# WEARABLE BASKETBALL JUMPSHOT MECHANICS ANALYZER
ECE 445 Final Report

---

Team #78
Tanmay Nair, Arjun Vyas, Aiden Zack

Professor: Joohyung Kim
TA: Mingrui Liu

## Abstract

This report documents the design, implementation, and verification of a wearable basketball jumpshot mechanics analyzer that provides objective feedback on shooting form. The system uses multiple inertial measurement units (IMUs) and a microcontroller to collect synchronized motion data during a jumpshot. The collected data is processed in a Python-based interface to extract key features such as joint timing and angular velocity. A “success profile” is generated from multiple successful shots and used as a baseline for evaluating future attempts. The final system demonstrates the use of synchronized sensor data to analyze athletic motion and provide quantitative feedback. This report presents the system architecture, implementation, verification results, and potential improvements.

Contents

# 1 Introduction

## 1.1 Purpose

### 1.1.1 Problem

A basketball jumpshot involves a chain of body mechanics that requires coordination from your feet to your wrist to achieve a simple goal that is much more complicated than what the average person sees: Making the ball go in the hoop. So many players across the world have exhibited different mechanics in their jumpshot, so when they reach out to coaching for help, they tend to hear subjective advice that is often inconsistent, difficult to put into numbers, and, more importantly, harder to fit into the player's perspective. Existing resolutions utilize shot trajectory and do not tap into the biomechanics that reside in the shooter. In essence, this leads to players lacking reliable, repeatable data to identify points of improvement in their mechanics, address consistency issues, and record progress.

### 1.1.2 Solution

This project will implement a system dedicated to quantifying a user's basketball jumpshot by analyzing the consistency and timing of the "kinetic chain". It starts with node sensors that will be worn on the user's shooting wrist, shooting elbow, hip, and the knee of the user's shooting side. These sensors will hold an IMU, microcontroller, and wired communication. The knee sensor will focus on lower-body motion and take measures related to shot success, such as the timing of the jump and how much the knee flexes to determine the dip [6]. The wrist sensor will look at the upper-body mechanics that finish out the shot, like the angular velocity and release timing of the wrist, along with how high it sits for the follow-through [6]. The hip sensor will measure vertical displacement and horizontal tilt [6]. The elbow sensor will measure the angular velocity and push of the shot [6]. These 4 data nodes will be synchronized in our system, extracted for timing measures like jump-to-release, and then processed for evaluation and feedback. This will focus on the repeatability and timing of the user's body mechanics, providing user-oriented assistance that adjusts as the user progresses.

## 1.2 Functionality

### 1.2.1 Synchronized Motion Capture

The system captures motion data from four inertial measurement units (IMUs) placed on the wrist, elbow, hip, and knee. A hardware FSYNC trigger is used to synchronize sampling across all sensors, while chip select signals ensure that incoming data is correctly associated with each IMU. This synchronized data collection enables accurate temporal alignment of joint movements, which is essential for analyzing the timing relationships within a basketball jumpshot.

### 1.2.2 Robust Motion Measurement

Each IMU subsystem measures both angular velocity and linear acceleration during the shooting motion. The sensors are configured to capture high-speed and high-impact movements, such as wrist flicks and landing forces, ensuring that the full range of motion throughout the jumpshot is recorded. This allows the system to reliably collect meaningful data across all phases of the shot.

### 1.2.3 Kinematic Analysis and Feature Extraction

The collected sensor data is processed in a Python-based backend to identify key kinematic events, such as peak joint velocities and critical timing points in the shooting sequence. These events are determined by analyzing changes in IMU data over time to capture important motion characteristics of each joint. Timing relationships between joints are calculated to evaluate coordination throughout the shot, allowing the system to model the user's kinetic chain and quantify how different body segments contribute to the overall motion. The extracted features are used to compare individual shots against a reference "success profile" constructed from multiple successful attempts. Based on this comparison, the system provides basic feedback highlighting deviations in timing, coordination, and movement consistency.

### 1.2.4 Shot Evaluation and Feedback Interpretation

The system evaluates shots by comparing recorded motion data against a reference "success profile" generated from successful trials. Each missed shot is assigned a similarity score that quantifies how closely the motion matches the reference profile. Shots with higher similarity scores indicate strong alignment with successful timing and coordination patterns, while lower scores reflect deviations in joint sequencing and movement consistency. By analyzing both high- and low-similarity shots, the system provides interpretable feedback that highlights differences in timing and coordination. This allows users to directly relate motion characteristics to shot quality and better understand inconsistencies in their form.

## 1.3 Subsystem Overview

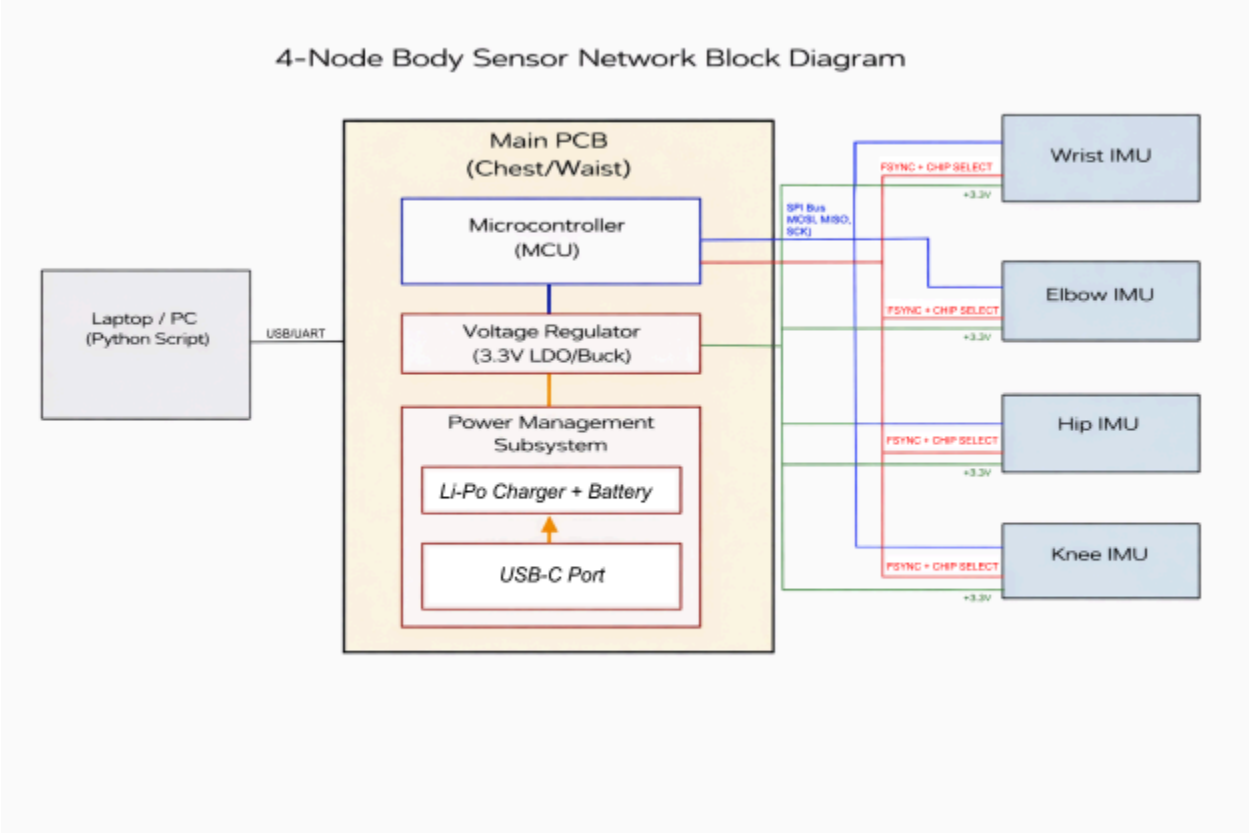



Figure 1: Top Level Block Diagram

The overall system consists of four primary subsystems: the IMU sensing subsystem, the microcontroller subsystem, the software data processing subsystem, and the battery management subsystem. A top-level block diagram of the system is shown in *Figure 1*.

