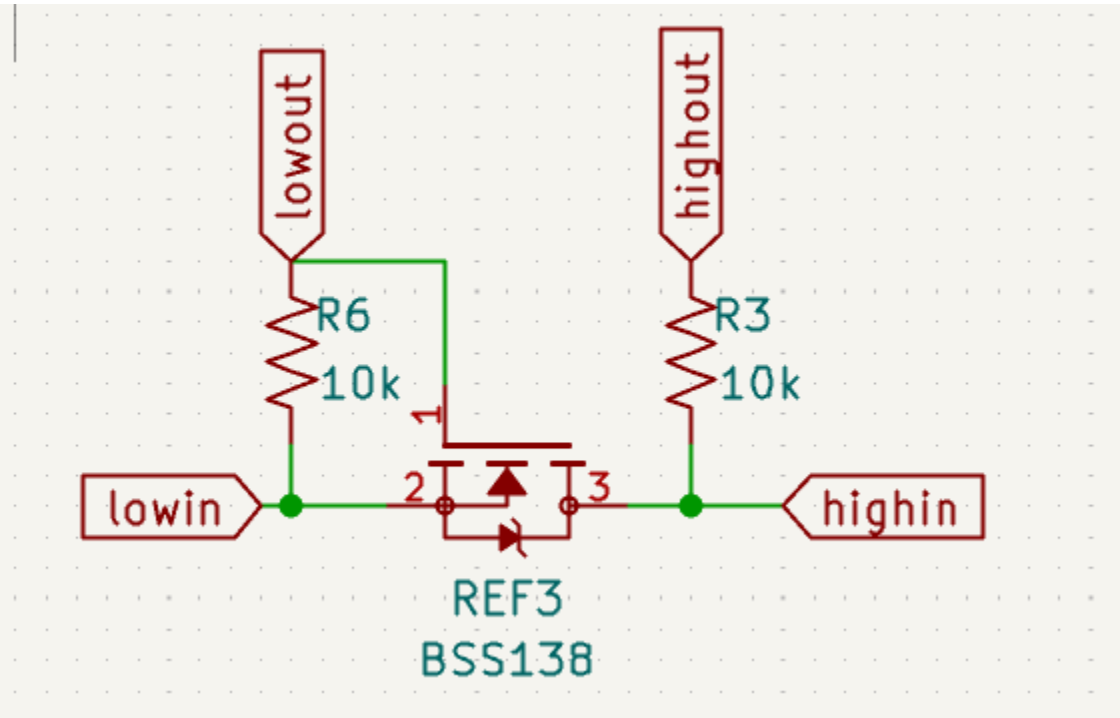
6.11.2 LDO

TLV70018 LDO application circuit generates a clean, low-noise 1.8 V rail from the regulated 3.3 V supply. The input capacitor (CIN, 1 µF–10 µF) filters supply noise and ensures regulator stability, while the output capacitor (COUT, 1 µF) smooths the regulated voltage and improves transient response. The EN pin enables control for low-power operation, and the simple two-capacitor layout keeps noise minimal — ideal for powering sensitive sensors such as the IMU and heart-rate module.



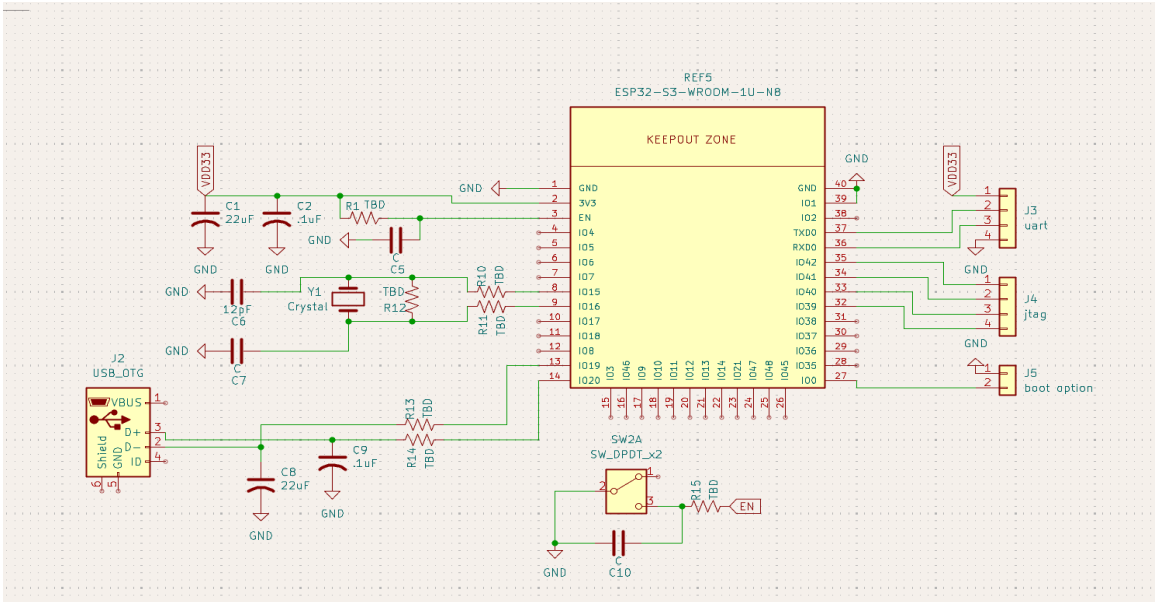
6.11.3 Level Shifter

BSS138 MOSFET level shifter circuit allows bi-directional voltage translation between two logic domains (e.g., 1.8 V and 3.3 V) by using pull-up resistors and the MOSFET's body diode to automatically pass signals in either direction—ideal for I²C or open-drain communication between mixed-voltage devices.



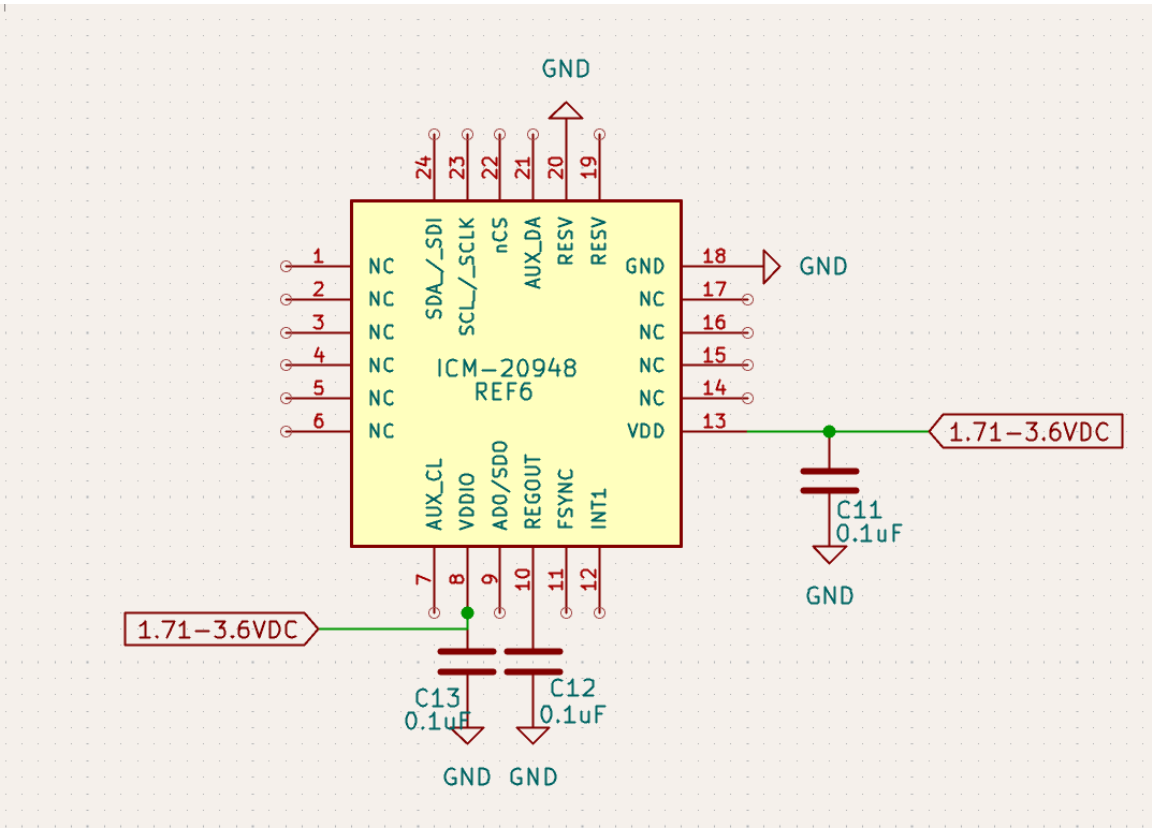
6.11.4 MCU

ESP32-S3 MCU application circuit provides the main processing and wireless control for the system. The USB-to-UART interface enables programming and serial communication, while the crystal oscillator ensures precise clock timing. Decoupling capacitors near power pins stabilize the 3.3 V rail, reducing noise and voltage ripple. Pull-up and pull-down resistors configure boot and enable states for reliable startup. Overall, this circuit forms the core of the wearable's control and data-processing system, interfacing with sensors, peripherals, and the power subsystem.



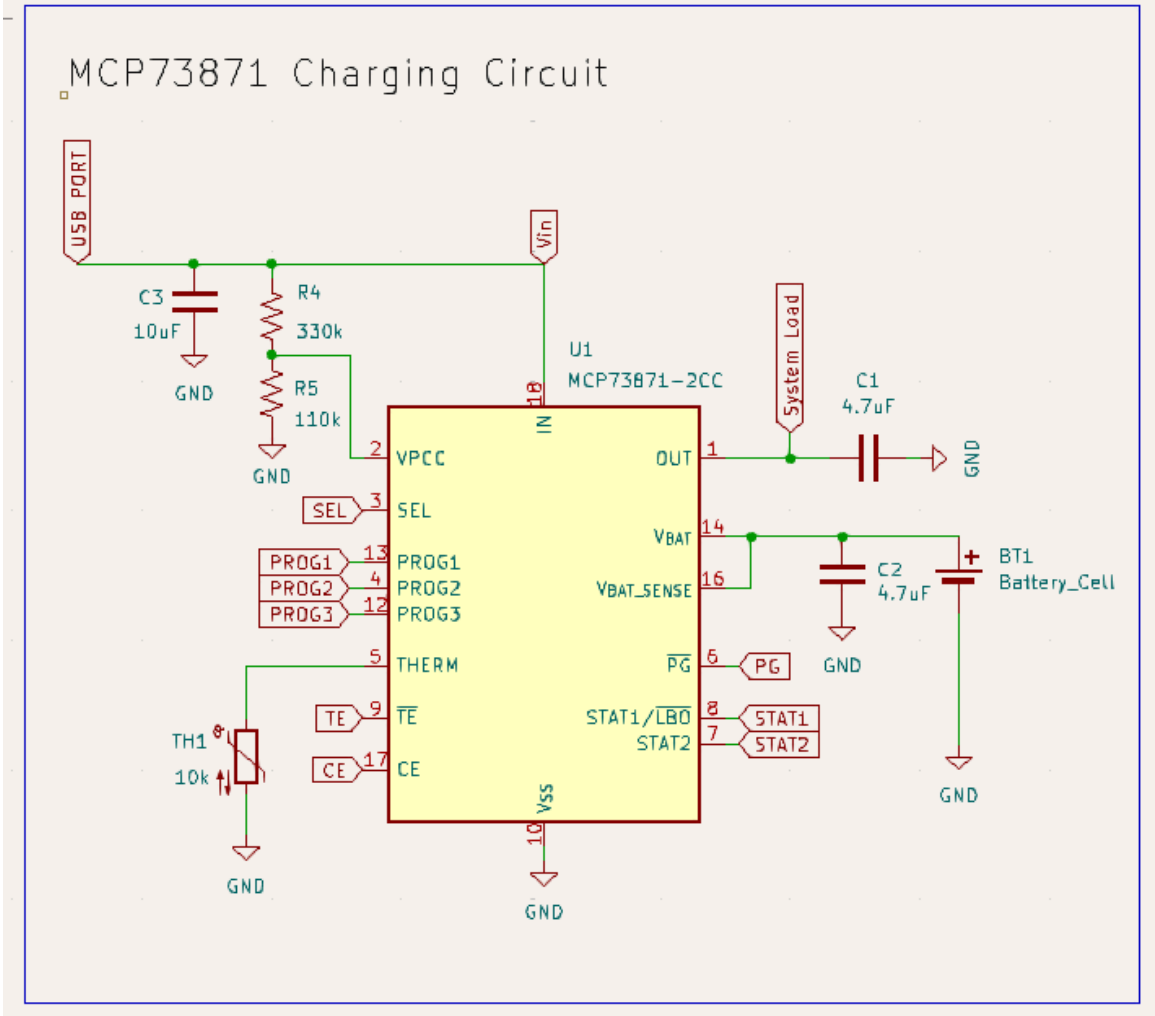
6.11.5 IMU

ICM-20948 IMU application circuit provides 9-axis motion sensing (accelerometer, gyroscope, and magnetometer) powered from the 1.8–3.6 V rail. Decoupling capacitors (C11–C13, 0.1 µF each) are placed close to the power pins to filter high-frequency noise and stabilize the sensor’s supply voltage. Communication occurs via the I²C bus (SDA/SCL) with optional auxiliary pins for additional sensors. This configuration ensures stable, low-noise operation essential for accurate motion tracking in wearable applications.



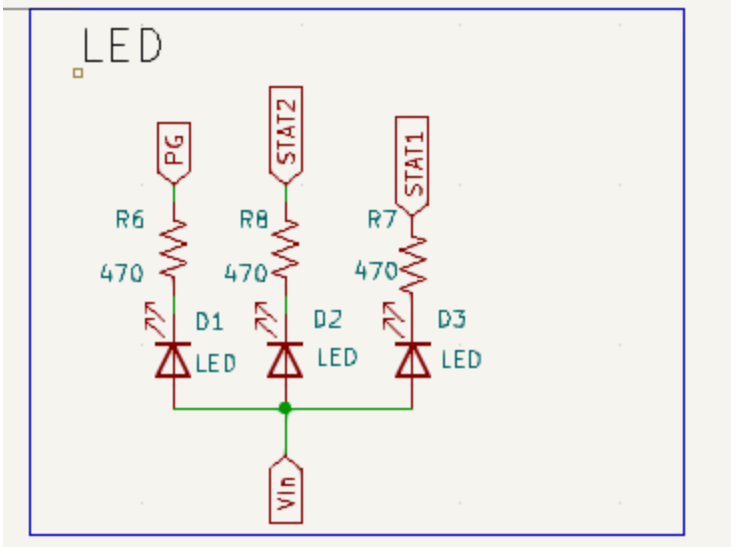
6.11.6 Charging Circuit

The LIPO charging circuit diagram is shown below. The circuit allows our system to operate and charge simultaneously. It provides the safety to manage LIPO charging and power sharing with the USB battery and Load. This circuit is a single-cell LIPO charging system built around MCP73871. It uses 5V from the USB port and filters through the capacitor, and goes through the charging circuit to charge the battery at BT1. We set a divider that sets the charging mode. PROG1-3 defines the charging current preconditioning and termination limits. In the datasheet, we were suggested to use a 10k ohm thermistor to monitor the varying temperature in the battery for the safety of the user. When USB power is present, it powers the battery and the system, and when the USB is removed, the battery will take over the system



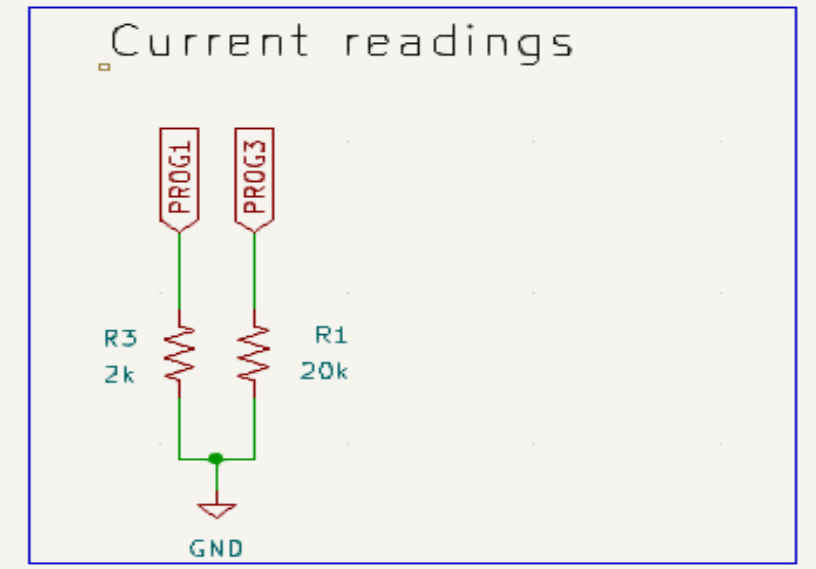
6.11.7 LED

The circuit below shows all the LED statuses for the charging circuit. Each one corresponds to a different output signal. PG LED lights when a valid input, like the USB, ends up being detected. STA1-2 LED is utilized as an open-drain output that lets us know the charging state.. It will either tell us if it is charging completely or if it has no battery. Paired with the LED is a 470 ohm resistor, which is used as a current limiter to slow the flow of current to the LEDs, which could potentially cause damage. The anodes of the LEDs connect to VIN, and the cathodes connect to the IC. So every time the status pin is low, the corresponding LED will turn on and tell the user the state of the charge.