The IMU sensing subsystem includes four inertial measurement units positioned on the wrist, elbow, hip, and knee to capture motion data during a basketball jumpshot. These sensors are connected via an SPI interface to the Arduino Nano ESP32 microcontroller, which forms the core of the microcontroller and communication subsystem. A hardware FSYNC signal is distributed to all IMUs to ensure synchronized sampling, while individual chip select lines allow the microcontroller to retrieve data from each sensor.

The battery management subsystem provides regulated power to the system through a USB-C input, Li-Po battery, and onboard voltage regulation circuitry. This subsystem ensures stable voltage delivery to peripherals, including IMU sensors and debugging LEDs, enabling reliable operation during data collection.

The microcontroller collects and organizes synchronized IMU data and transmits it over a USB serial connection to the software data processing subsystem. As shown in the block diagram, the microcontroller serves as the central hub connecting all IMU nodes and the external computer. To ensure segmentation, the microcontroller runs on a separate power rail and receives a 5V input from the on-board USB.

Finally, the Python-based software processing subsystem then extracts key motion features such as joint timing and peak movement characteristics to analyze the user's kinetic chain. The processed data is visualized through a user interface, allowing users to interpret their motion and understand how timing and coordination affect their shooting performance.

# 2 Design

## 2.1 Introduction

The wearable jumpshot analyzer consists of four main subcomponents: the Arduino Nano ESP32 microcontroller, ICM-20948 IMU sensors, a battery management system, and software-based data processing. The IMU sensors are connected to the microcontroller using SPI communication, allowing the collected motion data to be read by the microcontroller and transmitted to a PC through USB/UART for processing and user feedback. The IMU sensors are powered through the battery management system, which receives input voltage from the LiPo battery. This voltage is then stepped down to the required 3.3 V using a voltage regulator. Since the internal logic of the IMU sensors operates at 1.8 V, a resistor network was used to manually step down the signal voltage according to the IMU specifications.

## 2.2 Design

### 2.2.1 Microcontroller (Arduino Nano ESP32)

The Arduino Nano ESP32 serves as the bridge, expediting the transfer from IMU readings to the software program for processing. We power it through the USB-C port (requires 5V input via PC). The MCU reads off the accelerometer and gyroscope data from each of the 4 IMUs through the shared SPI bus, with chip select (CS) signals to identify each IMU and an FSYNC pulse to synchronize the IMUs to collect readings simultaneously.

Balancing the speed of the SPI bus with the sampling rate was crucial to the full-scale implementation. For starters, as per the Adafruit ICM-20948 documentation, the sampling rates for both the accelerometer and gyroscope are individually defined by an integer-based divisor we configured. This, along with setting the range of the sensors to 16g and 2000 degrees/second, was established in the figure below:

```
icm[idx].setAccelRange(ICM20948_ACCEL_RANGE_16_G);
icm[idx].setGyroRange(ICM20948_GYRO_RANGE_2000_DPS);

icm[idx].setAccelRateDivisor(4); // 225 Hz --> 1125 / (1.0 + accel_divisor)
icm[idx].setGyroRateDivisor(4);  // 220 Hz --> 1100 / (1.0 + gyro_divisor)
```

*Figure 2: Adafruit ICM-20948 Range and Rate Calibration*

A sampling rate of 220 Hz allows us to grab a sample every ~4.55 milliseconds. This is an efficient choice, as the fastest movement we can expect to see, a wrist flick at approximately 30 milliseconds, can still be captured with ~5-6 samples.

Each IMU in this project, although equipped with 9 axes of data, will only send over 3 axes of accelerometer and 3 axes of gyroscope data. This, along with an additional byte reserved for a register address, which allows the microcontroller to identify each IMU, adds up to **13 bytes/sample.** We can thus quantify our total bus traffic as:

$$13\ bytes/sample \cdot\ 220\ samples/sec\ \cdot 4\ IMUs\ =\ 11,400\ bytes/s\ =\ 11.4\ kBps$$

This only uses up 4.576% of the available bandwidth on a 2 MHz SPI bus, which provides a comfortable margin for our bus to function at. Additionally, it remains conservative, as we have to consider the peripherals' cable propagation delays. A 2 MHz SPI clock indicates a half-cycle period of 250 ns. Based on TI's typical signal propagation velocity of 5 ns/m [5], figure 2 illustrates the distance of each IMU from the microcontroller, the corresponding one-way delays, and its ratio to the half-cycle period:

| | Cable Length from MCU [4] | One-way Propagation Delay | Ratio with respect to Half-Cycle Period (%) |
|---|---|---|---|
| Wrist IMU | 1.00 m | 5.00 ns | 0.9% |
| Elbow IMU | 0.45 m | 2.25 ns | 1.2% |
| Hip IMU | 0.60 m | 3.00 ns | 1.8% |
| Knee IMU | 0.90 m | 4.50 ns | 2.0% |

*Figure 3: IMU Cable Lengths and Propagation Delays*

Our worst-case instance of propagation delay contributes only 5 ns to the half-cycle, promoting efficient and credible communication across the established bus. While this does not eliminate signal integrity concerns, these calculations validate the implementation of a 2 MHz SPI clock rate and ~220 Hz sampling rate.

This data is pushed to the PC via USB UART at a rate of 921600 baud. Originally, this was set to a 115,200 baud rate stream a 48-byte synchronized data packet, but implementation in later stages of the project identified the need to scale the rate by a factor of 8 to prevent buffer overflow at the host and leave room for additional metadata for the dashboard.

### 2.2.2 Off-Board Inertial Measurement Units (IMUs)

An IMU sensor network was implemented using four ICM-20948 6-axis IMU sensors strategically placed along the user's shooting side at the wrist, elbow, hip, and knee. These sensors collected biomechanical motion data, including gyroscope and accelerometer

measurements along the x, y, and z-axes. Each IMU sensor was supplied with a 3.3 V input from the battery management system. Since the sensor's internal logic operates at 1.8 V, a resistor network was used to provide the required logic level voltage according to the sensor specifications.

### 2.2.3 Battery Management System

Our customized battery management system serves as a separate 3.3V power rail for our IMUs and debugging LEDs. It consists of a USB-C receptacle, Charger & Power Path IC (TI BQ24074RGT), the Li-Po battery connector, and a 3.3V LDO regulator (TLV73333). When the USB-C is connected, the Charger IC drives the system output (VSYS at ~4.4V) to the LDO while simultaneously charging the Li-Po battery cell. When disconnected, the IC autonomously manages the power path by switching the Li-Po battery cell to drive our VSYS signal (now at 3.3-4.2V) and drops down to 3.3V by the LDO.