6.11.8 Current Readings

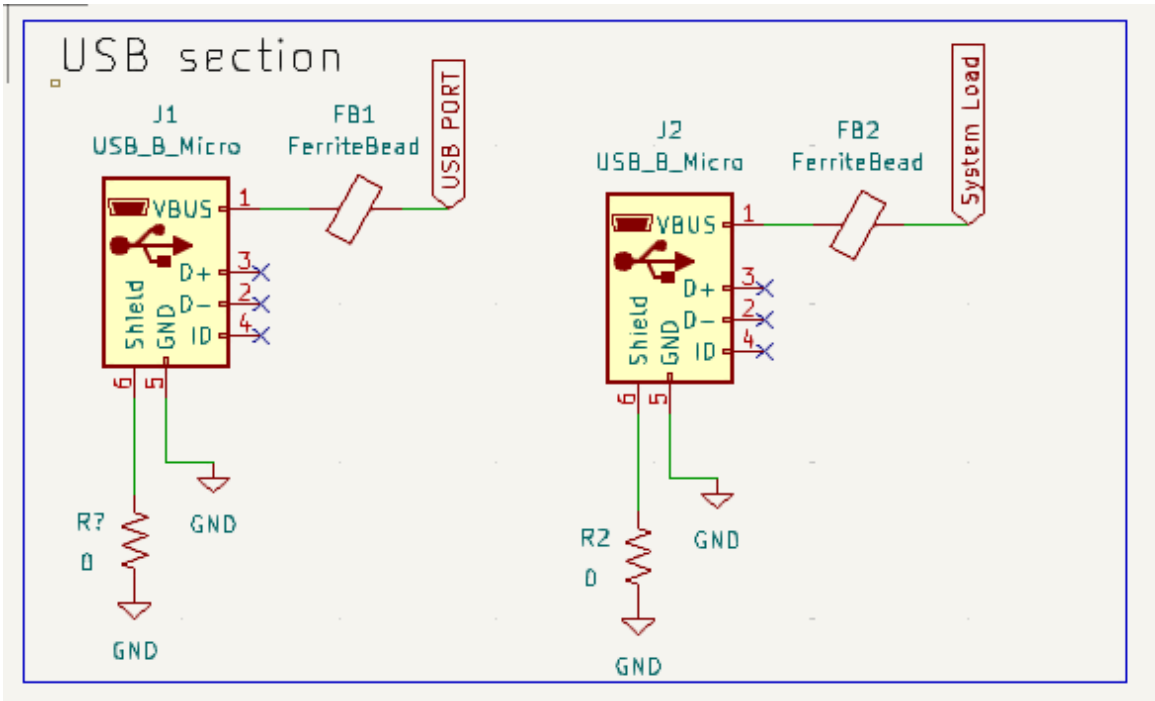
This section sets the current limits for the MCP73871 using both PROG1 and PROG3. The pins define specific current parameters as specified in the datasheet. Adjusting the resistor values gives us control over how quickly the battery charges. We can learn the behavior of the battery during low-voltage recovery. The resistors are critical for the safety of LIPO charging.



6.11.9 USB

This section provides the power and the data interface for the MCP73871 charging circuit and system load. There are two micro USB connectors, one serves as an input (J1) and the other serves as an output (J2) to pass through the system load and power the device. Each connector is a VBUS line that

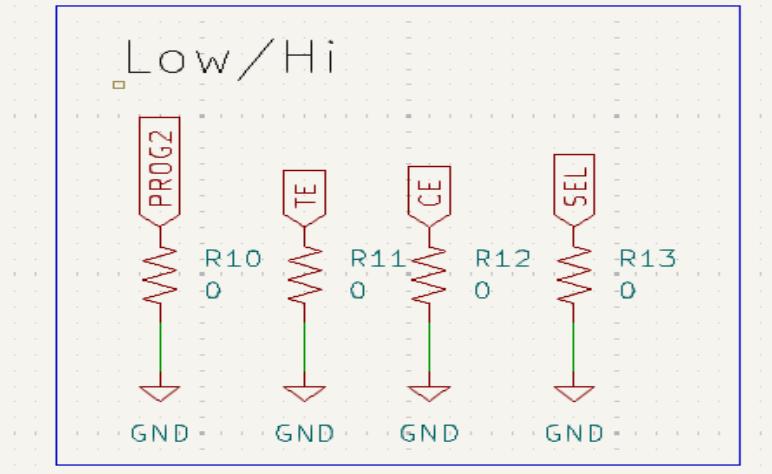
passes through a ferrite bead to reduce the frequency noise and protect against electromagnetic interference. The D+ and D- are used as the data lines, but because we are mainly utilizing them for power, we are leaving them unconnected. The other two pins are the Shield and the ground pins, and they are necessary to prevent static buildup. The decision behind the zero-ohm resistor is to allow for flexibility in design if we need to make changes later. This section creates a clean and stable supply voltage to charge the LIPO battery and minimize noise.



6.11.10 Low/Hi

This section configures digital control pins to allow them to act as changeable jumpers. Setting the resistors to 0 instead of direct shorts allows us to have them act as external controls or switches. This lets us fix logic levels and modify them during layout revisions. PROG2 sets the fast charge and termination and ends up grounding to its default low-level configuration. TE is used to disable and or select the default timing behavior for the charge safety. CE enables the charging

sequence, and SEL selects the USB mode and limits charge current to 500mA.



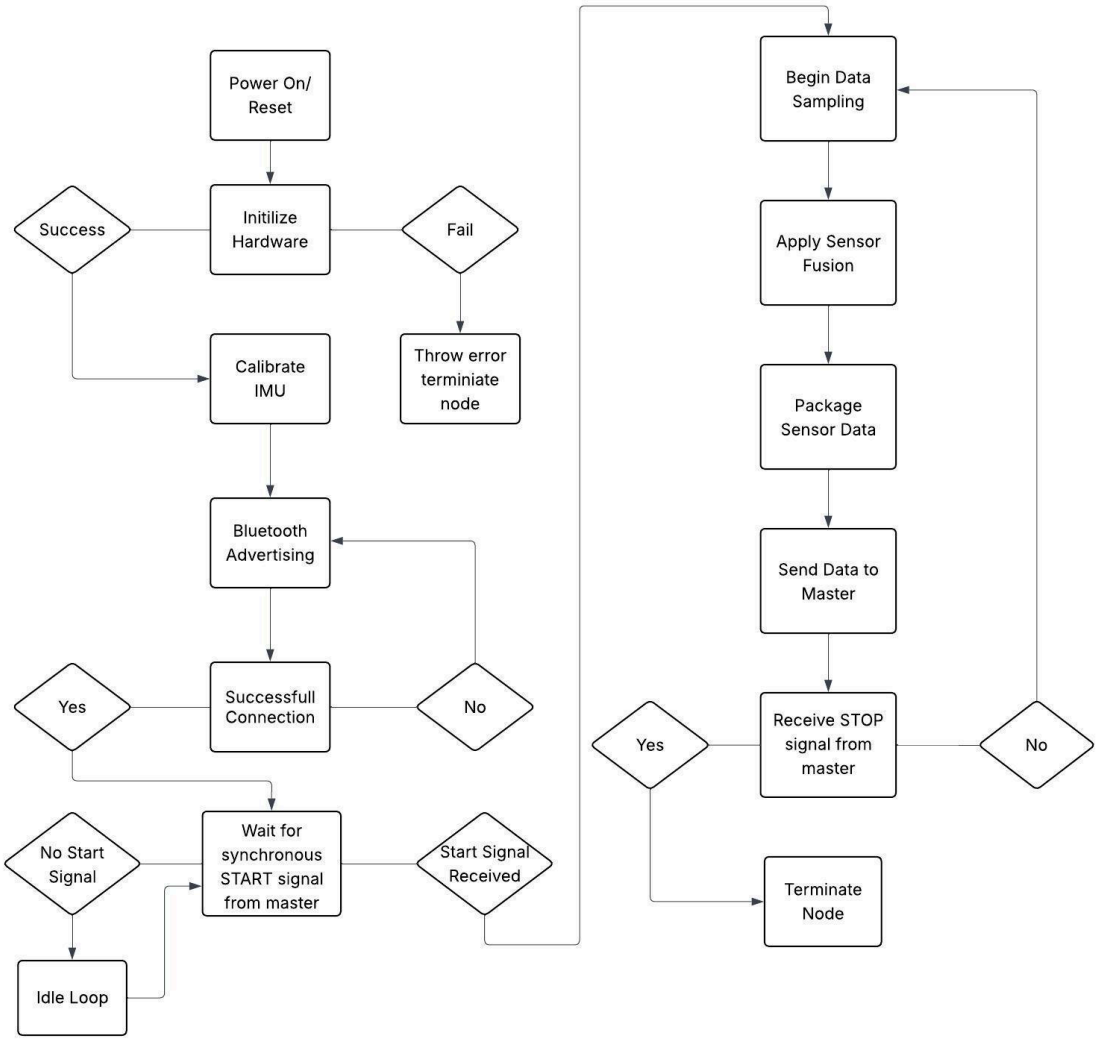
Chapter 7 – Software Design

7.1 Embedded Programming

7.1.1 Embedded Software Flow

The embedded software flow for each Strike Sense node begins with system initialization upon power on or device reset. If hardware initializations fail, for example the sensor module cannot be found, the node self terminates and throws an identifiable error. Once a successful initialization the IMU goes through a calibration routine ensuring accurate sensor readings. After calibration and hardware initializations the node begins Bluetooth advertising until a successful connection with the host is made. Upon a successful bluetooth connection the node enters an idle loop waiting for synchronous START signal from the master ensuring all sensor nodes begin sampling at the same time. Once the start signal is received the data-collection phase begins. The MCU continuously samples the IMU data, applies a sensor fusion algorithm, and neatly packages data (now in the form of quanterions) for wireless transmission. This loop continues until a STOP signal is received from the master, each node will then self terminate and

close communication. This flow will ensure synchronous data acquisition and transmission across all sensor nodes creating a reliable real-time system.



7.1.2 Attitude and Heading Reference System (AHRS)

IMU such as the ICM-20948 provides three different sensors:

- gyroscope to measure angular velocity (rotational velocity)
- accelerometer measures acceleration
- magnetometer which measures magnetic fields proving the heading.

Each sensor on its own presents a problem, the gyroscope is great short-term, however reading tends to drift over time because of tiny accumulated integration errors. While accelerometers and magnetometers are noisy, slow and unreliable during movements. The solution is to ‘fuse’ the sensor data together where the

gyroscope will provide fast updates and the accelerometer/magnetometer can be used for long-term stability and drift compensation. The combination of data is referred to as an Attitude and Heading Reference System (AHRS).

7.1.3 Madgwick Filter

Madgwick is a computationally efficient AHRS algorithm designed for 9-DoF systems. This filter employs a quaternion representation for orientation over estimating roll, pitch, and yaw directly. A quaternion is a mathematical representation of orientation that avoids a common issue with employing Euler angles known as “gimbal lock”. A quaternion takes the following form $q = [q_0,q_1,q_2,q_3,]$ this represents rotation in 3D space, and once you know q, roll, pitch, and yaw can be calculated using trigonometric formulas. To update the quaternion in real time, the Madgwick algorithm introduces an optimization step based on gradient descent. The idea is to minimize the difference between the measured direction of gravity and magnetic field (from the accelerometer and magnetometer) and the direction predicted by the current quaternion estimate(from the gyroscope). The algorithm will then adjust the orientation estimate by moving in the direction that most reduces the error between the predicted orientation and the sensor readings. A key tuning parameter (beta) is used to adjust how aggressive this correction is for example a large beta value corresponds to faster correction (i.e. more trust in accelerometer/magnetometer) this equates to less drift but more noisy/jumpy output. While a smaller beta value results in a slower correction, a smoother output but increased drift. This approach allows the filter to converge quickly toward the correct orientation with minimal computational cost, a major advantage for embedded systems like the ESP32.

7.1.4 Mahoney Filter

The Mahoney filter like the prior is another AHRS algorithm designed to combine gyro, accelerometer, and magnetometer data to estimate orientation in 3D space. Both algorithms start similarly, making a prediction of accelerometer and magnetometer data using quaternions derived from gyroscope data then compare and correct. The difference however comes down to how each makes the correction, Mahoney employs a feedback control approach rather than a gradient-decent optimization for correction. Specifically, a Proportional Integral controller (PI feedback). Applying PI feedback corrects the gyroscope readings

before the next integration, where the ‘P’ term pushes the estimate back immediately and the ‘I’ term cancels out long-term gyro bias.

7.1.5 Embedded Libraries

The following libraries were developed and maintained by Adafruit for use with the ICM-20948 sensor DevKit selected for this project.

7.1.5.1 Wire.h

The Wire.h library handles the I²C communication interface between the microcontroller (ESP32-S3 in our case) and auxiliary peripherals like the ICM-20948 sensor module. It implements the master/slave architecture of the I²C protocol including; start/stop conditions, addressing schemes, and data transfer methods. Within the header we can see at the lowest level it manages a two-wire bus (SDA for serial data and SCL for serial clock) using interrupt-driven buffers in order to coordinate between multiple devices preventing conflicts. In our project this library will be used to perform reads/writes to the ICM20948, enabling real-time data collection.

7.1.5.2 Arduino.h

The Arduino.h library is the primary header used in all Arduino-based projects. It defines key functions forming the Arduino runtime environment. This header also creates a hardware abstraction layer (HAL) that links C++ application code to specific MCU hardware drivers. It enables the use of interrupt handling and serial communication without the use of direct register-level manipulation

7.1.5.2 Adafruit_ICM20948.h

Highest level library for the ICM-20948 sensor. Building on Wire.h and Adafruit_Sensor.h, this library abstracts low-level register access into user-friendly functions. Internally it handles sensor initialization, configuration and also implementes a configurable digital low-pass filter. Additionally this library is responsible for converting raw register values into scaled SI units, and provides routines for calibrating offsets/bias.

7.1.5.2 Adafruit_ICM20X.h

Parent class for the ICM20X-series of sensors. Provides shared functions for register access, state management, and initialization. It is used by all ICM20X series sensors. The Adafruit_ICM20948 class inherits this library as a base and

builds upon it. The header ensures consistent behavior and code reusability across the various sensor variants. This simplifies maintenance and allows hardware changes with minimal software changes

7.1.5.2 Adafruit_AHRS.h

Implements Attitude and Heading Reference System (AHRS) algorithms such as the Madgwick and Mahony filters. These algorithms combine accelerometer, gyroscope, and magnetometer data to estimate orientation.