Translating a PCB design required mapping out resistors and capacitors close to pinouts to suppress noisy, transient signal components and stabilize signal integrity both on and off-board via cables. This includes the following (Finalized PCB Schematic and Design in Figures 8 and 9):

- Capacitors were placed by the charger IC and LDO to supply instant current and smooth out the voltage going into and out of these components.
- The 5.1kΩ resistors for the USB-C receptacle serve as a power sink that dissipates electrical-to-thermal energy safely.
- The BQ24074RGT Charger IC required installing resistors for certain pinouts to define its functionality [7].
    - The 1.92kΩ resistor connected to the ISET pin serves as a constraint on the battery charge current at 500 mA.
    - The 3.16kΩ resistor connected to the ILIM pin serves as a backup in the case of our current EN1/EN2 configuration (EN1 = 1, EN2 = 0 → USB500 mode). At this resistance, our maximum input current caps out at ~490 mA
    - As per the datasheet, no external thermistor is required from this charger IC, so the resistor is set at 10kΩ

### 2.3.4 Software Data Processing

The software subsystem is organized into three main components: data acquisition, data processing, and user interface. Sensor data is collected from the microcontroller over a high-speed serial connection and handled in a dedicated background thread. This design ensures that data collection is non-blocking and can operate continuously without interrupting the user interface or processing pipeline. The data processing component extracts key motion features

from the incoming IMU data. Shot segmentation is performed using threshold-based triggering, where knee motion initiates capture and wrist peak activity determines the end of the shot. Shot initiation is triggered when the knee angular velocity magnitude exceeds a threshold of 1.2, while shot termination is determined by detecting a wrist peak exceeding a threshold of 3. This approach was selected due to its computational efficiency and ability to operate in real time without requiring complex or resource-intensive algorithms. Once a shot is captured, relevant kinematic features such as joint timing, peak angular velocity, and movement magnitude are computed to analyze the user's kinetic chain. The success profile, recent shot status, and recent shot performance are displayed in Figure 15.

To improve system efficiency, two data acquisition modes were implemented: a lightweight "live" mode and a full capture mode. In live mode, only a subset of sensor data (knee gyroscope values) is transmitted and monitored for shot initiation. This reduces bandwidth usage and allows for low-latency trigger detection. Once a shot is detected, the system switches to full capture mode, where all IMU data is streamed and recorded for detailed analysis. This design minimizes unnecessary data processing while ensuring that complete motion data is available for relevant events. Compared to continuously streaming all sensor data, this approach reduces computational load and improves real-time responsiveness of the system. To improve measurement accuracy, a zero calibration procedure is implemented before data collection. During calibration, the system records a short window of IMU data while the sensors remain stationary. The average value of each sensor axis is computed and stored as a baseline offset. These offsets are subtracted from all subsequent measurements to reduce the effects of sensor bias and drift. This ensures that the recorded motion data more accurately reflects true movement rather than inherent sensor error. By applying zero calibration, the system improves the consistency and reliability of extracted features such as peak values and timing relationships. All stated dashboard functionality is shown in Figure 14.

The software subsystem uses mathematical operations to quantify motion and evaluate shot consistency. Angular velocity magnitude is computed using the Euclidean norm of the gyroscope axes:

$$|w| = \sqrt{g_x^2 + g_y^2 + g_z^2}$$

This provides a scalar representation of joint motion that is independent of orientation. Timing relationships between joints for evaluating coordination within the kinetic chain are calculated using differences in event timestamps:

$$\Delta t = t_2 - t_1$$

To assess shot quality, a similarity score is computed by comparing extracted shot metrics to a reference "success profile." This is implemented using normalized deviations (z-scores) across

multiple features, which are combined to produce an overall similarity metric, and is shown in Figure 16. This approach provides a quantitative measure of how closely a shot matches previously successful attempts, as the system is designed to evaluate the repeatability of an individual's shooting form rather than compare it to a universal "ideal" shot.

For the user interface, initial development was performed using Spyder with static plotting tools. While sufficient for early debugging, this approach lacked interactivity and scalability. The system was redesigned using Dash and Plotly to support real-time visualization and user interaction, significantly improving usability and demonstration capability. Additionally, alternative data processing methods such as advanced filtering or frequency-domain analysis were considered. However, the final implementation prioritizes time-domain feature extraction, which is sufficient for capturing key motion characteristics while maintaining low computational overhead.

### 2.3.5 Full System Integration

To fully integrate each subcomponent into the system, several design decisions were required. Standard 2.54 mm through-holes were used throughout the PCB, allowing 2.54 mm male header pins to connect the IMU sensors. For improved connection strength and ease of assembly, a 22 AWG 8-wire cable was crimped with 2.54 mm pitch connector terminals and inserted into a KF2510 housing connector. The cables were between 0.5m and 1m in length, and velcro straps were designed to route the wires around the user's body while maintaining enough slack to allow a full shooting motion. A velcro body harness and additional body straps were also created to interface with the 3D-printed PCB and IMU enclosures. Finally, a specialized 2.00 mm footprint was used for the LiPo battery through-holes on the PCB to meet the battery's connector specifications and ensure the system could be properly powered.

# 3 Cost and Schedule

## 3.1 Costs Analysis

The total cost of materials is $153.24. The project required approximately 150 hours per team member. Using an hourly rate of $43.32, based on the average starting salary for electrical engineers at the University of Illinois [4], the labor cost is $16,245 per member, or $48,735 for the three-person team. The total project cost, including materials and labor, is $48,888.24.

| Description | Manufacturer | Qty | Price | Link |
|---|---|---|---|---|
| Arduino Nano ESP32 Microcontroller [9] | Arduino | 1 | $19.30 | Link |
| IMU Sensor (6-Axis) [8] | Adafruit | 4 | $59.80 | Link |
| Lithium Polymer Battery | HXJNLDC | 1 | $14 | Link |
| USB-C Receptacle [10] | Molex | 1 | $2.67 | Link |
| IC Charger | Texas Instruments | 1 | $2.53 | Link |
| 2mm Header Pins (Strip of 8) | Adam Tech | 10 | $2.38 | Link |
| 2.54mm Header Pins (Strip of 10) | Hirose Electric Co Ltd | 10 | $10.58 | Link |
| 22AWG PVC Thermostat Wire | LEDLampsWorld | 1 | $21.99 | Link |
| KK 2.54mm Pitch Connector with Premium 22AWG Pre-Crimped Cables | Yoeruyo | 1 | $19.99 | Link |