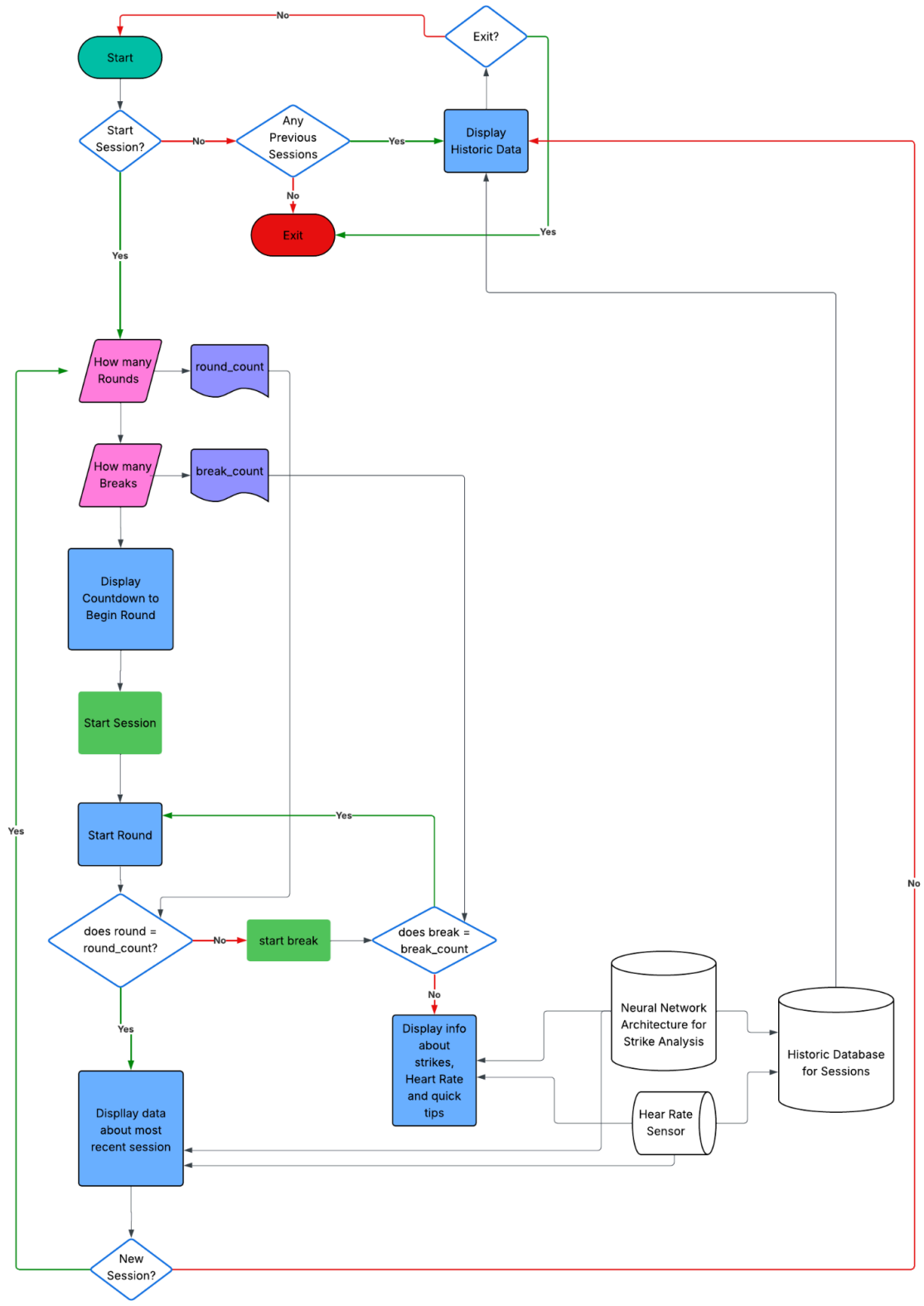
7.1.5.2 Adafruit_Sensor.h

Defines Adafruit's sensor interface, ensures consistent formatting across all Adafruit sensor libraries. This standardization simplifies multi-sensor integration

7.2 Dashboard Application

7.2.1 Introduction

One of the biggest aspects of StrikeSense is the ability to provide feedback in an effective manner. In order to do this, the way this information gets displayed must be clean, simple, yet informative. It must also be on a platform that is able to process the data being passed through. With both of these ideas in mind, the choice was made to select a desktop application as the vehicle for our dashboard. We've specifically selected the Qt framework which would include PySide6 and PyQt to be the framework we create in. There were other potential frameworks that could be used, however due to reasons that will be further broken down, Qt was ultimately selected. Additionally, the design language selected would be one focused on simplicity and quick feedback.



When the user launches StrikeSense, the interface first checks what they want to do. They can either start a new training session, view historic data, or exit the

system. If they choose not to start a session, the dashboard looks for any previous sessions. If historic data exists, it is displayed so the user can quickly review trends from earlier workouts before deciding whether to continue or close the app. If there are no previous sessions and they do not want to start, the system exits.

If the user chooses to start a session, the dashboard walks them through a simple setup. The user selects how many rounds they want to complete and how many breaks they want between rounds. These values are stored as round_count and break_count. Once the configuration is set, the system displays a countdown so the athlete has time to get ready, then officially starts the session.

During the session, the dashboard controls the flow of rounds and breaks. At the beginning of each round, StrikeSense starts recording strike data and heart rate data. After each round, the system checks whether the current round matches the planned round_count. If not, it triggers a break sequence. During breaks, it tracks how many breaks have occurred and compares this to break_count. If the planned number of breaks has not been reached, StrikeSense presents in-session feedback such as strike counts, heart rate information, and short coaching tips to guide the athlete before the next round begins.

In the background, live sensor inputs (including the heart rate sensor) are sent to the neural network architecture responsible for strike classification and intensity analysis. The processed results are both shown to the user during the session and written to the historic session database. This link between the dashboard, neural network, and database ensures that every session contributes to a growing performance history.

Once all scheduled rounds are complete, the dashboard displays a summary of the most recent session, including key metrics derived from the sensors and model. The user is then asked if they want to start a new session. If they select yes, the flow loops back to the configuration step. If they select no, the system can return to the historic data view or exit. Overall, the flowchart captures a full cycle: define a workout, capture and analyze data in real time, store it, show it in a clear way, and then give the athlete the choice to continue training or finish.

7.2.2 Qt Framework

Qt is a cross platform graphical user interface framework used to build full desktop applications with windows, buttons, menus, dialogs, charts, and everything else you expect from real software that a user can click. It is written in C++ at its core, but it exposes clean bindings in other languages, including Python. Qt handles a lot of hard problems for us. It manages windows and rendering in a consistent way on different operating systems, gives us layout tools so the UI can scale and resize, provides event handling when the user clicks or types, and includes higher level widgets like tables, plots, tree views, file browsers, and status bars. In other words, it lets us focus on logic and not on redrawing pixels by hand.

Under the hood, Qt is built around a signal and slot model. A signal is something that happens, like a button being pressed. A slot is a function that should run when that event happens. Connecting signals to slots is how Qt links the interface to our code. That design maps very naturally to how our system works. For example, we can connect a button labeled Start Capture to the function that begins sampling sensor data from StrikeSense hardware or connect a timer signal to a UI update slot that refreshes live heart rate and motion output on screen.

Qt is also mature and well supported. It has been used for professional tooling, scientific dashboards, internal engineering utilities, and even commercial end user products. That matters for StrikeSense, because our desktop application is not a toy. We are building something that needs to visualize real time biometric data, accept input from the user in a clean way, and package in a way that users can actually run on their own laptops without having to set up an entire development environment.

PySide6 and PyQt are Python bindings for Qt. In plain terms, they let us write a Qt application in pure Python instead of C++. Functionally, they are very similar. They both let us create windows, update the interface, listen for events, and ship a full desktop application. For our purposes, either one gives us the same basic capability. We will refer to this layer in general as "Qt in Python."

Using Qt through PySide6 or PyQt is important for our software design because our stack already leans on Python. Our data pipeline and inference code will be in Python. Our model code will be in Python. Our logging and analysis scripts are

in Python. By using PySide6 or PyQt, we can keep the entire application, including the user interface, in one language. That lowers complexity and lowers friction for development.

From an engineering point of view, PySide6 and PyQt also make it natural to stitch together the live data path and the presentation layer. We can receive streaming sensor data from StrikeSense hardware, buffer it in memory, run light processing or inference, and then update the UI labels, plots, or status indicators inside the same process. We do not have to spin up a second runtime, call over HTTP, or do cross language message passing just to draw a number on screen. That simplicity makes debugging and iteration much faster.

Another benefit is that PySide6 and PyQt allow us to use threading and asynchronous patterns that fit our workload. Running a model or reading from hardware can block if you do it directly on the main UI thread. With Qt in Python, we can push that work to a worker thread or background task and then send results back to the UI thread using signals. That will let us run live inference or data capture without freezing the interface. This is key for StrikeSense because we want to show metrics in real time and still allow the user to interact with controls while data is streaming in.

Qt, through PySide6 or PyQt, is a strong fit for StrikeSense for three main reasons: native desktop feel, tight integration with our sensor and model pipeline, and realistic deployment.

First, native desktop feel. We are not just dumping raw numbers to a console. We want a real operator console for StrikeSense. That includes things like a live heart rate panel, motion classification output from our model, timestamps, session controls, maybe even visual cues like green or red indicators for fatigue or overexertion. Qt is designed for this exact style of interface. It supports layouts that respond to window size changes, it supports custom widgets if we want to graph acceleration, and it gives us standard desktop interactions that fighters, coaches, or testers can understand immediately. We can build something that looks like an actual product dashboard, not a bland lab script.

Second, direct integration with the model. We plan to run local PyTorch inference as part of StrikeSense. For example, we may classify the type of strike, estimate intensity, or monitor physiological load during activity. Because PySide6 and PyQt are Python based, the UI code can import the same model code that we already run offline. We can load a TorchScript model or a standard trained PyTorch module, pass it data from the IMU and heart rate sensor, and then immediately reflect the inference results back into the UI. There is no need for a

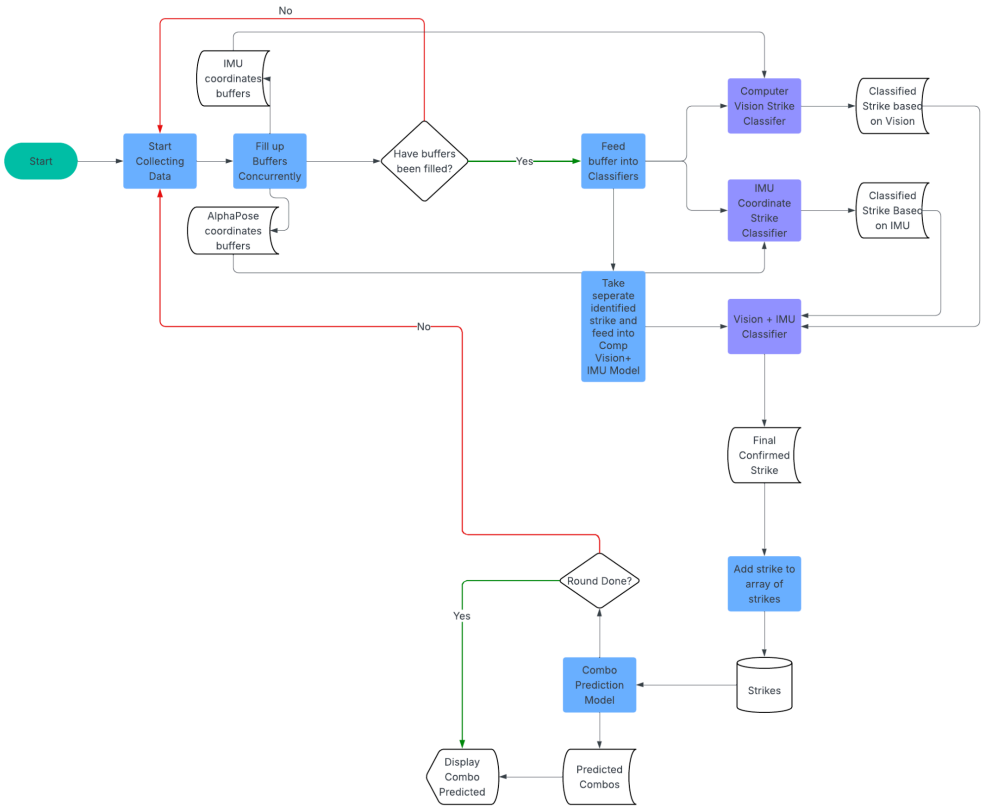
separate backend service or a web server to glue things together. Fewer moving parts means fewer failure points and faster iteration.

Third, deployment. Our goal is that the StrikeSense desktop tool can be run on a typical machine without a complicated setup. A Qt plus Python application can be bundled with tools like PyInstaller so that the end user does not have to install Python, does not have to pip install dependencies, and does not have to worry about system packages. Everything can ship in one installer along with the trained model weights. That lets us hand this to a reviewer, a TA, an advisor, or even a potential partner in a gym and say, here, run this and you will get live data. That is valuable for validation and valuable for credibility.

There is also a practical human factor. Python is a more approachable language that our team has been exposed to that is very powerful. Choosing Qt through PySide6 or PyQt means we are not forcing ourselves to context switch into a second tech stack for the front end. We can spend our time on the actual problems we care about, which are sensing, synchronization, interpretation, and presentation of live biometric and motion data from StrikeSense.

In short, Qt gives us a proven desktop UI framework. PySide6 and PyQt let us access that framework directly from Python, right next to our PyTorch logic and sensor handling code. That combination lines up with how StrikeSense needs to work in practice and supports the final vision for the project.

7.3 Neural Network Architecture



Once a session begins, StrikeSense starts collecting data from two main sources: the IMU on the athlete and the vision system (AlphaPose keypoints). These two data streams are written into separate buffers at the same time. The system waits until both buffers hold enough synchronized information to represent a potential striking motion. If the buffers are not full yet, it continues collecting data until there is a complete segment to evaluate.

When the buffers are ready, their contents are fed into the model stack for classification. First, the computer vision classifier labels the strike based only on pose information, while the IMU classifier labels the strike based only on motion and orientation data. In parallel, the system takes the detected strike segment and feeds it into the joint Vision plus IMU model. This fused model compares and combines both sources to generate a final confirmed strike label. The idea is to use the strengths of each sensor path so that noisy motion data or imperfect pose tracking alone does not cause incorrect predictions.

Each confirmed strike is then stored in an array of strikes for the current round. After every classified strike, the system checks whether the round is finished. If

the round is not done, the pipeline loops back to continue collecting and classifying new motion segments. This loop allows StrikeSense to track a continuous sequence of punches in real time while preserving every validated strike.

Once the round ends, the collected sequence of confirmed strikes is sent into the combo prediction model. This model analyzes patterns across the round to detect likely strike combinations, such as standard boxing combos or user specific patterns. The predicted combinations are then displayed to the athlete on the dashboard. This gives the user more than just isolated punch counts and provides meaningful insight into how they structure their offense, which can support coaching feedback, self review, and long term performance tracking.

Chapter 8 - System Prototype and Fabrication

8.1 Real World Prototyping

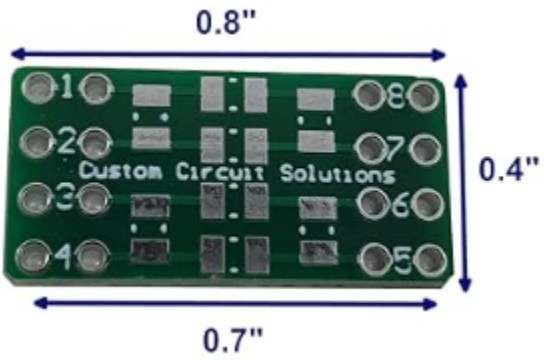
8.1.1 Prototype-First Development Strategy

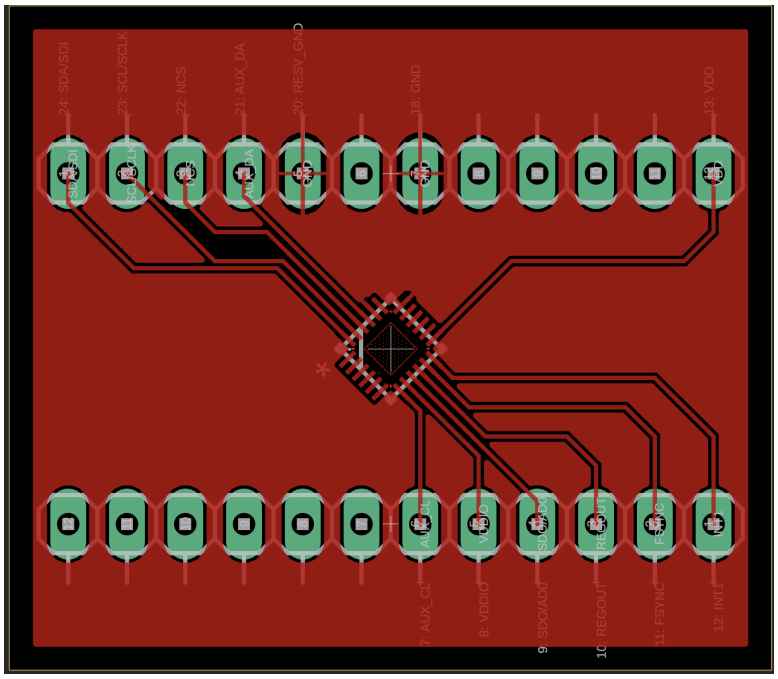
Before committing to PCB fabrication, our team intentionally prioritized functional prototyping. This allowed us to validate every subsystem—power regulation, communication buses, sensors, logic-level translation, and the ESP32 microcontroller—before locking in a physical board layout. Attempting to design a PCB earlier in the semester would have produced a draft that required multiple respins, added cost, and slowed progress. By focusing on prototyping, we now enter Senior Design 2 with a nearly complete, working system and a clear, validated roadmap for PCB fabrication.