*Figure 4: List of Components and Cost*

## 3.2 Schedule

| Week | Task | Person |
|---|---|---|
| **3/1-3/7** | Finalize PCB part selection for ordering and review with TA | Everyone |
| | Begin breadboard prototyping and validation of MCU–IMU communication | |
| **3/8-3/14** | Verify power regulation and voltage levels | Everyone |
| | Read and log accelerometer and gyroscope data | |

| | | |
|---|---|---|
| | **Breadboard Demo** - MCU communication with two IMUs | |
| **3/15-3/21** | Spring Break | Everyone |
| **3/22-3/28** | Begin the physical wiring | Everyone |
| | Validate off-board IMU communication | |
| **3/29-4/4** | Full system integration (multiple IMUs + hub) | Everyone |
| | Initial data streaming to the laptop | |
| **4/5-4/11** | **Progress Demo** | Everyone |
| | Validate system functionality during controlled motion | |
| | Debug communication and synchronization issues | |
| **4/12-4/18** | Solder components on the final round PCB | Everyone |
| | Improve wearable mounting and cable routing | |
| | Refine data processing and timing analysis algorithms | |
| **4/19-4/25** | **Mock Demo** | Everyone |
| | Finish building and implementing UI | |
| **4/26-5/2** | **Final Demo** | Everyone |

*Figure 5: Project Schedule*

# 4 Requirements and Verification

## 4.1 Microcontroller Subsystem

Several tests were performed on the Arduino Nano ESP32 microcontroller to validate that it functioned properly for our system requirements. First, we confirmed that the onboard LED turned on when the board was supplied with 5 V through the USB-C receptacle. We then used a multimeter to measure the voltage between the VBUS and GND pins, verifying that the reading was approximately 5 V. Next, we confirmed that data from the IMU sensors could be read over the SPI bus and displayed on the Arduino serial monitor. This verified that a communication link was established between the microcontroller and the IMU sensors. Additional SPI validation methods were also used, which are discussed in more detail in the IMU sensor network section.

## 4.2 IMU Sensor Network

To ensure full functionality of the IMU sensors, we first used a multimeter to measure the voltage difference between Vin and GND across all four IMUs, confirming that each sensor received approximately 3.3 V. Next, SPI bus communication was validated using the built-in WHO_AM_I register on each IMU. The figure below depicts the register value printed to the serial monitor, where the expected value of 0xEA was observed, matching the ICM-20948 datasheet.

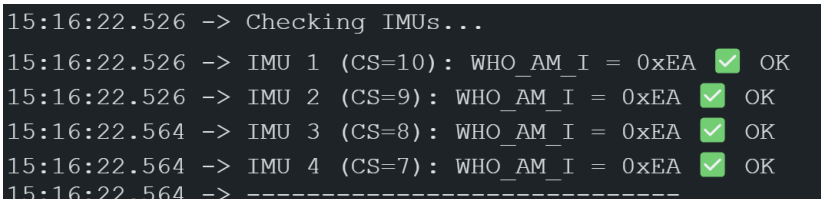



*Figure 6: WHO_AM_I IMU Register Validation*

After confirming power and communication, we verified that live telemetry data responded in real time by shaking and rotating the sensors and observing the corresponding changes in the output, as shown in Figures 12 & 13. The IMU setup begins with the user in a fully relaxed position to establish a consistent starting point for each shot. Once the user is at rest, the sensor data is zero-calibrated, as described in the software subsystem. This calibration improves the accuracy and consistency of comparisons between shooting motions.

## 4.3 Battery Management Subsystem

Validating the functionality of our battery management system involved the use of a multimeter to check voltage signals and continuity across certain pin-outs. Specifically, we verified trace continuities, such as the main 3.3V signal from VBAT all the way to the cable-length connection

to the off-board IMUs. VSYS was checked to match the input behavior when dependent on the plugged-in USB-C signal versus when dependent on the Li-Po battery pack. The LDO voltage regulator output was consistently checked for 3.3V to ensure stabilized power for peripherals, as shown in Figures 10 & 11. Lastly, battery pack usage was measured over time to verify the feasibility of long-term use and minimize the need for charging. The table below represents in-lab observations of the battery voltage as it powered our full system:

| Elapsed Time (Hrs) | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| VBAT (V) | 3.96 | 3.95 | 3.93 | 3.92 | 3.92 | 3.90 | 3.89 | 3.88 | 3.87 | 3.86 | 3.84 |

### 4.4 Software Subsystem

The system is designed to transfer IMU data from the microcontroller to the Python backend for processing, including filtering, feature extraction, and user interface display. To validate this functionality, a serial connection was established using the pyserial library, allowing data from the Arduino to be read directly in Python. The sensors were manually moved during testing, and corresponding changes were observed in the Python interface, confirming accurate data transmission and processing. This behavior is further demonstrated in the final system, where distinct motion events, such as wrist peaks, are clearly reflected in the processed data.

To reduce unnecessary data storage, the system is designed to capture only the portion of data corresponding to the full kinetic chain of a jumpshot. This was verified by conducting repeatable testing to determine appropriate thresholds for shot initiation (knee motion) and termination (wrist motion). Once these thresholds were established, the system recorded data segments corresponding to individual shots. Verification was performed by examining accelerometer and gyroscope plots, which showed that data capture begins just before the knee dip and continues briefly after the wrist release. This confirms that the system successfully captures the complete motion sequence of the jumpshot.

To provide meaningful feedback, the system generates a personalized “success profile” based on multiple successful shots. This profile stores key motion features, such as timing offsets and movement magnitudes, which serve as a baseline for comparison. Verification was performed by saving successful shots and printing the stored profile data to confirm that all relevant accelerometer and gyroscope information was correctly recorded. Once a sufficient number of samples was collected, new shots were evaluated against the success profile. The system generated similarity scores and feedback based on deviations in timing and coordination, as shown in Figure 16. This was verified by comparing known “good” and “bad” shots and confirming that shots with greater deviation produced lower similarity scores and more pronounced feedback. Additionally, the system was verified to continuously update the success

profile by adding new successful shots. This was confirmed by printing and comparing the updated profile after each new “make,” ensuring that the dataset expanded and improved the accuracy of future evaluations.

# 5 Conclusion

## 5.1 Accomplishments

The project successfully demonstrated a wearable system that captures synchronized motion data across multiple joints during a basketball jumpshot. The Arduino Nano ESP32 and four IMU sensors were integrated to reliably collect and transmit data with consistent sampling and temporal alignment. The system streamed data to a Python backend, where key features such as joint timing and movement magnitude were extracted. A “success profile” was generated from multiple trials, allowing new shots to be evaluated and feedback to be produced. Overall, the system showed that synchronized multi-sensor data can be used to analyze shooting mechanics and provide meaningful performance insights.

## 5.2 Uncertainties

While our project was a success in its entirety, certain components of the system require revisions that could have led to an even more stable, consistent design. We discovered tracing inconsistencies like connecting the same-net pins (such as those on the LDO for the ENABLE pin that was driven low originally, which required us to drive it high by tying it to the IN pin). Additionally, component positioning was another issue we faced, specifically for the USB-C receptacle. By not positioning the footprint towards the edge of the board, we were required to cut a notch through the ground plate fit for a USB-C cable. While these issues did not hinder the success of our project, it did require attention and workarounds to push our project forward.

A big discrepancy we noticed was the efficiency of our IMUs, specifically the baseline drift they experience as time passes and activity occurs. Our design initially zero-calibrates the sensors at the “rest” position, and once the reader starts, it begins to measure accelerometer and gyroscope data of shots without being recalibrated again. Figure 7 reminds us to assume that angles such as the pitch, roll, and yaw of the gyroscope (gx, gy, gz) can drift approximately 5-15 degrees after multiple samples [10].