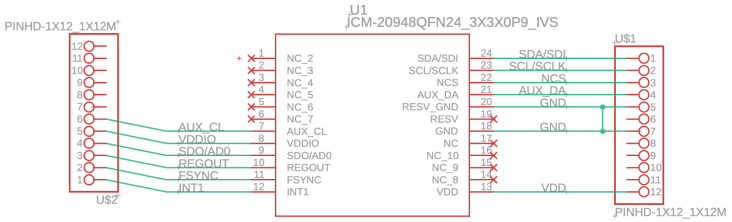
8.1.2 SOT Footprint Selection

During prototyping we standardized as many components as possible to SOT-series packages (SOT-23, SOT-23-5, SOT-223). These devices are small enough to emulate real PCB parts while still being easy to hand-solder. Additionally, low-cost SOT breakout adapters are readily available with next-day shipping, eliminating the need to design a temporary PCB. This approach

provided flexibility for trying multiple regulators, MOSFETs, and level-shifting circuits without incurring fabrication delays.







8.1.3 Soldering and Assembly

SOT-based regulators and MOSFET level shifters soldered easily and reliably. The only difficult device was the ICM-20948 IMU due to its QFN package, small

pitch, and exposed center pad. As a result, during prototyping we used the Adafruit breakout module, allowing us to validate behavior without attempting to hand-reflow a complex QFN device. This confirmed that QFN components should be integrated only after the PCB is finalized and proper reflow processes are available.

8.1.4 Breadboard and Bench Testing

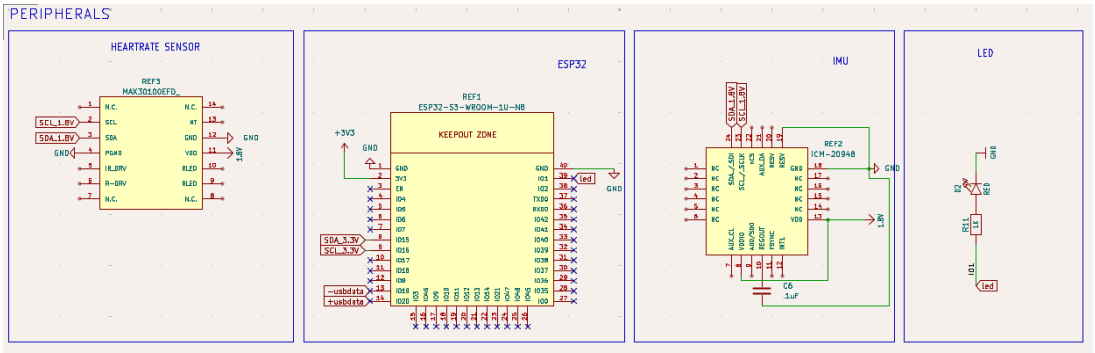
All subsystems were initially assembled on breadboards using jumper wires, bench power supplies, and USB interfaces. This allowed us to test components individually before integrating them. Bench power simplified early debugging by allowing precise control of supply voltage and current limits. Once each unit (MCU, IMU, regulators, level shifter, sensors) demonstrated consistent behavior, we combined them into a full prototype to evaluate system-wide interactions such as noise, I²C communication stability, and startup behavior.

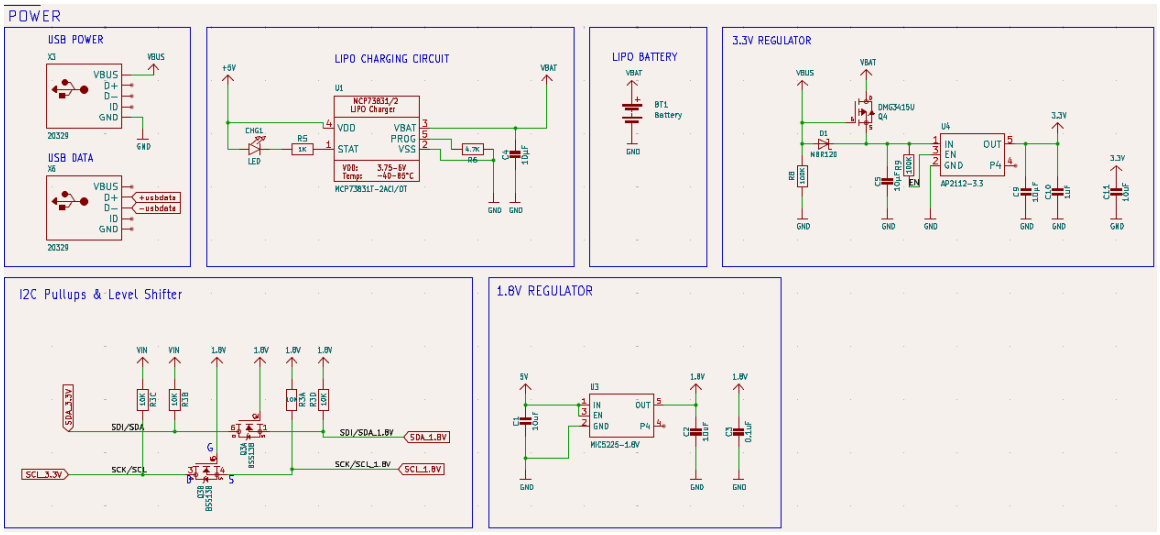
8.1.5 Justification for Delaying PCB Fabrication

Skipping early PCB fabrication allowed us to:

- Correct regulator choices after observing real startup and noise behavior
- Validate pull-up resistor values for I²C translation between 3.3 V and 1.8 V
- Confirm ESP32 boot stability under radio load
- Characterize sensor noise and decoupling requirements
- Identify necessary filtering and grounding strategies
- Test high-g acceleration response before designing mechanical mounting

If a PCB had been made earlier, these discoveries would have required redesign and re-fabrication. Instead, our prototype-first workflow led to a stable, nearly finished hardware design, ensuring that the Senior Design 2 PCB will be based on validated electrical and functional behavior rather than assumptions.





Chapter 9 - System Testing and Evaluation

9.1 Hardware Testing

9.1.1 ESP32

In order to test each ESP32-S3 DevKit worked as expected, we first verified power then functionality. To verify power the ESP32-S3 DevKit was plugged into a 5v USB hub and using a multimeter we measured both the 3.3v and 5v power bus. These values were within nominal ranges. Additionally the onboard LEDs immediately lit up providing additional means of verifying power. In order to verify functionality a couple test scripts were flashed to each DevKit. These scripts included a simple ‘hello world’ script which verified writing to a serial port and a blinking led script that verified the functionality of various GPIO pins.

9.1.2 Level Shifters

In order to test the BSS138 Mossfet’s application as a level shifter we mounted the IC on an SOT-23-3 breakout board acquired from Amazon. This allowed for us to breadboard the levelshifting application circuit. The premise of a level shifter is to transcribe serial data lines to different voltage levels, in our case a bidirectional translation of 3.3 to 1.8v. Our chosen serial data transfer protocol is I2C which consists of two lines SDA and SCL. These lines are held high via a pull up resistor (logic ‘1’). Either master or slave can pull these lines low (logic ‘0’). In order to test this behavior we used a 10k resistor to pull up the data lines on each side of the level shifter to their respective high voltages. For the BSS138

source is connected to the logic line with the lower voltage and therefore pulled up to 1.8v and drain is connected to the logic high line and therefore pulled up 3.3v. Lastly,the gate is tied to the lower voltage reference(1.8v). To mimic a slave or master pulling a line low we shorted one side to ground and verified the opposite side was pulled low and that each side was pulled back up to their respective high after removing the short. This was tested both ways.

9.1.3 ICM 20948

To test the ICM 20948 DevKit developed and purchased from Adafruit, we verified power then functionality using a simple test script. The DevKit is powered by 3.3v which was provided by the ESP32. Once powered we were able to measure the output of the included 3.3v -1.8v regulator using a multimeter, additionally a power LED provided some reassurance that the board was working appropriately. To test functionality we adapted sample codes provided within the Adafruit ICM20948 libraries that printed raw IMU data in the form of (x,y,z) axis from each sensor (accelerometer, gyroscope, magnetometer) to a serial port.

9.1.4 3.3 V Regulator (AP2112-3.3)

To prototype the 3.3 V regulation stage, we soldered the AP2112-3.3 LDO onto an SOT-23 breakout board from Amazon, allowing easy breadboarding. The circuit used the recommended 10 µF input and output capacitors, placed close to the device. We powered the regulator using a benchtop supply set to 5 V and 3.7–4.2 V (to simulate USB and battery), and verified proper operation using a multimeter. The AP2112 consistently produced a stable 3.3 V output under small test loads, confirming it as a reliable supply for the ESP32 and 3.3 V peripherals.

9.1.5 1.8 V Regulator (MIC5225-1.8

The 1.8 V rail was tested by mounting the MIC5225-1.8 regulator on an SOT-23 breakout board for easy prototyping. Following the datasheet, we used a 10 µF input capacitor and a 10 µF + 0.1 µF output capacitor for stability. The circuit was powered from the regulated 3.3 V rail and verified with a multimeter to ensure a steady 1.8 V output. Light resistive loads were applied to confirm the regulator remained stable, validating its suitability for powering the IMU’s sensitive 1.8 V core.

9.1.6 Battery Management System

TIn this section, we will discuss a potential prototyping design of our battery management system. When we began prototyping, the goal was to create a reliable charging module for the single-cell Lipo module. We utilized the data sheet to refine our design of the breadboard. To summarize the goal of this essential component, we need to discuss how the BMS has more value than just a charger. At a minimum, the BMS must perform the following tasks to ensure user safety.

This IC provides simplicity to the overall design of the BMS. With only a few external components, this IC allows us to focus on the more complex aspects of the BMS prototype. We utilized a Linear charger. This dissipates, which occurs and causes a voltage drop in the input and the battery voltage. This makes the design an essential consideration. While designing the breadboard for the BMS, we needed to select the right type of resistor to connect to the PROG pin and use capacitors that create a stable regulation loop. To get to this stage, we need to put together all the necessary components

For a successful BMS prototype, we had to ensure that it had a stable source. Every IC has a USB 5V input, providing a convenient and widely available power source. We decided to include a poly fuse that can protect against accidental shorts, which can essentially cut off current automatically. We utilized a USB Type-A breakout board so that it gives clean access to 5V and GND. These parts ensure that the charging of the IC can have a clean and safe input.

Here we will discuss the support components of the Charger IC. We will start with the Input capacitor, this 4.7 microfarad ceramic capacitor can stabilize the voltage at the VDD pin. In our circuit design, we want to make sure the internal pass transistor can be controlled predictably. The output battery capacitor, 4.7 - 10 microfarads for the output is used to stabilize the VBAT pin for the constant voltage charging phase. PROG resistor sets the charging current in our prototyping design, we decided to make it about 2k ohms so that we can get a charging current output of 500mA. Doing so, we had to make sure we considered the battery capacity and thermal limits. STAT LED series resistors offer visual feedback when the IC is actively charging it pulls the STAT low and turns the LED on, and when the battery is full, the LED will turn off.

Once the main charging circuit is finished, the next step in the prototyping process is to integrate the protection circuit. This ensures that the LIPO cell

would remain safe under all operation scenarios. While breadboarding, we added test points so that the voltages and currents could be measured during each phase. This validated our system behavior while making sure that we are flexible for adjustments through the prototyping phase of our design.

Now moving onto testing the power path and load behavior of the system, we added a small DC regulator to supply a stable voltage and saw how the charger interacted with an active load. We evaluated how the system responded when the battery was low, nearly full, and removed, confirming that the IC worked correctly with the supporting components. Through this hands-on testing, we were able to verify that the prototype behaved like a complete BMS rather than a simple charger, laying a reliable foundation for future PCB integration and system refinement.

9.2 Software Testing

9.2.1 Introduction

While our software stack for the prototype was not directly reflective of our final product, the result of our prototype was. For our prototype we utilized the following software: Visual Studio Code, Arduino IDE, Python, Platform IO, Influx DB, and Grafana. Each of these served a distinct purpose when developing the prototype and allowing us to understand how our final vision would look like.

Our main area of interest for testing software was reading and displaying movement data from our IMU modules as well as heart rate information. This was done by defining a data processing pipeline and then fitting it for different use cases.

9.2.2 Preliminary Steps; Setting up the Environment

As a team, our first goal was to confirm that the Adafruit Feather Sense was giving us reliable IMU data. Using the Serial Monitor, we watched the raw readings update as we moved the device. This step was important because it allowed us to make sure the IMU was communicating correctly and understand the data format we would eventually parse into a database.

Once we confirmed that the data we needed was consistently printed to the Serial Monitor, we closed the Serial Monitor so no other program would compete

for the COM port. Only one application can read from the serial connection at a time, so shutting it down was a necessary part of the workflow.

This early testing phase helped us build confidence in the sensor before moving toward automated data collection.

9.2.3 Sensor Fusion Algorithm Testing

In order to verify the functionality of the chosen sensor fusion library from XioTechnologies, we adapted the raw data script and first included the sensor fusion library. Mimicking the libraries sample code we initialised the fusion instance and set up some basic parameters, these parameters will be manipulated later down the line to improve accuracy, at this stage the goal was focused on printed roll, pitch, and yaw to a serial port and seeing this data updated appropriately based on the devices movement. After creating the instance and define parameters, we use the same method to sample the IMU for raw data, the difference is now this data is passed through a helper function with a time delta in order to calculate quaternions which will then be converted into Euler's angles (roll, pitch, and yaw). The output was verified and although at rest we see some bias, rotating the sensor results in a spike is the appropriate data stream.

9.2.4 BLE Testing on the ESP32

In order to verify the chosen BLE library, NimBLE, we adapted their client and server test scripts. The goal of this testing was to see any type of data sent over BLE and then printed to a serial port; For this we choose a timestamp that records time since boot of the ESP32. For our application the ESP32 will act as a BLE server, to set this up first we generated and defined a unique Service and Characteristics UUID. Next we created one instance of a BLE server and an instance of a BLE characteristic (what holds the actual data). From there some initial server setup was required, which was all handled by helper functions within the NimBLE library, such as linking the Service and Characteristic UUIDs to the BLE server as well as the Characteristic instance itself. Next the server is started and an advertising packet is defined, containing a unique advertising name and listing the Service UUID. The client side scans for this unique name, connects, and subscribes to notifications. This initially failed as the Server could not be found by its name, to overcome this we modified the Client to connect based on the servers address. We found this address by downloading a BLE packet sniffer, and after some brute force we had the server's address and the Client then could

easily connect. After connecting we could see that the timestamp data was sent over BLE and printed to a serial terminal on the Client side.

9.2.5 Translating to Python

Once the sensor data was validated, we shifted our workflow to Python. The Arduino IDE was great for early debugging, but we needed more control over parsing values, handling timestamps, and forwarding the data into a database. Python gave us full flexibility to read the serial prints, split out the IMU values, tag each value correctly, and format them in a structure suitable for a time series database.

We wrote a Python script that listened to the same serial port the Arduino IDE had been using. This script continuously read the incoming lines of text, extracted the pitch, yaw, and roll values, and prepared them as data points. Each point was then written directly into our local InfluxDB instance.

Switching to Python also helped us prepare for the final version of StrikeSense, where the sensor will communicate over Bluetooth instead of a wired USB connection. The same parsing logic will apply, so building the pipeline in Python was a future-proof choice.

9.2.4 InfluxDB

We chose InfluxDB as our database because it is designed for high-frequency time series data, which fits motion tracking perfectly. The database performs well with rapid inserts, handles timestamps automatically, and includes built-in tools for filtering, aggregating, and visualizing trends.

Another major factor was the strong support for Python. The official InfluxDB Python library allowed us to send data with only a few lines of code. This made integration simple and reliable during prototyping.

We hosted a local instance of InfluxDB located at http://localhost:8086/. Inside this instance, we created a bucket per IMU module labeled **IMU1** and **IMU2** respectively. These buckets served as the main storage location for all incoming motion readings. Each data point written from Python included fields for pitch, roll, and yaw otherwise known collectively as a quaternion along with a timestamp, allowing Grafana to query the data in real time. Once data began

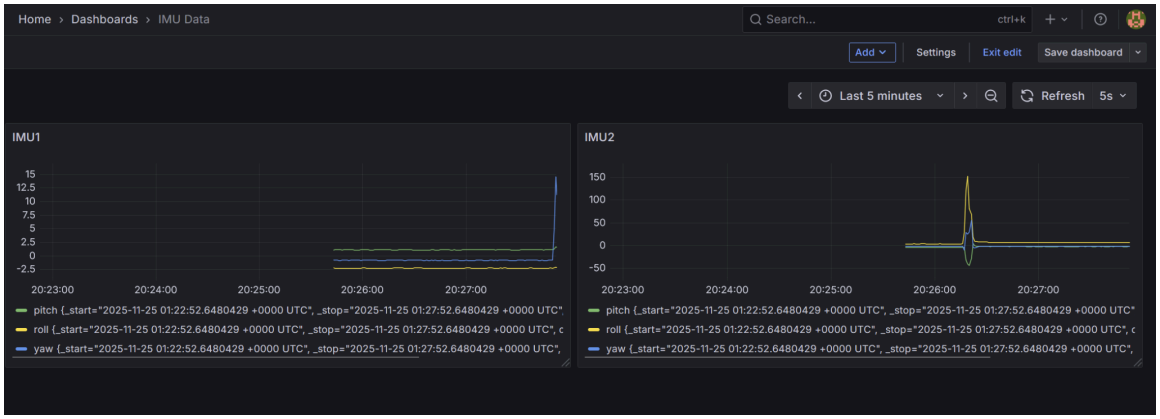
flowing into the bucket, we could already validate our script by running quick Flux queries inside the InfluxDB interface

9.2.5 Data Visualization Through Grafana

With InfluxDB fully running, the next step was to visualize the IMU output. We chose Grafana because it is widely used in HPC cluster monitoring and home labs, so it is a proven system for real time dashboards. Grafana focuses on creating clean, interactive panels, and it integrates natively with InfluxDB without extra tools or plug-ins.

To set up the dashboard, we added InfluxDB as a data source and linked it to the IMU Motion Data bucket. Once the data source was connected, we wrote simple Flux queries that returned pitch, roll, and yaw values over the last few minutes. Grafana plotted these values automatically, which gave us smooth, live graphs that updated as the sensor moved.

This visualization stage confirmed that our entire data pipeline was working. We could move the Feather Sense around, send motion data through serial to Python, write it into InfluxDB, and watch the motion curves update instantly in Grafana. Below you can see the graphs on the dashboard that we created



InfluxDB and Grafana worked well for us because they complement each other. InfluxDB handles fast time series inserts without slowing down, and its structure fits our IMU data perfectly. Grafana connects directly to InfluxDB and turns the stored data into live visualizations.

Using both tools allowed us to see motion patterns clearly, identify outliers, and validate that our sensors were behaving as expected. The setup also mirrors how monitoring works in larger engineering environments, which gives our prototype a strong technical foundation.

9.3 System Integration

9.3.1 Introduction:

With all hardware individually breadboarded and tested along, with functional software capable of testing our design It is key to move on to integrating our individual components together starting with subsystems then full system integration of hardware and software.

9.3.2 Software Integration

It is imperative to create one streamless program to run on the ESP32, In conjunction with one streamless program running on a PC to capture and graph the data.

9.3.2.1 Integration of Sensor Fusion and BLE with multi-node support

Individually at this point we have a program that can sample raw data from the IMU in the form of x,y,z data points form each sensor (accelerometer, gyroscope, magnetometer) to convert it using sensor fusion to the forms of quaternions. And a pair of BLE programs, a server that runs on the ESP-32 and a client that connects and subscribes to notification from said server running on a PC. Integration of these two components including multinode support would entail a single program flashed on the ESP32 that can sample and convert IMU data to quaternions, neatly package this into a BLE characteristic and send the data in a timely and ordered manner and a BLE client that can connect and unpackage the data neatly from multiple servers. To accomplish this a structure is defined that contains three floating point values (roll, pitch, and yaw). A BLE characteristic is also initialized containing the same empty structure. After each sample/fusion instead of the data being printed to a serial port a new instance of the structure is created containing the most recent data and using a helper function the characteristic is updated and the client is notified. Additionally to help decipher between each data stream, each BLE server advertises a set name to ease connectivity. On the client side we could verify the integration of BLE and Sensor Fusion by subscribing to both nodes by their unique name and unpacking the data to print to a serial port. This was successful after a few issues dealing with

advertising names. Initially within the advertising packet, data such as the serviceUUID was included and due to the limit size of this packet the name was dropped and the node was assigned the name ‘none’.

9.3.3 Hardware Integration

It is imperative that all individual hardware components can be combined to form a complete and functional system.

9.3.3.1 Integration of ICM20948 and BSS138 Level Shifters

For all of our software testing and validation we used an ICM20948 DevKit. In our final design we will incorporate our own PCB design utilizing the ICM20948, knowing that, it is important to perform some initial testing on our own design. To accomplish this we designed our own custom breakout board in order to breadboard a barebones ICM20948. It is important to integrate the level shifters at this point, as in order to verify functionality we will run the same test script that prints raw sensor data to a serial port, this issue with this is logic level for the ESP32 is 3.3v while the ICM20948 is 1.8v thus the need for level shifting arises. Using two level shifters, one for SDA and SCL, we connected the low side to the ICM20948 and the high side to the ESP32 and used a test bench power supply to provide both a 3.3v rail and a 1.8v rail. After some initial debugging, primarily not connecting the ESP32 ground to the common ground of the circuit we successfully received data on the serial terminal.

9.3.4 Full System Integration

This final integration step will confirm the complete functionality of our system. In this step we integrated the following subsystems; power system, sensor system and the ESP32 running the fully integrated program mentioned above in the software integration section. This integration step was seamless as prototyping individual components and integrating to create subsystems ironed out any bugs.

Chapter 10 - Administrative Content

10.1 Bill of Materials

Bill of Materials						
#	Category	Part / Option	P/N (Pkg)	Qty	Est. Unit $	Notes
1	**MCU / Wireless**	ESP32-WROOM-32E	ESP32-WROOM-32E (SMD)	1	5	Wi-Fi/BT; 3.3 V I/O; FCC modular.
2	**9-DoF IMU**	ICM-20948	ICM-20948 (QFN-24, 3x3mm)	1	6.5	Accel+Gyro+Mag; VDDIO ~1.8 V
3	**Level Shifter**	TXS0108E	TXS0108EPWR (TSSOP-20)	1	1.5	Bidirectional 3.3 V <-> 1.8 V
4	**Li-ion Charger IC**	MCP73871	MCP73871 (QFN/MSOP)	1	2.5	USB charger + power-path
5	**Battery Connector**	JST B2B-PH	JST B2B-PH-SM4-TB	1	0.5	2-pin, vertical, 2.0 mm pitch
6	**USB-C Receptacle**	USB4110-G	USB4110-G	1	1	USB-C 2.0; add 5.1 kΩ CC resistors
7	**3.3 V Regulator**	TLV75533	TLV75533PDBVR	1	0.5	LDO 500 mA

Bill of Materials						
8	**1.8 V Regulator**	TLV75718	TLV75718 PDBVR	1	0.5	LDO 300 mA
9	**Input Protection**	Polyfuse	MF-R110	1	0.3	USB input protection fuse
10	**Surge/ESD**	TVS diode	ESD9B5.0 ST5G	1	0.2	USB ESD protection
11	**Pull-ups (I²C)**	Resistor	0603, 2.2–4.7kΩ	2	0.02	I²C bus pull-ups
12	**CC Resistors**	Resistor	0402, 5.1kΩ	2	0.02	USB-C CC pulldowns
13	**Bulk Capacitors**	Capacitor	0603/0805, 10µF	6	0.03	Power filtering
14	**Decoupling Capacitors**	Capacitor	0402/0603, 0.1µF	12	0.01	One per power pin
15	**Sense Divider**	Resistors	0402/0603, 100kΩ+200kΩ	4	0.02	Battery sense divider
16	**Status LEDs**	LED + R	0603, Red/Green + 1kΩ	4	0.02	Charge/power status
17	**Power Switch**	SPDT Switch	JS102011 SAQN	1	0.3	Battery disconnect

10.2 Milestones

To ensure a successful project, remaining organized is key. Creating a guideline for the project will enable a strong dynamic among members and contribute to overall success. The project was kicked off by group selection and project topic discussion. The schedule below presents our plan for Senior Design 1:

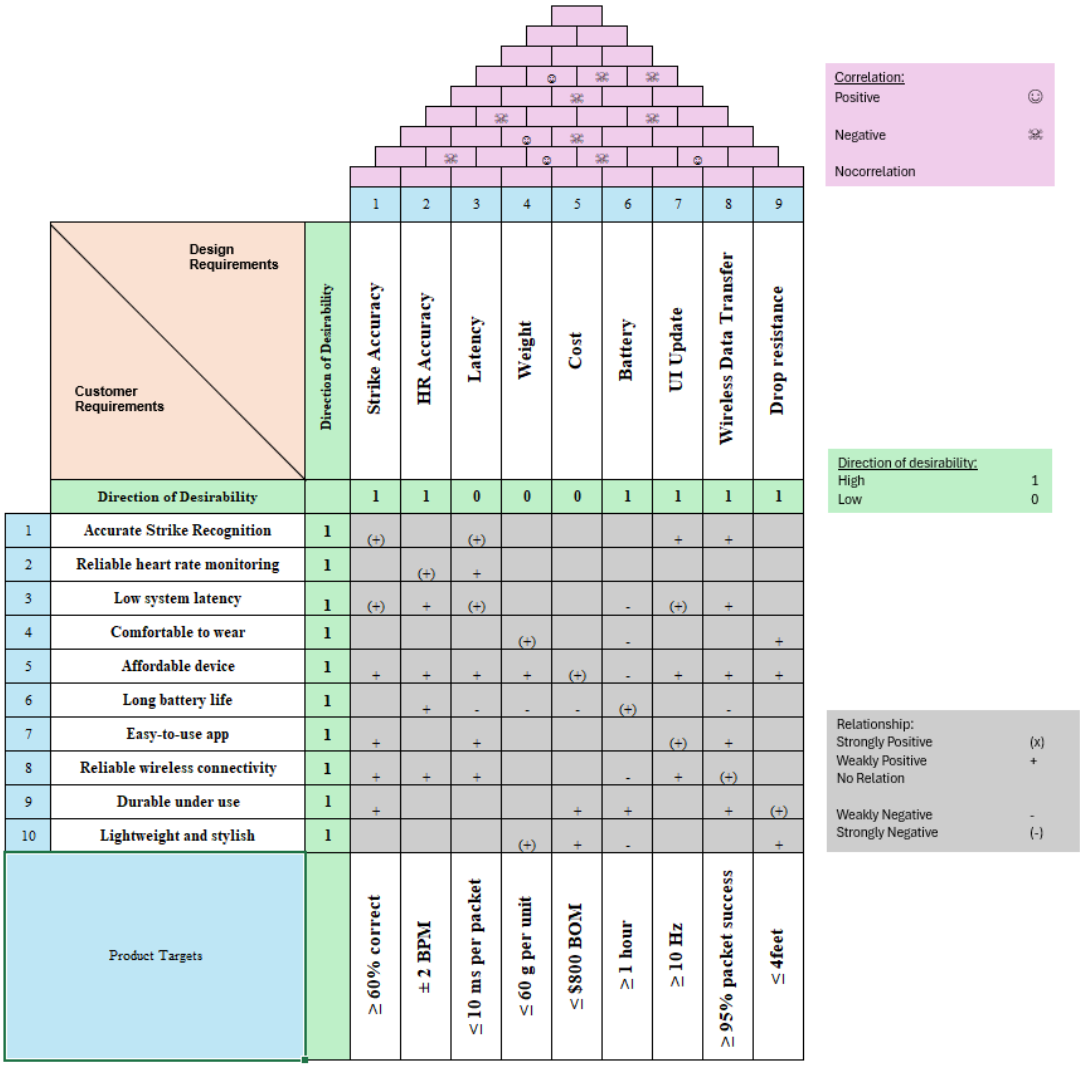
SD1 Project Schedule				
Task:	Start:	End:	Assigned Due Date:	Description:
Team formation	5/07/2025	6/07/2025	08/19/2025	Team: Landon Wright Benjamin Bolton, Shem Fils-Aime, Timothy Azinord
Brainstorming	6/07/2025	8/12/2025	8/12/2025	Researched multiple ideas for our project and elected a final one.
Initial Document	8/28/2025	9/04/2025	9/05/2025	Complete the Divide and Conquer Assignment to get a synopsis of our project.
Sponsor meeting	9/08/2025	9/08/2025	9/08/2025	Discuss ideas and get feedback on potential improvements.
30-page milestone	9/10/2025	9/30/205	9/30/205	Begin writing SD1 report
Component Selection	9/10/2025	10/20/2025	10/20/2025	Complete a BOM of all the necessary parts of the project
60-page milestone	9/10/2025	10/31/2025	10/31/2025	Hit the halfway point for our SD1 Report

First PCB Design/ Breakout Boards	9/10/2025	11/03/2025	11/03/2025	Design our first PCB for our sensor and create break out boards to begin testing
120-page milestone	9/10/2025	11/25/2025	11/25/2025	Produce the final Report for SD1

Considering planning for Senior Design 2 allows us to successfully carry out our design. At this stage of our project, we have a placeholder (TBD) allowing for flexibility till we get further down the line in Senior Design 1

SD2 Project Assembly				
Task:	Start:	End:	Assigned Due Date:	Description:
System Design	1/06/2025	1/20/2025	TBD	Complete system design and finish the overall schematic for the project
System Testing	1/20/2025	1/27/2025	TBD	Test each section of our project individually and check for compatibility.
Completed PCB Design	TBD	TBD	TBD	Finish the PCB and ensure robustness
PCB Test/Optimize	TBD	TBD	TBD	Continue testing for any more improvements.
Prototype Completion	TBD	TBD	TBD	Have a final working Prototype for SD2

10.3 House of Quality



	Design Requirements / Customer Requirements	Direction of Desirability	1 Strike Accuracy	2 HR Accuracy	3 Latency	4 Weight	5 Cost	6 Battery	7 UI Update	8 Wireless Data Transfer	9 Drop resistance
	Direction of Desirability		1	1	0	0	0	1	1	1	1
1	Accurate Strike Recognition	1	(+)		(+)				+	+	
2	Reliable heart rate monitoring	1		(+)	+						
3	Low system latency	1	(+)	+	(+)			-	(+)	+	
4	Comfortable to wear	1				(+)		-			+
5	Affordable device	1	+	+	+	+	(+)	-	+	+	+
6	Long battery life	1		+	-	-	-	(+)		-	
7	Easy-to-use app	1	+		+				(+)	+	
8	Reliable wireless connectivity	1	+	+	+			-	+	(+)	
9	Durable under use	1	+				+	+		+	(+)
10	Lightweight and stylish	1				(+)	+	-			+
	Product Targets		≥ 60% correct	± 2 BPM	≤ 10 ms per packet	≤ 60 g per unit	≤ $800 BOM	≥ 1 hour	≥ 10 Hz	≥ 95% packet success	≤ 4feet

Chapter 11 - Conclusion

11.1 Closing Remarks

In conclusion, the Strike Sense project represents a meaningful step forward in how athletes can understand and improve their striking performance. As outlined

in the executive summary, our goal was to move beyond simple punch counting and create a system that captures the flow, efficiency, and physical demands of real martial arts training. Through a combination of two 9 DoF IMUs, heart rate monitoring, and advanced software pipelines, we created a platform that brings together biomechanics, physiology, and intelligent analysis into a single wearable system.