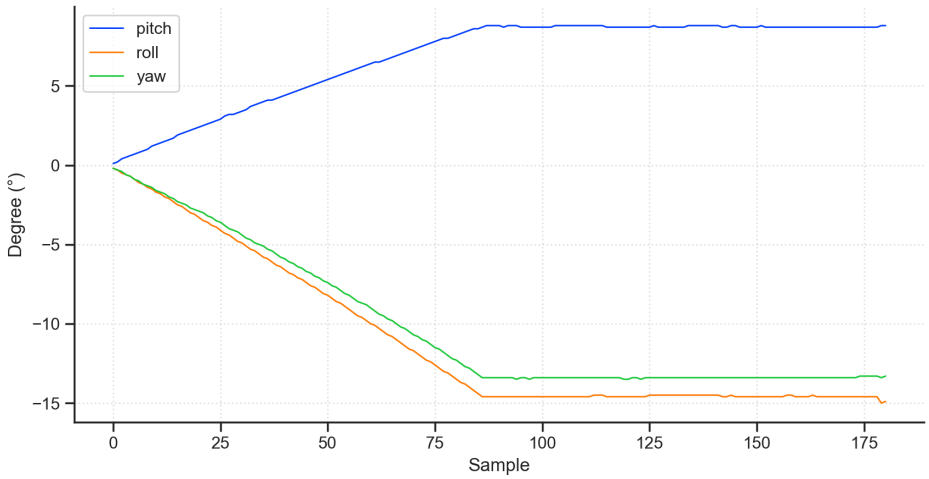



*Figure 7: ICM-20948 Potential Gyroscopic Drift over Multiple Samples*

While the metrics we utilized were relative-based and this drift is not significant, especially when measuring high-motion movements in our application, such as a wrist flick or knee bend, it's critical to understand the growing potential of precision loss over time after 400-500+ shots.

### 5.3 Future Work / Alternatives

Future improvements include transitioning to a wireless implementation (Bluetooth) to improve user mobility and reduce wiring constraints. This would require addressing challenges related to synchronization accuracy and communication latency. The physical design could also be refined into a more integrated wearable system to improve comfort, durability, and consistency of sensor placement. The system could also be extended to other sports or motion-based activities by adapting the data analysis methods to different movement patterns. For other sports, this would primarily involve modifying the trigger and end-of-action detection, as well as adjusting the feedback logic to reflect sport-specific mechanics.

### 5.4 Ethical Considerations

This project considers applicable engineering standards relevant to wearable, low-power electronic systems and data-driven software applications. IEEE standards and best practices guide the design of the embedded hardware and software with respect to electrical safety, system reliability, and responsible interaction with users [2]. General UL safety guidelines for low-voltage, battery-powered consumer electronics inform component selection, power management, and enclosure design to reduce electrical and mechanical hazards [3]. Additionally, ACM ethical guidelines influence the design of the data analysis and feedback software to ensure transparent interpretation of sensor data and to avoid misleading performance claims [1].

The ethical risk of injury resulting from changes in shooting mechanics relates directly to IEEE Code of Ethics Principles 1, 3, and 6 [2]. To avoid ethical breaches, the system is designed to

prioritize user safety by limiting feedback to observational metrics focused on timing consistency and repeatability, rather than instructing users to make abrupt or corrective changes to their mechanics. In accordance with Principle 3, the system avoids absolute claims about a “proper” shooting form, as there is no “perfect” form, and clearly communicates its limitations. To further prevent harm under Principle 6, safeguards are implemented to discourage abrupt form changes, including warnings against overuse and guidance to discontinue use if discomfort or pain occurs. These design choices ensure that feedback supports user awareness without encouraging unsafe changes that exceed the physical limitations of the user’s body. These considerations align with course ethics guidelines emphasizing harm prevention, transparency, and responsible system use.

# 6 References


[1] Association for Computing Machinery, "ACM Code of Ethics and Professional Conduct," 2018. [Online]. Available: https://www.acm.org/code-of-ethics

[2] I*EEE Code of Ethics*, IEEE, 2020. [Online]. Available: https://www.ieee.org/about/corporate/governance/p7-8.html

[3] *UL Solutions*, "Lithium-ion battery safety concerns and precautions for brands," UL, [Online]. Available: https://www.ul.com/insights/lithium-ion-battery-safety-concerns-and-precautions-brands

[4] University of Illinois Urbana-Champaign, "Electrical Engineering undergraduate program" Grainger College of Engineering, [Online]. Available: https://grainger.illinois.edu/academics/undergraduate/majors-and-minors/electrical-engineering

[5]T. Kugelstadt, "Extending the SPI bus for long-distance communication." Accessed: Apr. 01, 2026. [Online]. Available: https://www.ti.com/lit/an/slyt441a/slyt441a.pdf?ts=1775077654504&ref_url=https%253A%252F%252Fwww.google.com%252F

[6] *Physiopedia*, "Biomechanics of the Basketball Jump Shot", 2024 [Online]. Available: https://www.physio-pedia.com/Biomechanics_of_the_Basketball_Jump_Shot

[7] *Texas Instruments,* "BQ2407x Standalone 1-Cell 1.5-A Linear Battery Chargers with Power Path", 2021 [Online]. Available: https://www.ti.com/lit/ds/symlink/bq24074.pdf?ts=1772108676050&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FBQ24074

[8] *ICM-20948 9-DoF IMU Breakout Board*, Adafruit Industries LLC. [Online]. Available: https://www.digikey.com/en/products/detail/adafruit-industries-llc/4554/17039173

[9]"Amazon.com: Arduino Nano ESP32 with Headers [ABX00083] - ESP32-S3, USB-C, Wi-Fi, Bluetooth, HID Support, MicroPython Compatible for IoT & Embedded Projects : Electronics," Amazon.com, 2026. https://www.amazon.com/Arduino-ABX00083-Bluetooth-MicroPython-Compatible/dp/B0C947BHK5?th=1 (accessed May 07, 2026).

[10]adamgarbo, "Euler angles & drift," GitHub, Mar. 2021. https://github.com/sparkfun/SparkFun_ICM-20948_ArduinoLibrary/issues/40 (accessed May 07, 2026).

# Appendix A: Requirements and Verification Tables

| Requirements | Verification |
|---|---|
| • Arduino Nano ESP32 will receive an input voltage of 5V via USB-C connector. We will use a variable voltage source (i.e Scopy) to supply the ESP32 with required 5V. | • Verify the power LED on the ESP32 is on. Use a DC voltage multimeter to read the voltage between the VBUS and GND pins and verify the reading is ~5V. |
| • Arduino Nano ESP32 will communicate with the IMU sensors via SPI bus. | • Verify the SPI bus is intact by reading the IMU's WHO_AM_I register value. The returned value for the ICM-20948 IMU should be 0xEA. |
| • Arduino Nano ESP32 will read the recorded data from the IMU sensors. | • Verify the data is being read by waking the sensor and physically moving the IMU around. Read the acceleration numbers on the Arduino IDE and validate that the numbers being read are changing in real time. |
| • Arduino Nano ESP32 will transfer the read data to the PC for data processing and user feedback. | • Verify the data appears in the Arduino IDE serial monitor. When the IMU sensor is moving, validate that the numbers displayed in the serial monitor change relatively in real-time. |