Modern combat sports have an emphasis on the precision, energy efficiency and tactical sequencing that can't be captured on a simple screen. Our sensor addresses the gap by generating high resolution orientation and physiological data. Which is usable when it comes to giving athletes feedback that reflects performance rather than basic metrics. The project embraces a more holistic approach which is essential due to the complexity of human movement. By merging these domains the sensor establishes a new standard for wearable technology.

The hardware and software components developed throughout this project were designed to work together in a coordinated way.The 9 DOF IMUs we've selected provide high fidelity motion data through quaternion based tracking, offering stable orientation information across a wide range of strikes and impact conditions. Traditional methods like the euler based representation duffer from gimbal lock and cumulative drift. We utilize quaternions and sensor fusion algorithms so that we can have a robust representation of joint and limb level motion. These considerations were important because in martial arts applications stability is the key.

At the same time, the MAX30102 heart rate module provides cardiovascular insight that helps frame the athlete's performance within the context of effort, fatigue, and recovery. While the IMUs capture how the athletes would move the heart rate sensor allows it to see how the athlete responds internally. These streams form the backbone of our data pipeline and enable the real time classification, sequence detection, and post session analysis that distinguish Strike Sense from traditional punch trackers. The hardware pipeline was necessary and required careful attention to sampling rates and noise reduction and power constraints. Considering these ensured that machine learning systems could rely on stable and consistent input without loss.

The intelligent layer built around this hardware pushes the project even further. By incorporating transformer based models, LSTM architectures, and future expansion paths involving AlphaPose and digital twin visualization, the system is not just capturing what strikes occur, but also how and when they unfold. This

opens the door to coaching that is grounded in patterns, tendencies, power generation, and the physiological cost of each combination. The result is a tool that can guide fighters toward more informed and efficient training sessions while also encouraging continual self evaluation and growth.

Sequence recognition supports real coaching applications such as detecting fatigue. Machine learning also enables personalized athlete profiles. It will enable users to track long-term progress and suggestive adaptive measurements that can estimate potential injury risk and technique breakdown due to fatigue. These show that the strike sense stands out from the crowd for athlete insight.

This document provides a full account of the design process, including theoretical foundations, hardware analysis, software architecture, system constraints, and administrative planning. The consistent effort put forward by the Strike Sense team demonstrates our commitment to solving a well defined problem with a thoughtful engineering solution. As we transition into further development and testing, the work accomplished here establishes a strong foundation for a fully integrated and reliable performance analysis system. The insights gained throughout this process will support not only the final stages of Strike Sense, but also future innovations in wearable movement technology.

11.2 Plan for Senior Design 2

11.2.1 Hardware Design and PCB Development

11.2.2 Four-Layer PCB Research and Implementation

We will research and design a **4-layer PCB stackup** to improve signal integrity and noise performance. The expected layer structure is:

- Top: Components + high-speed signals
- Layer 2: Solid ground plane
- Layer 3: Power distribution (1.8 V, 3.3 V, battery rails)
- Bottom: Secondary routing + low-speed signals

Control goals include reduced EMI, improved return current paths, cleaner I²C/SPI communication, and minimized analog noise for the IMU and heart-rate sensor.

11.2.3 Datasheet-Guided Routing Practices

Each component (ESP32, ICM-20948, LDOs, TPS63070, MAX86150, BSS138, etc.) provides strict layout requirements. In SD2 we will implement:

- Trace width and spacing per datasheet recommendations

• Via placement rules to prevent ground loop formation
• Symmetric capacitor placement on LDO/buck-boost input/output
• Proper inductor orientation to minimize magnetic coupling
• Shielding analog traces from digital switching lines

Datasheet compliance will be verified for every critical component.

11.2.4 Decoupling and Noise Mitigation Strategy

We will finalize a complete decoupling network:
• One 0.1 µF ceramic capacitor at each power pin
• Bulk capacitors sized according to regulator startup transient behavior
• Local LC/RC filters where required by sensors
• Star-routing for high-current rails to prevent shared impedance coupling
• Ground plane segmentation only where necessary (IMU analog region)

EMI and ripple behavior will be validated using oscilloscope measurements on the final PCB.

11.2.5 Impedance Control and High-Speed Routing

Although our system is low-to-moderate speed, certain lines (ESP32 USB, SPI, sensor buses) require care:
• Target ~50 Ω single-ended trace impedance
• Length-matching for SPI clock/data where needed
• Avoiding discontinuities at vias
• Ensuring short, direct connections for crystal and RF components

These practices ensure stable operation and prevent intermittent communication errors.

11.2.6 Miniaturization

One major SD2 objective is shrinking the system footprint to approach smartwatch-class size. We will:
• Replace breakout modules with bare ICs
• Optimize placement around a central battery pocket
• Use smaller SMD packages (0402/0201) where routing allows
• Evaluate flex-PCB or segmented PCBs if needed

The final design will aim for a compact, ergonomic, and mechanically robust form factor.

11.2.7 Wearable System and Materials Research

To improve user comfort and practicality, SD2 will include research into:
• Elastic and breathable straps for prolonged wear
• Soft-material housings that reduce pressure on the wrist/forearm
• Sweat-resistant and washable casing materials
• 3D-printed or injection-molded enclosures
• Mounting geometry that aligns the IMU consistently with the limb

The goal is to produce a device that is not only functional, but also **comfortable, stable, and safe** for athletic use.

11.2.8 Sensor Calibration and Advanced Processing

11.2.8.1 Calibration Pipeline

Each sensing subsystem will undergo complete calibration:
• IMU: accelerometer bias, gyro drift, soft-iron/hard-iron calibration, alignment
• High-g accelerometer: baseline noise, offset, scaling factor
• Optical heart-rate sensor: dark current, ambient light rejection
• Level-shifter and logic bus validation under load and motion

Calibration matrices and offsets will be stored in firmware for runtime correction.

11.2.8.2 Quaternion → Actionable Motion Data

We will implement algorithms to convert IMU quaternion output into:
• Strike orientation
• Arm velocity profile
• Acceleration peaks
• Punch type classification

This includes building a dataset of recorded punches using both IMU and high-g accelerometer readings.

11.2.8.3 Machine Learning Model Development

A supervised ML pipeline will be created using Python, TensorFlow, and/or Scikit-Learn:
• Collect training data from multiple users
• Label punches (jab, cross, hook, uppercut, block)
• Extract features from both IMU and high-g accelerometer
• Train a model (SVM, LSTM, 1D-CNN, etc.)
• Deploy inference model onto ESP32 or backend server

The ML component forms the core of StrikeSense’s intelligent feedback system.

11.3 Software, Dashboard, and Computer Vision

11.3.1 Backend/API Development

We will build a backend capable of:

- Receiving real-time strike packets via Wi-Fi/BLE
- Storing user sessions in a database
- Running cloud-based ML inference (if not on-device)
- Supporting history, analytics, and coaching feedback

11.3.2 Frontend Dashboard (StrikeSense App/Web UI)

The dashboard will include:

- Real-time strike visualization
- Force magnitude graphs
- Punch classification results
- Historical analytics and training summaries
- Heart-rate syncing for conditioning metrics

The user interface will be responsive and mobile-friendly.

11.3.3 Computer Vision Research

To enhance classification accuracy or enable hybrid training, SD2 will include optional CV research:

- Capture video synchronized with IMU data
- Use pose-estimation libraries (MediaPipe, OpenPose)
- Compare CV-based strike labels vs IMU classifier accuracy
- Evaluate whether CV is integrated or used only for dataset labeling

11.3.4 Code Optimization & Power Efficiency

We will streamline the embedded firmware by:

- Removing redundant sensor polling
- Optimizing I²C/SPI timing
- Using sleep modes when inactive
- Reducing floating-point operations where possible

The goal is to extend battery life and improve responsiveness.

11.4 Final Integration and Validation

In the final phase of SD2, we will:

- Assemble the final 4-layer PCB

- Validate power rails (noise, stability, dropout, ripple)
- Test all communication protocols at full speed
- Evaluate heat, battery life, and mechanical durability
- Run full-system tests with athletes performing real strikes
- Conduct user testing for comfort and wearability

This ensures the device is ready for demo, competition, and eventual commercialization.

Chapter 12: Appendices

Appendix A: References

[1] Analog Devices. (n.d.). *Speed up Li-ion battery charging and reduce heat with a switching PowerPath manager.* Analog Devices. https://www.analog.com/en/resources/technical-articles/speed-up-li-ion-battery-charging-and-reduce-heat.html

[2] Battery Power Tips. (2023). *Difference between lithium-ion and lithium-polymer batteries (FAQ).* Battery Power Tips. https://www.batterypowertips.com/difference-between-lithium-ion-lithium-polymer-batteries-faq/

[3] Battery University. (n.d.). *How to prolong lithium-based batteries (BU-808).* Battery University. https://batteryuniversity.com/article/bu-808-how-to-prolong-lithium-based-batteries

[4] Bosch Sensortec GmbH. (2022). *BMM150 geomagnetic sensor: Data sheet (BST-BMM150-DS001).* Bosch Sensortec. https://www.bosch-sensortec.com/media/boschsensortec/downloads/datasheets/bst-bmm150-ds001.pdf

[5] DNK Power. (2022). *IEC 62133 compliance testing for lithium battery packs.* DNK Power. https://www.dnkpower.com/battery-iec-62133

[6] Enix Power Solutions. (2023). *IEC 62133 and IEC 62619 certification overview.* Enix Power Solutions. https://www.enix-power-solutions.com/iec62133-iec62619-certification

[7] Espressif Systems. (2023). *ESP32-S3 overview.* Espressif Systems. https://www.espressif.com/en/products/socs/esp32-s3

[8] Espressif Systems. (2023). *ESP32-S3 series datasheet (version 2.0).* Espressif Systems. https://www.espressif.com/documentation/esp32-s3_datasheet_en.pdf

[9] Espressif Systems. (2023). *ESP32-S3 technical reference manual.* Espressif Systems. https://www.espressif.com/documentation/esp32-s3_technical_reference_manual_en.pdf

[10] Espressif Systems. (2023). *ESP32-S3-DevKitC-1 hardware design files [GitHub repository].* GitHub. https://github.com/espressif/esp-idf-hw-design

[11] European Commission. (2011). *Directive 2011/65/EU on the restriction of hazardous substances (RoHS) in electrical and electronic equipment.* https://environment.ec.europa.eu/topics/waste-and-recycling/rohs-directive_en

[12] European Union. (2006). *Directive 2006/66/EC on batteries and accumulators and waste batteries and accumulators. Official Journal of the European Union.* https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=celex%3A32006L0066

[13] GlobalSpec. (n.d.). *Compasses and magnetometers information.* GlobalSpec. https://www.globalspec.com/learnmore/motion_controls/orientation_position_sensing/compasses_magnetometers

[14] International Electrotechnical Commission. (2017). *IEC 62133-2:2017 – Secondary cells and batteries containing alkaline or other non-acid electrolytes – Safety requirements for portable sealed secondary cells, and for batteries made from them, for use in portable applications – Part 2: Lithium systems.* IEC Webstore. https://webstore.iec.ch/publication/32662

[15] International Electrotechnical Commission. (2017). *IEC 62619 – Secondary lithium cells and batteries for use in industrial applications – Safety requirements.* IEC Webstore. https://webstore.iec.ch/publication/26180

[16] Intertek Testing Services. (2023). *UN 38.3 transport testing for lithium batteries.* Intertek. https://www.intertek.com/batteries/un-38-3-testing

[17] Large Battery. (2023). *Understanding LiPo battery voltage for optimal performance.* Large Battery. https://www.large-battery.com/blog/lipo-battery-voltage-guide/

[18] Large Battery Company. (2024). *Comparison of UN 38.3, IEC 62133 and UL standards for lithium batteries.* Large-Battery Blog. https://www.large-battery.com/blog/

[19] Li-Polymer Battery Company. (2023). *IEC 62133-2 certification of lithium polymer batteries.* Li-Polymer Battery Co. https://li-polymer-battery.com/whats-the-iec-62133-22017-certification-of-li-polymer-battery

[20] MEMSIC Inc. (2019). *MMC5983MA ultra-low noise, 3-axis magnetometer data sheet (Rev. A).* MEMSIC. https://media.digikey.com/pdf/Data%20Sheets/MEMSIC%20PDFs/MMC5983MA_RevA_4-3-19.pdf

[21] MEMSIC Inc. (2021). *MMC5983MA magnetometer data sheet (Rev. A).* MEMSIC. https://www.memsic.com/Public/Uploads/uploadfile/files/20220119/MMC5983MADatasheetRevA.pdf

[22] Microchip Technology Inc. (2016). *MCP73871 Li-Ion/Li-Polymer charge management controller datasheet (DS20002090C).* Microchip. https://ww1.microchip.com/downloads/en/DeviceDoc/20002090C.pdf

[23] Microchip Technology Inc. (2020). *MCP73871 data sheet (DS20002090F).* Microchip. https://ww1.microchip.com/downloads/aemDocuments/documents/APID/ProductDocuments/DataSheets/MCP73871-Data-Sheet-DS20002090F.pdf

[24] Microchip Technology Inc. (2021). *MCP73831 family data sheet (DS20001984H).* Microchip. https://ww1.microchip.com/downloads/aemDocuments/documents/APID/ProductDocuments/DataSheets/MCP73831-Family-Data-Sheet-DS20001984H.pdf

[25] Monolithic Power Systems. (n.d.). *Boost converters (step-up converter).* Monolithic Power Systems. https://www.monolithicpower.com/en/learning/mpscholar/power-electronics/dc-dc-converters/boost-converters

[26] Monolithic Power Systems. (n.d.). *Buck converters (step-down converter).* Monolithic Power Systems. https://www.monolithicpower.com/en/learning/mpscholar/power-electronics/dc-dc-converters/buck-converters

[27] Monolithic Power Systems. (n.d.). *Voltage regulator types and working principles.* Monolithic Power Systems. https://www.monolithicpower.com/en/learning/resources/voltage-regulator-types

[28] Monolithic Power Systems. (n.d.). *Magnetic sensors: Common types, key components, parameters, usage considerations, and applications.* Monolithic Power Systems. https://www.monolithicpower.com/en/learning/resources/magnetic-sensors-common-types-key-components-parameters-usage-considerations-and-applications

[29] Richtek Technology Corporation. (n.d.). *Comparing buck converter topologies.* Richtek Technology.

[30] Standardization Administration of China (SAC). (2014). *GB 31241-2014 – Safety requirements for lithium-ion cells and batteries used in portable electronic equipment.* https://www.sac.gov.cn

[31] Texas Instruments. (2019). *Buck regulator topologies for wide input/output voltage differentials.* Texas Instruments. https://www.ti.com/lit/wp/snva594a/snva594a.pdf

[32] Texas Instruments. (2022). *BQ24075 Li-ion linear charger with power path management data sheet.* Texas Instruments. https://www.ti.com/lit/ds/symlink/bq24075.pdf

[33] Texas Instruments. (2024). *Battery management system design guide.* Texas Instruments. https://www.ti.com/power-management/battery-management

[34] Texas Instruments. (n.d.). *Control-mode quick reference guide.* Texas Instruments. https://www.ti.com/lit/an/slyt710b/slyt710b.pdf

[35] Texas Instruments. (n.d.). *Understanding and applying current-mode control theory (SNVA555).* Texas Instruments. https://www.ti.com/lit/an/snva555/snva555.pdf

[36] TÜV SÜD. (2022). *Battery testing and certification for lithium-ion systems.* TÜV SÜD. https://www.tuvsud.com/en-us/services/product-certification/battery-testing

[37] Underwriters Laboratories. (2024). *UL 1642 – Standard for safety of lithium batteries.* UL Standards & Engagement. https://ul.com/services/battery-safety-testing

[38] Underwriters Laboratories. (2024). *UL 2054 – Standard for safety of household and commercial batteries.* UL Standards & Engagement. https://ul.com/services/ul-2054-battery-pack-safety-testing

[39] UL Solutions. (2024). *UL 94 flammability of plastic materials for parts in devices and appliances.* UL Solutions. https://ul.com/offerings/ul-94-flammability

[40] United Nations Economic Commission for Europe (UNECE). (2023). *Manual of tests and criteria, Part III, Sub-section 38.3 – Recommendations on the transport of dangerous goods: Lithium metal and lithium-ion batteries.* https://unece.org/manual-tests-and-criteria

[41] United States Department of Transportation. (2023). *Lithium battery air transport regulations (IATA PI 967).* Federal Aviation Administration Hazardous Materials Program. https://www.faa.gov/hazmat

[42] Winder, S. (2017). *Boost–buck converter.* In *Elsevier eBooks* (pp. 117–154). Elsevier. https://doi.org/10.1016/B978-0-08-100925-3.00007-0

[43] YoungWonks. (2023). *What is a magnetometer and how does it work?* YoungWonks. https://www.youngwonks.com/blog/What-is-a-Magnetometer-and-How-Does-It-Work

[44] Zhang, Y., Wang, X., & Chen, Z. (2023). *Evaluation of charging methods for lithium-ion batteries. Electronics, 12*(19), 4095. https://www.mdpi.com/2079-9292/12/19/4095

[45] All About Circuits. (2021). Regulator basics. All About Circuits. https://www.allaboutcircuits.com/technical-articles/voltage-regulator-basics

[46] Altium; Cadence. (2024). Best practices for PCB connector placement and selection [Blog posts]. Altium & Cadence. https://resources.altium.com

[47] Analog Devices. (2020). Voltage reference applications. Analog Devices. https://www.analog.com/en/resources/technical-articles/voltage-reference-applications.html

[48] Analog Devices. (2020). NMOS vs. PMOS LDO design guide. Analog Devices. *https://www.analog.com/en/resources/technical-articles/nmos-vs-pmos-ldo-design-guide.html*

[49] Analog Devices. (2021). Switching regulator noise considerations. Analog Devices. *https://www.analog.com/en/resources/technical-articles/switching-regulator-noise-considerations.html*

[50] Analog Devices. (2022). AN-1140: Power layout for mixed-signal systems. Analog Devices. *https://www.analog.com/media/en/technical-documentation/application-notes/AN-1140.pdf*

[51] Analog Devices. (2024). MAX86150 heart-rate sensor datasheet. Analog Devices. *https://www.analog.com/en/products/max86150.html*

[52] Arduino Forum. (2019). When to use a level shifter? Arduino Forum. *https://forum.arduino.cc/t/when-to-use-a-level-shifter/581707*

[53] Buck converter. (2025). Wikipedia. Retrieved October 30, 2025, from *https://en.wikipedia.org/wiki/Buck_converter*

[54] Buck–boost converter. (2025). Wikipedia. Retrieved October 30, 2025, from *https://en.wikipedia.org/wiki/Buck–boost_converter*

[55] Digi-Key Electronics. (2020). What is a voltage regulator? Maker.io. *https://www.digikey.com/en/maker/projects/what-is-a-voltage-regulator*

[56] Digi-Key Electronics. (2020). 78XX and 79XX regulator families. Digi-Key. *https://www.digikey.com/en/articles/understanding-78xx-79xx-voltage-regulator-families*

[57] Digi-Key Electronics. (2020). Logic level shifting basics. Digi-Key. *https://www.digikey.com/en/articles/logic-level-shifting-basics*

[58] EDN. (2017). What every engineer should know about buck-boost converters. EDN Network. *https://www.edn.com/what-every-engineer-should-know-about-buck-boost-converters*

[59] ElectricalTechnology.org. (2020). Buck converter design. ElectricalTechnology.org. *https://www.electricaltechnology.org/2020/05/buck-converter-basics.html*

[60] Electronics Notes. (2021). Series voltage regulation explained. Electronics Notes. *https://www.electronics-notes.com/articles/analogue_circuits/voltage-regulator/series-voltage-regulator.php*

[61] Electronics Stack Exchange. (2011). Difference between level shifter, regulator, and DC–DC converter. Stack Exchange. *https://electronics.stackexchange.com/questions/19795*

[62] Electronics Tutorials. (2020). Linear voltage regulator. Electronics Tutorials. https://www.electronics-tutorials.ws/blog/linear-voltage-regulator.html

[63] Espressif Systems. (2020). ESP32-WROOM-32 datasheet (Rev. 3). Espressif Systems. https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32_datasheet_en.pdf

[64] Instructables. (n.d.). Guide to using MAX30102 heart rate and oxygen sensor. Instructables. https://www.instructables.com/Guide-to-Using-MAX30102-Heart-Rate-and-Oxygen-Sens/

[65] InvenSense (TDK). (2024). ICM-20948 datasheet. TDK InvenSense. https://invensense.tdk.com/products/motion-tracking/9-axis/icm-20948/

[66] Maxim Integrated. (2018). MAX30105 optical heart-rate sensor datasheet. Maxim Integrated. https://www.analog.com/en/products/max30105.html

[67] Microchip Technology. (2016). MCP73831 Li-ion/Li-polymer charge management controller datasheet. Microchip. https://ww1.microchip.com/downloads/en/DeviceDoc/20001984H.pdf

[68] Microchip Technology. (2022). LDO fundamentals. Microchip. https://www.microchip.com/en-us/solutions/power-management/ldo-regulators/fundamentals

[69] MinebeaMitsumi. (2022). Shunt regulation overview. MinebeaMitsumi. https://product.minebeamitsumi.com/en/technical/techguide/power/shunt-regulator.html

[70] MinebeaMitsumi. (2022). Types of linear regulators. MinebeaMitsumi. https://product.minebeamitsumi.com/en/technical/techguide/power/linear-regulator.html

[71] ResearchGate. (2023). Power management in wearables. ResearchGate. https://www.researchgate.net/publication/373743589_Power_Management_in_Wearables

[72] SparkFun. (2017). Accelerometer basics. Learn.SparkFun.com. https://learn.sparkfun.com/tutorials/accelerometer-basics

[73] SunFounder. (n.d.). MAX30102 sensor basics. SunFounder. https://docs.sunfounder.com/projects/ultimate-sensor-kit/en/latest/components_basic/15-component_max30102.html

[74] Texas Instruments. (2017). TLV700 200-mA low dropout regulator datasheet (Rev. F). Texas Instruments. https://www.ti.com/lit/ds/symlink/tlv700.pdf

[75] Texas Instruments. (2018). PCA9306 bidirectional level translator for I²C bus (Rev. C). Texas Instruments. https://www.ti.com/lit/ds/symlink/pca9306.pdf

[76] Texas Instruments. (2018). TPS63070 2A high efficiency buck-boost converter datasheet (Rev. E). Texas Instruments. https://www.ti.com/lit/ds/symlink/tps63070.pdf

[77] Texas Instruments. (2019). Power supply layout guidelines (SLVA763). Texas Instruments. https://www.ti.com/lit/an/slva763/slva763.pdf

[78] Texas Instruments. (2021). Boost converter basics (SNVA559C). Texas Instruments. https://www.ti.com/lit/an/snva559c/snva559c.pdf

[79] Texas Instruments. (2021). Low-dropout regulators (LDOs): TI training series. Texas Instruments. https://training.ti.com/ti-power-supply-design-seminar-ldo-fundamentals

[80] Texas Instruments. (2021). Power management fundamentals. Texas Instruments. https://www.ti.com/power-management

[81] Texas Instruments. (2021). What is an LDO? [Video]. Texas Instruments. https://www.youtube.com/watch?v=iw4z07oJGq0

[82] Texas Instruments. (2022). Line and load regulation in voltage regulators (Application Note). Texas Instruments. https://www.ti.com/lit/an/slva104.pdf

[83] Texas Instruments. (2023). Fundamentals of switching regulators. Texas Instruments. https://www.ti.com/lit/slup101.pdf

[84] Texas Instruments. (2024). Battery-powered system design considerations. Texas Instruments. https://www.ti.com/lit/wp/slva958/slva958.pdf

[85] The Signal Path. (2020). Switching converter overview. The Signal Path. https://www.youtube.com/watch?v=aK5A9cTbn2o

[86] Voltage regulator. (2025). Wikipedia. Retrieved October 30, 2025, from https://en.wikipedia.org/wiki/Voltage_regulator

[87] Wilcoxon Sensing. (2020). Understanding the accelerometer noise specification (TN33). Wilcoxon Sensing Technologies. https://www.wilcoxon.com/resources/TN33.pdf

[88] Wilcoxon Sensing. (2020). Noise integration practice for RMS estimation over bandwidth [Derived from TN33]. Wilcoxon Sensing Technologies. https://www.wilcoxon.com/resources/TN33.pdf

[89] YouTube. (2019). How to use a Zener diode [Video]. YouTube. https://www.youtube.com/watch?v=5vec422YMik

[90] YouTube. (2020). Series regulators [Video]. YouTube.
https://www.youtube.com/watch?v=J8obBcHviaY

[91] YouTube. (2020). Series regulator tutorial [Video]. YouTube.
https://www.youtube.com/watch?v=4sBQHYbItgY

[92] YouTube. (2020). What does a voltage regulator do? [Video]. YouTube.
https://www.youtube.com/watch?v=AX5SqSJu6R4

[93] Total Phase, "I²C vs SPI: Protocol Analyzer Differences and Similarities," 2021. [Online]. Available:
https://www.totalphase.com/blog/2021/07/i2c-vs-spi-protocol-analyzers-differences-and-similarities/

[94] Prodigy Technovations, "I²C vs SPI Comparison," 2022. [Online]. Available:
https://www.prodigytechno.com/i2c-vs-spi

[95] YouTube, "I²C Explained Simply," 2023. [Online]. Available:
https://www.youtube.com/watch?v=j9yx8LOslng

[96] YouTube, "SPI vs I²C: Key Differences," 2023. [Online]. Available:
https://www.youtube.com/watch?v=IyGwvGzrqp8&t=270s

Appendix B: Copyright Permissions

LW Landon Wright
To: ConnectionsHRServices@analog.com
Reply Reply all Forward ···
Tue 11/25/2025 10:01 AM

Hello,

My name is Landon Wright, and I am a Computer Engineering student at the University of Central Florida. I am writing to request permission to adapt figures 3 and 4 from the Analog Devices article titled "Understanding the Architecture of the Bluetooth Low Energy Stack" by Erick Reyes, Mary Grace Legaspi, and Eric Peña (published December 11, 2024) for inclusion in my senior design report.

My team's project utilizes Bluetooth Low Energy, and these figures provide valuable insight into the technology's architecture that would greatly enhance our technical breakdown and explanation within the report.

Thank you very much for considering my request. Please let me know if you need any additional information.

Best regards,

Landon Wright

Appendix C: Software Code

BLE Test Script (Server)

```
#include <Arduino.h>
#include <NimBLEDevice.h>

#define SERVICE_UUID        "6E400001-B5A3-F393-E0A9-E50E24DCCA9E"
#define CHARACTERISTIC_UUID "6E400003-B5A3-F393-E0A9-E50E24DCCA9E"

NimBLEServer* pServer;
```

```
NimBLECharacteristic* pIMUCharacteristic;
bool deviceConnected = false;

class ServerCallbacks : public NimBLEServerCallbacks {
   void onConnect(NimBLEServer* pServer, NimBLEConnInfo& connInfo) override {
       deviceConnected = true;
   }
   void onDisconnect(NimBLEServer* pServer, NimBLEConnInfo& connInfo, int
reason) override {
       deviceConnected = false;
       NimBLEDevice::startAdvertising();
   }
};

void setup() {

   Serial.begin(115200);
   NimBLEDevice::init("IMU-Server");
   pServer = NimBLEDevice::createServer();
   pServer->setCallbacks(new ServerCallbacks());
   NimBLEService* pService = pServer->createService(SERVICE_UUID);
   pIMUCharacteristic = pService->createCharacteristic(
       CHARACTERISTIC_UUID,
       NIMBLE_PROPERTY::NOTIFY | NIMBLE_PROPERTY::READ);
   pIMUCharacteristic->setValue("Waiting for data...");
   pService->start();
   NimBLEAdvertising* pAdvertising = NimBLEDevice::getAdvertising();
   pAdvertising->addServiceUUID(SERVICE_UUID);
   //pAdvertising->setScanResponseData();
   pAdvertising->start();
}

void loop() {
   // Example dummy IMU sample
   if (deviceConnected) {
       String sample = String(millis()); // Replace with IMU data
       pIMUCharacteristic->setValue(sample.c_str());
       pIMUCharacteristic->notify();
       delay(10); // ~100Hz;     } else {
       delay(100);
   }
}
```

Raw IMU data test script

```
#include <Arduino.h>
#include <Wire.h>

#include <Adafruit_ICM20X.h>
#include <Adafruit_ICM20948.h>
#include <Adafruit_Sensor.h>

#include <Fusion.h>

// Built-in LED
#define LED_PIN 4

// I2C pins
#define I2C_SDA_PIN 8
#define I2C_SCL_PIN 9

Adafruit_ICM20948 icm20948;

sensors_event_t accel;
sensors_event_t gyro;
sensors_event_t mag;
sensors_event_t temp;

// Fusion AHRS
FusionAhrs ahrs;
unsigned long lastMillis = 0;

void setup() {
 pinMode(LED_PIN, OUTPUT);
 digitalWrite(LED_PIN, LOW);

 Serial.begin(115200);
 delay(1500);
 Serial.println("=== ESP32-S3 BOOT ===");

 Wire.begin(I2C_SDA_PIN, I2C_SCL_PIN);
 Wire.setClock(400000);

 Serial.println("Initializing IMU...");
```

```
if (!icm20948.begin_I2C()) {
  Serial.println("IMU NOT FOUND!");
  while (1) {
    digitalWrite(LED_PIN, HIGH); delay(200);
    digitalWrite(LED_PIN, LOW); delay(200);
  }
}

Serial.println("ICM-20948 FOUND.");

// ------------------------------------
//  Initialize Fusion
// ------------------------------------
FusionAhrsInitialise(&ahrs);

FusionAhrsSettings settings;
settings.convention = FusionConventionNwu;
settings.gain = 0.5f;
settings.gyroscopeRange = 2000.0f;    // deg/sec — must match IMU config
settings.accelerationRejection = 10.0f;
settings.magneticRejection = 10.0f;
settings.recoveryTriggerPeriod = 25;   // 5 sec at 5 Hz
FusionAhrsSetSettings(&ahrs, &settings);

lastMillis = millis();


}

void loop() {
// LED heartbeat
digitalWrite(LED_PIN, HIGH);
delay(50);
digitalWrite(LED_PIN, LOW);

// Read IMU
icm20948.getEvent(&accel, &gyro, &mag, &temp);

// Compute sample period
unsigned long now = millis();
float deltaTime = (now - lastMillis) / 1000.0f;
```

```
  lastMillis = now;

  // Convert gyro rad/s -> deg/s
  FusionVector gyroscope = {
    gyro.gyro.x * 180.0f / PI,
    gyro.gyro.y * 180.0f / PI,
    gyro.gyro.z * 180.0f / PI
  };

  // Accelerometer m/s^2
  FusionVector accelerometer = {
    accel.acceleration.x,
    accel.acceleration.y,
    accel.acceleration.z
  };

  // Magnetometer uT
  FusionVector magnetometer = {
    mag.magnetic.x,
    mag.magnetic.y,
    mag.magnetic.z
  };

  // Update AHRS
  FusionAhrsUpdate(&ahrs, gyroscope, accelerometer, magnetometer, deltaTime);

  // Convert output quaternion -> Euler angles
  FusionEuler euler = FusionQuaternionToEuler(FusionAhrsGetQuaternion(&ahrs));

  Serial.print("Pitch: "); Serial.print(euler.angle.pitch, 2);
  Serial.print("  Roll: "); Serial.print(euler.angle.roll, 2);
  Serial.print("  Yaw: "); Serial.println(euler.angle.yaw, 2);

  delay(150);
}
```