Table 1: Microcontroller – Requirements & Verification

| Requirements | Verification |
|---|---|
| • The ICM-20948 IMU sensors will receive an input voltage of 3.3V from the battery management system. | • Verify the IMU sensors are receiving 3.3V by using a DC multimeter and measuring the voltage between the VIN and GND pins. |
| • The ICM-20948 IMU sensor will communicate with the Arduino Nano ESP32 via SPI bus. | • Verify the SPI bus is intact by reading the IMU's WHO_AM_I register value. The returned value for the ICM-20948 IMU should be 0xEA. |
| • The live ICM-20948 data will be read by the Arduino Nano ESP32. | • Verify the data is being read by waking the sensor and physically moving the IMU around. Read the acceleration numbers on the Arduino IDE and validate that the numbers being read are changing in real time. |

Table 2: IMU Connectors – Requirements & Verification

| Requirements | Verification |
|---|---|
| • The USB-C Receptacle will receive an input of 5V from the variable voltage source (i.e Scopy). | • Verify the receptacle receives 5V by using a DC multimeter. Measure the voltage between VBUS and GND and validate this value is ~5V. |
| • The Charger will receive an input of 5V from the USB-C Receptacle. | • Verify the charger receives 5V by using a DC multimeter. Measure the voltage between IN and GND and validate this value is ~5V. |
| • The Li-Po Battery will be charged by the charger. | • Validate that the Li-Po battery is actually being charged by plugging in a partially discharged battery. Validate that the battery voltage increases over time by using a DC multimeter. |
| • The system output voltage (VSYS) will receive 4.4V after internal stabilization when the USB-C is plugged in. When unplugged, VSYS will receive 3.0V - 4.2V. | • Verify the VSS receives 4.4V when the USB-C is plugged in by using a DC multimeter. Measure the voltage between VSYS and GND and validate this value is ~4.4V. When the USB-C is unplugged, repeat the same process and validate this value is between 3.0- 4.2V. |
| • The 3.3V LDO Regulator will step down the 4.4V or 3.3-4.2V (depending on if the charger is connected or not) to 3.3V. | • Verify the LDO regulator is stepping down the voltage to 3.3V by using a DC multimeter. Measure the voltage between OUT and GND and validate this value is ~3.3V. This is the voltage that is also being supplied to the 4 IMU sensors. For a sanity check also measure the voltage between VIN and GND on the IMU sensors to verify the proper voltage is being distributed. |

Table 3: Battery Management System – Requirements & Verification

| Requirements | Verification |
|---|---|
| • The Arduino IDE will transfer the data to Python. | • Verify data from the Arduino IDE is being transferred to Python by sending sensor values over USB-C via the pyserial library. Validate that the Python data is accurate by moving the IMU sensors and having it reflected in the dashboard. |

| | |
|---|---|
| • Python program will capture samples that fully encompass the entire kinetic chain of the jumpshot. | • Verify that the "start of the jumpshot" event occurs when the gyroscope magnitude (particularly at the knee, which starts the movement) exceeds a set threshold.<br>• Verify "end of jumpshot" occurs when the Wrist IMU's gyroscope magnitude in the y-axis exceeds a set threshold<br>• Ensure the sample captures a substantial number of frames before the knee trigger and after the wrist flick.<br>• Cross-check with the preceding sample of inactivity. |
| • Python program will create a personalized "Success Profile" based on trial shots for key timing offsets and motion magnitudes. The program will also produce feedback for the following shots, based on the discrepancies of the "Success Profile." | • Verify storage of "successful" shot attempts and their associated, significant accelerometer/gyroscope data.<br>• With a valid sample size built, verify program feedback aligns with "successful" data and cross-checks with current user attempts to deduce resolution.<br>• Verify that missed shots are labeled with a "similarity score" to the success profile, along with highlighted issues and severity score.<br>• Verify program adds additional "successful" shot attempts to the sample database to better generalize the user's shot profile. |