Sensor Fusion Test Script

```
#include <Arduino.h>
#include <Wire.h>

#include <Adafruit_ICM20X.h>
```

```
#include <Adafruit_ICM20948.h>
#include <Adafruit_Sensor.h>

#include <Fusion.h>

// Built-in LED
#define LED_PIN 4

// I2C pins
#define I2C_SDA_PIN 8
#define I2C_SCL_PIN 9

Adafruit_ICM20948 icm20948;

sensors_event_t accel;
sensors_event_t gyro;
sensors_event_t mag;
sensors_event_t temp;

// Fusion AHRS
FusionAhrs ahrs;
unsigned long lastMillis = 0;

void setup() {
 pinMode(LED_PIN, OUTPUT);
 digitalWrite(LED_PIN, LOW);

 Serial.begin(115200);
 delay(1500);
 Serial.println("=== ESP32-S3 BOOT ===");

 Wire.begin(I2C_SDA_PIN, I2C_SCL_PIN);
 Wire.setClock(400000);

 Serial.println("Initializing IMU...");

 if (!icm20948.begin_I2C()) {
   Serial.println("IMU NOT FOUND!");
   while (1) {
     digitalWrite(LED_PIN, HIGH); delay(200);
     digitalWrite(LED_PIN, LOW); delay(200);
   }
```

```
}

Serial.println("ICM-20948 FOUND.");

// ------------------------------------
//  Initialize Fusion
// ------------------------------------
FusionAhrsInitialise(&ahrs);

FusionAhrsSettings settings;
settings.convention = FusionConventionNwu;
settings.gain = 0.5f;
settings.gyroscopeRange = 2000.0f;    // deg/sec — must match your IMU config
settings.accelerationRejection = 10.0f;
settings.magneticRejection = 10.0f;
settings.recoveryTriggerPeriod = 25;   // 5 sec at 5 Hz
FusionAhrsSetSettings(&ahrs, &settings);

lastMillis = millis();



}

void loop() {
// LED heartbeat
digitalWrite(LED_PIN, HIGH);
delay(50);
digitalWrite(LED_PIN, LOW);

// Read IMU
icm20948.getEvent(&accel, &gyro, &mag, &temp);

// Compute sample period
unsigned long now = millis();
float deltaTime = (now - lastMillis) / 1000.0f;
lastMillis = now;

// Convert gyro rad/s -> deg/s
FusionVector gyroscope = {
  gyro.gyro.x * 180.0f / PI,
  gyro.gyro.y * 180.0f / PI,
  gyro.gyro.z * 180.0f / PI
```

```
};

// Accelerometer m/s^2
FusionVector accelerometer = {
  accel.acceleration.x,
  accel.acceleration.y,
  accel.acceleration.z
};

// Magnetometer uT
FusionVector magnetometer = {
  mag.magnetic.x,
  mag.magnetic.y,
  mag.magnetic.z
};

// Update AHRS
FusionAhrsUpdate(&ahrs, gyroscope, accelerometer, magnetometer, deltaTime);

// Convert output quaternion -> Euler angles
FusionEuler euler = FusionQuaternionToEuler(FusionAhrsGetQuaternion(&ahrs));

Serial.print("Pitch: "); Serial.print(euler.angle.pitch, 2);
Serial.print("  Roll: "); Serial.print(euler.angle.roll, 2);
Serial.print("  Yaw: "); Serial.println(euler.angle.yaw, 2);

delay(150);
}
```

Completed Strike Sense Module

```
// Basic
#include <Arduino.h>
#include <Wire.h>

// ICM20948 Libraries
#include <Adafruit_ICM20X.h>
#include <Adafruit_ICM20948.h>
#include <Adafruit_Sensor.h>

// Sensor Fusion Library
#include <Fusion.h>
```

```
// BLE Library
#include <NimBLEDevice.h>

// BLE UUIDs (UART-like)
#define SERVICE_UUID        "6E400001-B5A3-F393-E0A9-E50E24DCCA9E"
#define CHARACTERISTIC_UUID "6E400003-B5A3-F393-E0A9-E50E24DCCA9E"

// Testing LED
#define LED_PIN 4

// ICM20948 I2C pins
#define I2C_SDA_PIN 8
#define I2C_SCL_PIN 9

Adafruit_ICM20948 icm20948;

sensors_event_t accel;
sensors_event_t gyro;
sensors_event_t mag;
sensors_event_t temp;

// Fusion AHRS
FusionAhrs ahrs;
uint32_t lastMillis = 0;

// BLE
NimBLEServer* pServer = nullptr;
NimBLECharacteristic* pIMUCharacteristic = nullptr;
bool deviceConnected = false;

// Packet we send over BLE
struct FusionData {
 float pitch;
 float roll;
 float yaw;
} __attribute__((packed));  // make sure no padding is added

// BLE server callbacks
class ServerCallbacks : public NimBLEServerCallbacks {
   void onConnect(NimBLEServer* pServer, NimBLEConnInfo& connInfo) override {
       deviceConnected = true;
```

```
        Serial.println("Client connected.");
    }
    void onDisconnect(NimBLEServer* pServer, NimBLEConnInfo& connInfo, int
reason) override {
        deviceConnected = false;
        Serial.println("Client disconnected, restarting advertising.");
        NimBLEDevice::startAdvertising();
    }
};

void setup() {
 pinMode(LED_PIN, OUTPUT);
 digitalWrite(LED_PIN, LOW);

 Serial.begin(115200);
 delay(1500);
 Serial.println("=== ESP32-S3 BOOT ===");

 // I2C + IMU init
 Wire.begin(I2C_SDA_PIN, I2C_SCL_PIN);
 Wire.setClock(400000);

 Serial.println("Initializing IMU...");

 if (!icm20948.begin_I2C()) {
   Serial.println("IMU NOT FOUND!");
   while (1) {
     digitalWrite(LED_PIN, HIGH); delay(200);
     digitalWrite(LED_PIN, LOW); delay(200);
   }
 }

 Serial.println("ICM-20948 FOUND.");

 // ------------------------------------
 //  Initialize Fusion
 // ------------------------------------
 FusionAhrsInitialise(&ahrs);

 FusionAhrsSettings settings;
 settings.convention = FusionConventionNwu;
 settings.gain = 0.5f;
```

```
 settings.gyroscopeRange = 2000.0f;    // deg/sec — match IMU config if you
change it
 settings.accelerationRejection = 10.0f;
 settings.magneticRejection = 10.0f;
 settings.recoveryTriggerPeriod = 25;  // 5 sec at 5 Hz (we'll be ~50-100 Hz
but it's fine)
 FusionAhrsSetSettings(&ahrs, &settings);

 lastMillis = millis();

 // ------------------------------------
 //  Initialize BLE (NimBLE)
 // ------------------------------------
 NimBLEDevice::init("StrikeSense-IMU-1");  // Device name you will see on
PC/phone
 NimBLEDevice::setPower(ESP_PWR_LVL_P9);

 pServer = NimBLEDevice::createServer();
 pServer->setCallbacks(new ServerCallbacks());

 NimBLEService* pService = pServer->createService(SERVICE_UUID);

 pIMUCharacteristic = pService->createCharacteristic(
     CHARACTERISTIC_UUID,
     NIMBLE_PROPERTY::NOTIFY | NIMBLE_PROPERTY::READ
 );

 // Initial value
 FusionData zero = {0.0f, 0.0f, 0.0f};
 pIMUCharacteristic->setValue((uint8_t*)&zero, sizeof(zero));

 pService->start();

 NimBLEAdvertising *pAdvertising = NimBLEDevice::getAdvertising();
 //pAdvertising->addServiceUUID(SERVICE_UUID);

 pAdvertising->setName("StrikeSense-IMU-1");
 //pAdvertising->enableScanResponse(true);
 pAdvertising->start();

 Serial.println("BLE advertising started.");
}
```

```
void loop() {
 // Target sample period (ms)
 const uint32_t targetIntervalMs = 20;   // ~50 Hz

 uint32_t now = millis();
 if (now - lastMillis < targetIntervalMs) {
   // Not time for a new sample yet
   return;
 }

 float deltaTime = (now - lastMillis) / 1000.0f; // seconds
 lastMillis = now;

 // LED heartbeat (short blink)
 digitalWrite(LED_PIN, HIGH);
 delayMicroseconds(1000);
 digitalWrite(LED_PIN, LOW);

 // Read IMU
 icm20948.getEvent(&accel, &gyro, &mag, &temp);

 // Convert gyro rad/s -> deg/s
 FusionVector gyroscope = {
   gyro.gyro.x * 180.0f / PI,
   gyro.gyro.y * 180.0f / PI,
   gyro.gyro.z * 180.0f / PI
 };

 // Accelerometer m/s^2
 FusionVector accelerometer = {
   accel.acceleration.x,
   accel.acceleration.y,
   accel.acceleration.z
 };

 // Magnetometer uT
 FusionVector magnetometer = {
   mag.magnetic.x,
   mag.magnetic.y,
   mag.magnetic.z
 };
```

```
// Update AHRS
FusionAhrsUpdate(&ahrs, gyroscope, accelerometer, magnetometer, deltaTime);

// Convert output quaternion -> Euler angles
FusionEuler euler = FusionQuaternionToEuler(FusionAhrsGetQuaternion(&ahrs));

FusionData fusionData = {
  euler.angle.pitch,
  euler.angle.roll,
  euler.angle.yaw
};

//print occasionally for sanity check
static uint32_t printCounter = 0;
if (++printCounter >= 10) {  // every ~10 samples
  printCounter = 0;
  Serial.print("Pitch: "); Serial.print(fusionData.pitch, 2);
  Serial.print("  Roll: "); Serial.print(fusionData.roll, 2);
  Serial.print("  Yaw: "); Serial.println(fusionData.yaw, 2);
}

// Send over BLE if connected
if (deviceConnected) {
  pIMUCharacteristic->setValue((uint8_t*)&fusionData, sizeof(fusionData));
  pIMUCharacteristic->notify();
}
}
```

Reading Data from IMU and Moving to InfluxDB

```python
# InfluxDB imports copied from sensor_to_influx.py
from influxdb_client import InfluxDBClient, Point, WritePrecision
from influxdb_client.client.write_api import SYNCHRONOUS

# ----------------------------------------------------
# CONFIG
# ----------------------------------------------------

# Match what you used in NimBLEDevice::init()
TARGET_NAME1 = "StrikeSense-IMU-1"
TARGET_NAME2 = "StrikeSense-IMU-2"
TARGET_NAMES = [TARGET_NAME1, TARGET_NAME2]

# Must match CHARACTERISTIC_UUID in your ESP32 sketch
IMU_CHAR_UUID = "6E400003-B5A3-F393-E0A9-E50E24DCCA9E"
NUS_SERVICE_UUID = "6e400001-b5a3-f393-e0a9-e50e24dcca9e"

# InfluxDB config (same as sensor_to_influx.py, except bucket names)
INFLUX_URL   = "http://localhost:8086"
INFLUX_TOKEN = "rkoZmDCljY2unObMizpyM5vy3wkO9sKK8JCC-nQQcflDbr_WUBojRsnVXK5W95VWppGtA5cXe239UM8MlK2r3w=="
INFLUX_ORG   = "mugi"

BUCKET_IMU1 = "IMU1"
BUCKET_IMU2 = "IMU2"


# ----------------------------------------------------
# Scan helper
# ----------------------------------------------------

async def find_devices():
    """Scan for all target devices and return list of matching devices."""
    print(" Scanning for BLE devices...")
    devices = await BleakScanner.discover(timeout=10.0)

    found_devices = []

    if not devices:
        print("No BLE devices found at all.")
        return None

    print("\nFound devices:")
    for i, d in enumerate(devices):
        print(f"[{i}] name={d.name} | address={d.address}")

        # Try match by name
        if d.name in TARGET_NAMES:
            print(f' Found target: {d.name} @ {d.address}')
            found_devices.append(d)

    if not found_devices:
        print("\n Target devices not found.")
        return None

    print(f"\n Found {len(found_devices)} target device(s)")
    return found_devices
```

```python
# ----------------------------------------------------
# Notification callback: decode FusionData and write to Influx
# ----------------------------------------------------

def make_notification_handler(label: str, write_api, bucket: str):
    """
    Create a notification handler bound to a specific device label
    and Influx bucket.
    """
    def notification_handler(sender: int, data: bytearray):
        # Expecting 12 bytes: float pitch, roll, yaw
        if len(data) == 12:
            pitch, roll, yaw = struct.unpack("<fff", data)
            print(
                f"[{label}] pitch={pitch:6.2f}°  roll={roll:6.2f}°  yaw={yaw:6.2f}°"
            )

            # Build a point similar to sensor_to_influx.py
            p = (
                Point("orientation")  # measurement name
                .tag("device", label)
                .field("pitch", float(pitch))
                .field("roll", float(roll))
                .field("yaw", float(yaw))
                .time(time.time_ns(), WritePrecision.NS)
            )

            try:
                write_api.write(bucket=bucket, org=INFLUX_ORG, record=p)
                # Optional: print confirmation
                # print(f"[{label}] wrote point to bucket {bucket}")
            except Exception as e:
                print(f"[{label}] Error writing to InfluxDB: {e}")

        else:
            # If something weird comes through, dump raw
            print(f"[{label} RAW] from {sender}: {data.hex()} (len={len(data)})")

    return notification_handler
```

```python
# ------------------------------------------------------
# Handle single device connection
# ------------------------------------------------------

async def handle_device(device, device_label: str, write_api, bucket: str):
    """
    Connect to a single device and listen for notifications.
    Runs indefinitely until cancelled or disconnected.
    """
    print(f"\n🔗 [{device_label}] Connecting to {device.name} @ {device.address}...")

    def disconnected_callback(client):
        print(f"⚠️ [{device_label}] Disconnected from {client.address}")

    try:
        async with BleakClient(device.address, disconnected_callback=disconnected_callback) as client:
            if not client.is_connected:
                print(f"❌ [{device_label}] Failed to connect.")
                return

            print(f"✅ [{device_label}] Connected!")

            # Create a unique notification handler for this device and bucket
            handler = make_notification_handler(device_label, write_api, bucket)

            # Subscribe to notifications
            print(f"📡 [{device_label}] Subscribing to characteristic: {IMU_CHAR_UUID}")
            try:
                await client.start_notify(IMU_CHAR_UUID, handler)
            except Exception as e:
                print(f"⚠️ [{device_label}] Could not start notifications: {e}")
                return

            print(f"👂 [{device_label}] Listening for IMU notifications...")

            # Keep connection alive and listen for notifications
            try:
                while client.is_connected:
                    await asyncio.sleep(1.0)
            except asyncio.CancelledError:
                print(f"\n🛑 [{device_label}] Stopping notifications...")
                await client.stop_notify(IMU_CHAR_UUID)
                raise

    except Exception as e:
        print(f"❌ [{device_label}] Error: {e}")
```

```python
# ------------------------------------------------------
# Main client logic
# ------------------------------------------------------

async def main():
    """Main function to scan and connect to multiple devices concurrently."""
    # Set up InfluxDB client and write_api once, reuse for all devices
    influx_client = InfluxDBClient(url=INFLUX_URL, token=INFLUX_TOKEN, org=INFLUX_ORG)
    write_api = influx_client.write_api(write_options=SYNCHRONOUS)

    devices = await find_devices()

    if devices is None or len(devices) == 0:
        print("Exiting client. Once your ESP32s are advertising, run again.")
        return

    # Create tasks for each device connection
    tasks = []
    for idx, device in enumerate(devices):
        device_label = device.name if device.name else f"Device-{idx+1}"

        # Decide which bucket to use based on device name
        if device.name == TARGET_NAME1:
            bucket = BUCKET_IMU1
        elif device.name == TARGET_NAME2:
            bucket = BUCKET_IMU2
        else:
            # Fallback if some other device slips in
            bucket = BUCKET_IMU1

        task = asyncio.create_task(handle_device(device, device_label, write_api, bucket))
        tasks.append(task)

    print(f"\n🚀 Starting {len(tasks)} concurrent connection(s)...")
    print("Press Ctrl+C to stop all connections.\n")

    try:
        # Run all device handlers concurrently
        await asyncio.gather(*tasks)
    except KeyboardInterrupt:
        print("\n\n🔴 Keyboard interrupt received. Shutting down all connections...")
        # Cancel all tasks
        for task in tasks:
            task.cancel()
        # Wait for all tasks to complete cancellation
        await asyncio.gather(*tasks, return_exceptions=True)
        print("✅ All connections closed.")


if __name__ == "__main__":
    asyncio.run(main())
```

Grafana Query from InfluxDB

A (influxdb)

```
from(bucket: "IMU2")
  |> range(start: -5m)
  |> filter(fn: (r) => r["_measurement"] == "orientation")
  |> filter(fn: (r) => r["device"] == "StrikeSense-IMU-2")
  |> filter(fn: (r) =>
      r["_field"] == "pitch" or
      r["_field"] == "roll"  or
      r["_field"] == "yaw"
  )
  |> aggregateWindow(
      every: 1s,          // change to 100ms / 500ms / 2s if you want
      fn: mean,
      createEmpty: false
  )
  |> pivot(
      rowKey: ["_time"],
      columnKey: ["_field"],
      valueColumn: "_value"
  )
```