Table 4: Software Data Processing – Requirements & Verification

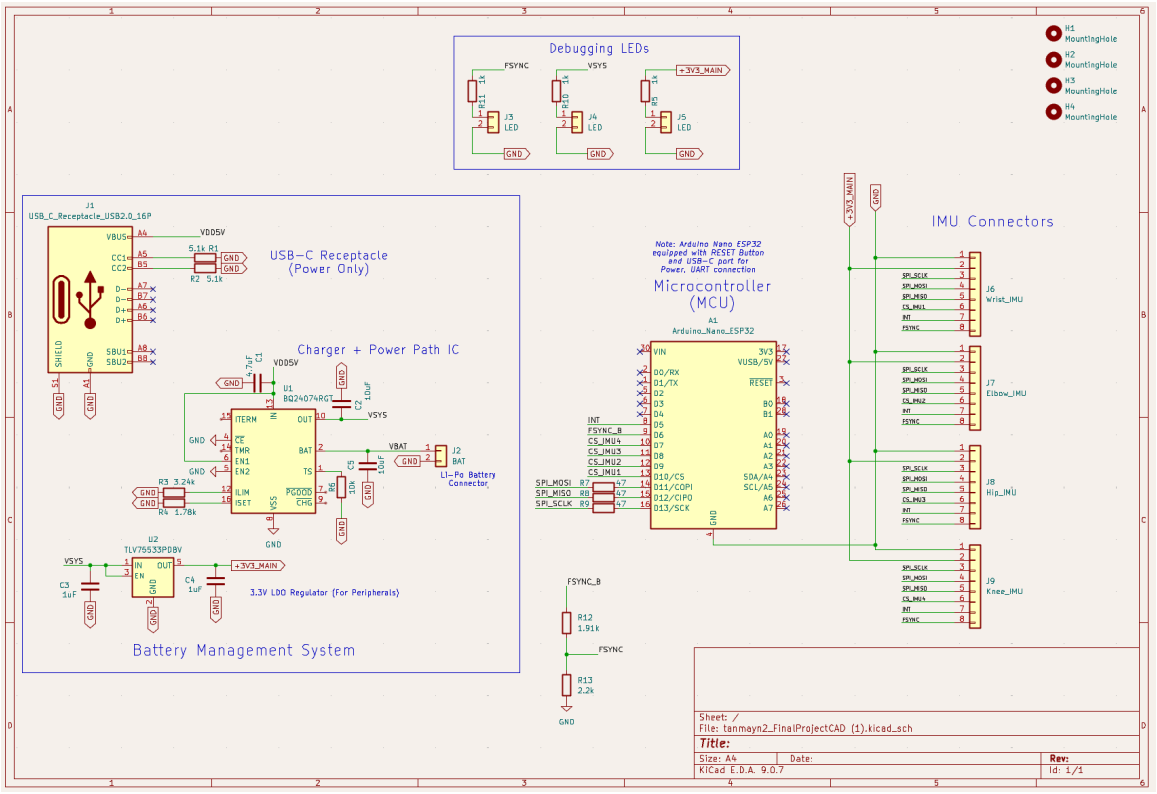



*Figure 8: Final PCB Schematic*

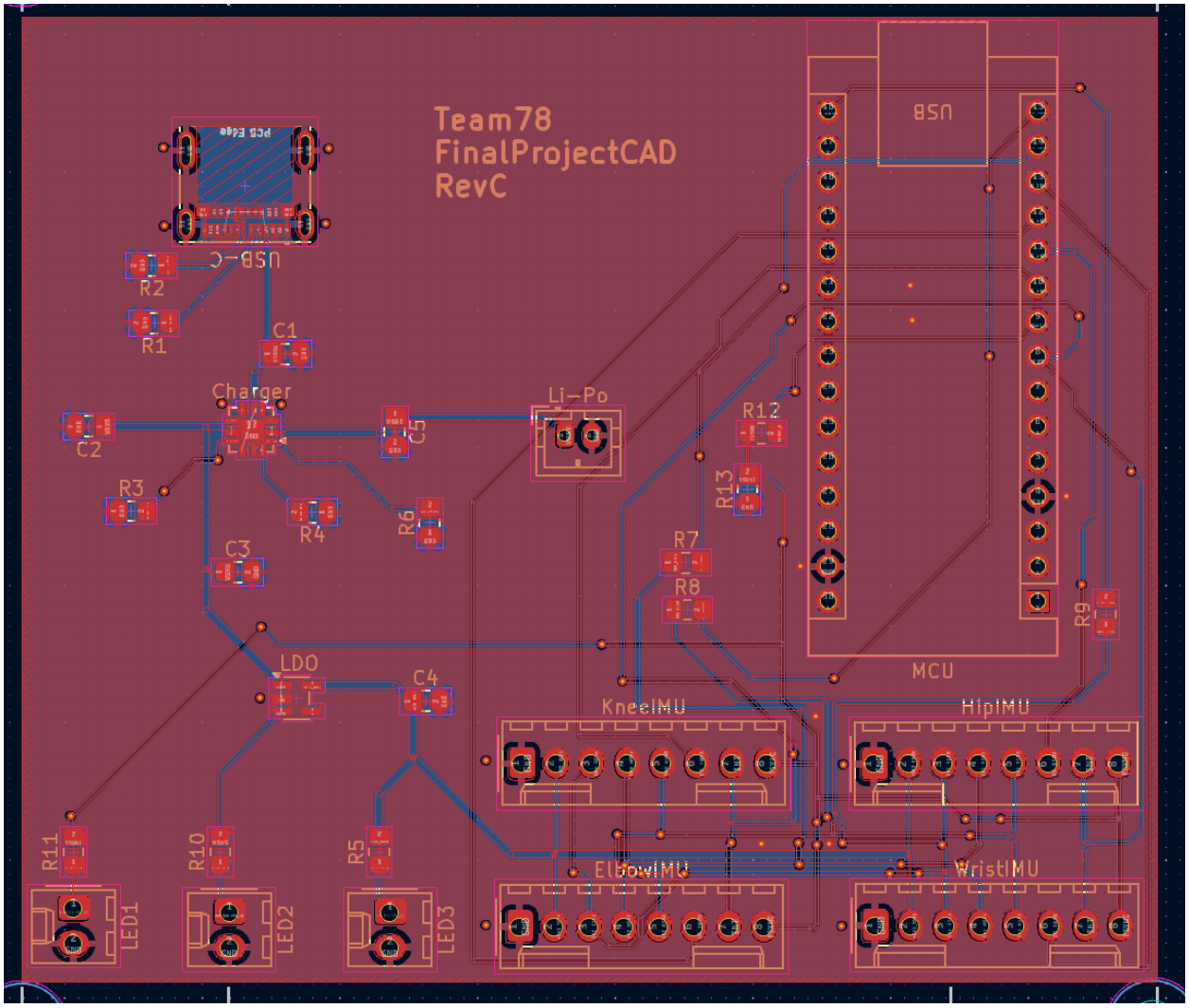
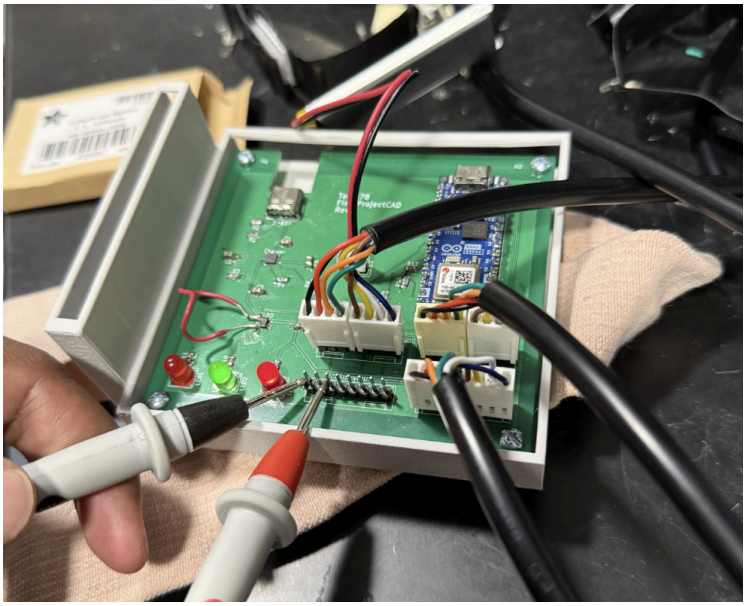
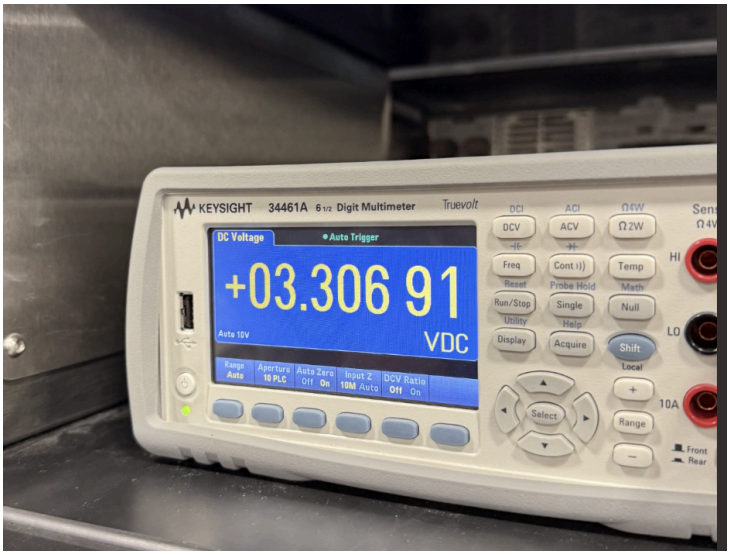



*Figure 9: PCB Design Layout*






*Figures 10 & 11: 3.3V Supply to IMUs*

```
13:55:55.234 -> Frame @ 295295016
13:55:55.234 -> IMU1 A:-0.41,0.24,9.96 G:-0.01,0.01,0.00 t(us):1
13:55:55.234 -> IMU2 A:-0.14,-0.14,9.97 G:0.02,0.00,0.00 t(us):440
13:55:55.234 -> IMU3 A:-0.27,-0.08,9.74 G:0.01,0.02,0.01 t(us):874
13:55:55.234 -> IMU4 A:-0.04,-0.16,9.79 G:-0.02,0.02,0.01 t(us):1302
13:55:55.234 -> Δ12=439 Δ23=434 Δ34=428 Δ14=1301
13:55:55.234 -> --------------------------
```

*Figure 12: IMU Data at Rest*

```
13:54:12.447 -> Frame @ 192493016
13:54:12.447 -> IMU1 A:-3.69,-0.15,9.66 G:0.06,3.17,-0.27 t(us):1
13:54:12.447 -> IMU2 A:-2.94,-0.62,9.99 G:0.10,3.15,-0.18 t(us):440
13:54:12.447 -> IMU3 A:-2.87,-0.63,10.13 G:0.10,3.17,-0.17 t(us):874
13:54:12.447 -> IMU4 A:-3.59,-0.65,9.57 G:0.07,3.17,-0.13 t(us):1302
13:54:12.447 -> Δ12=439 Δ23=434 Δ34=428 Δ14=1301
13:54:12.447 -> --------------------------
```

*Figure 13: IMU Data in Motion*

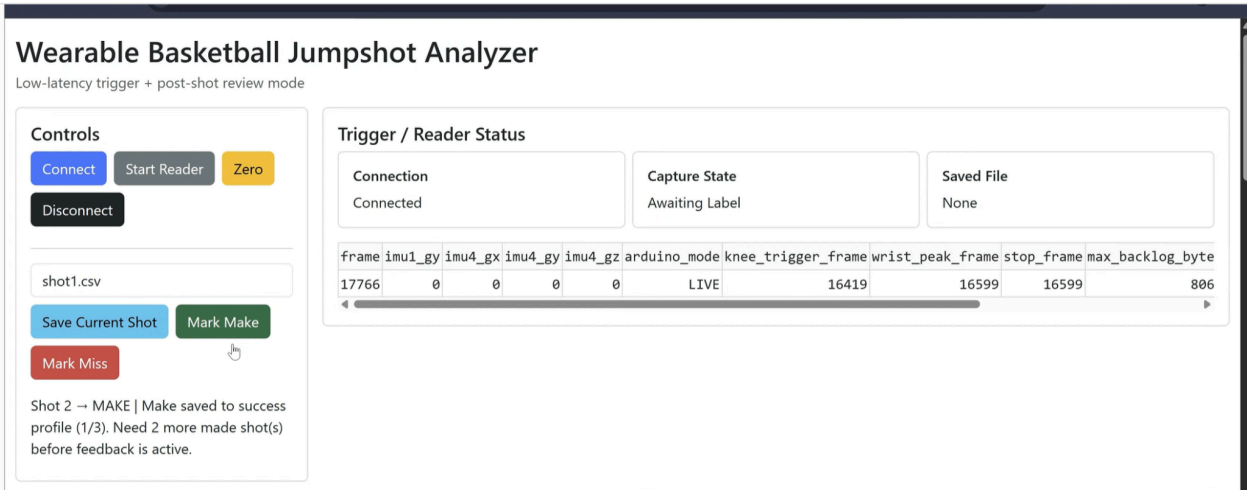



*Figure 14: UI Dashboard Controls and Live Reader*

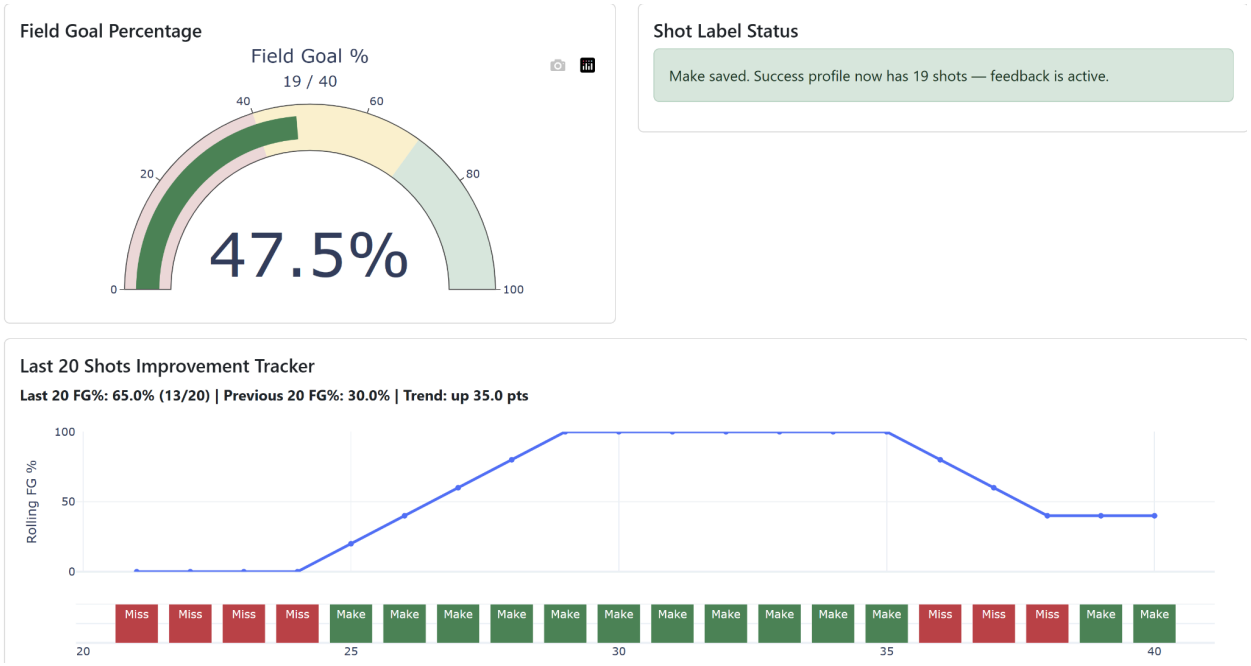



*Figure 15: Dashboard's FG%, Shot Label, and Improvement Tracker*

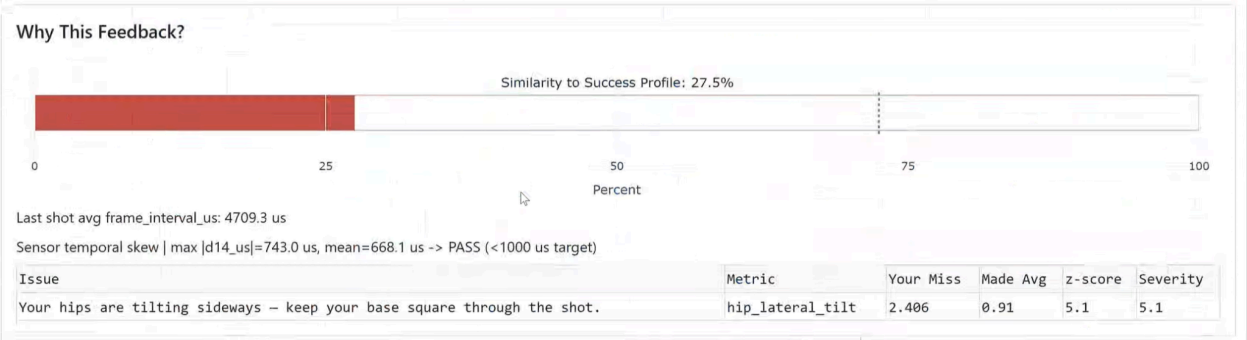



*Figure 16: UI Dashboards' Feedback Section*