STEVE ZAFEIRIOU

MICROCONTROLLERS IN ART (AND BEYOND)

V1.0

SENSOR DATA TRANSFER

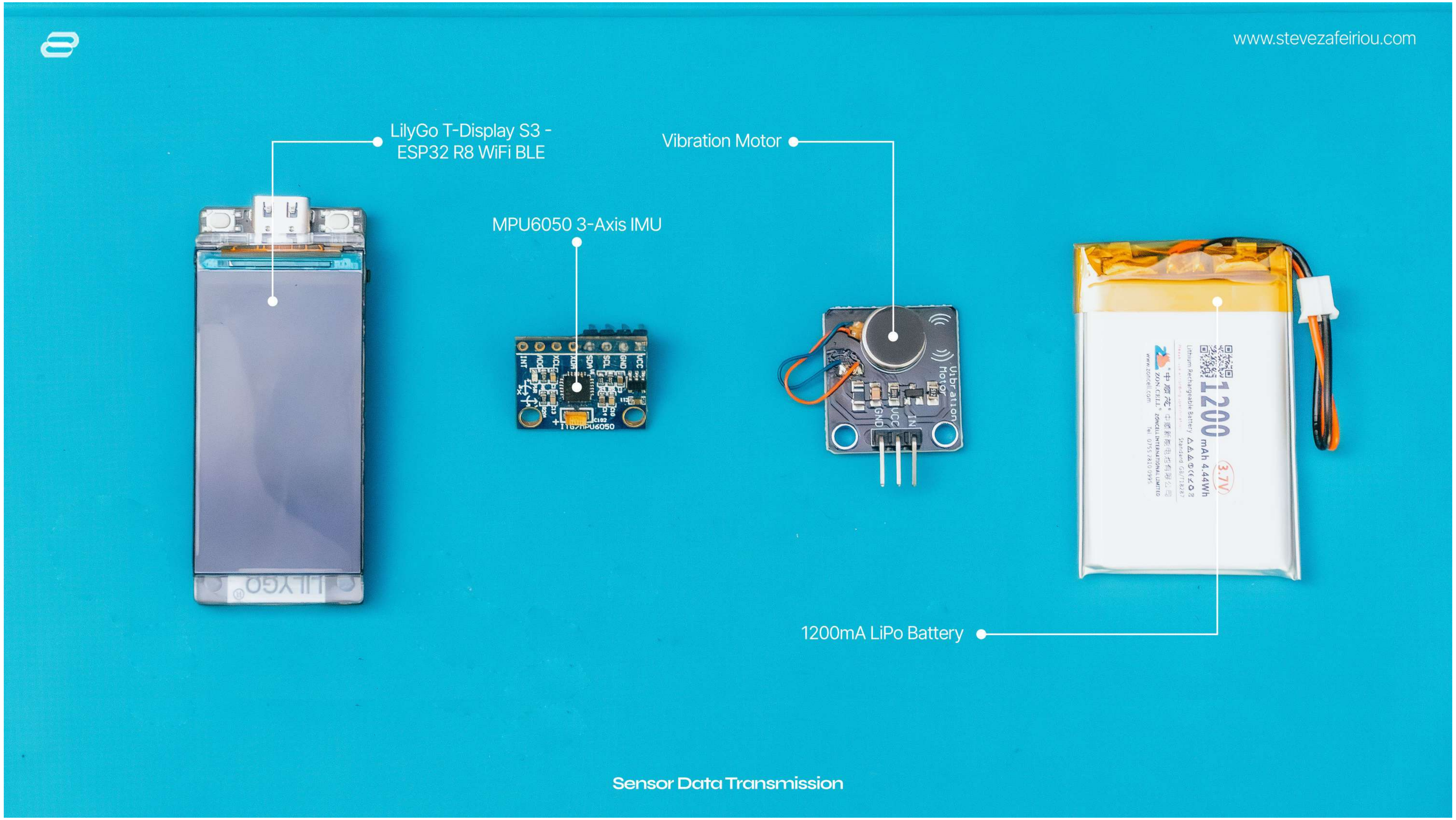
www.stevezafeiriou.com

01 Hardware Architecture Overview

A Powerful Motion Sensor Hub

You'll learn how to capture dynamic motion, provide tactile vibrations, and wirelessly relay information to a Node.js server—opening a world of possibilities for immersive experiences in art, interactive installation, or/and IoT applications.

1.1 Core Components



LilyGo T-Display Setup Guide: www.stevezafeiriou.com/lilygo-t-display-s3-setup

LilyGo T-Display S3 ESP32:

Dual-core 240MHz Xtensa LX6 processor

Integrated 1.9" IPS TFT display (170×320 resolution)

Built-in lithium battery management

8MB PSRAM + 16MB Flash memory

MPU6050 6-DoF IMU

3-axis accelerometer (±4g range)

3-axis gyroscope (±500°/sec)

I²C interface (17/18 pins for SDA/SCL)

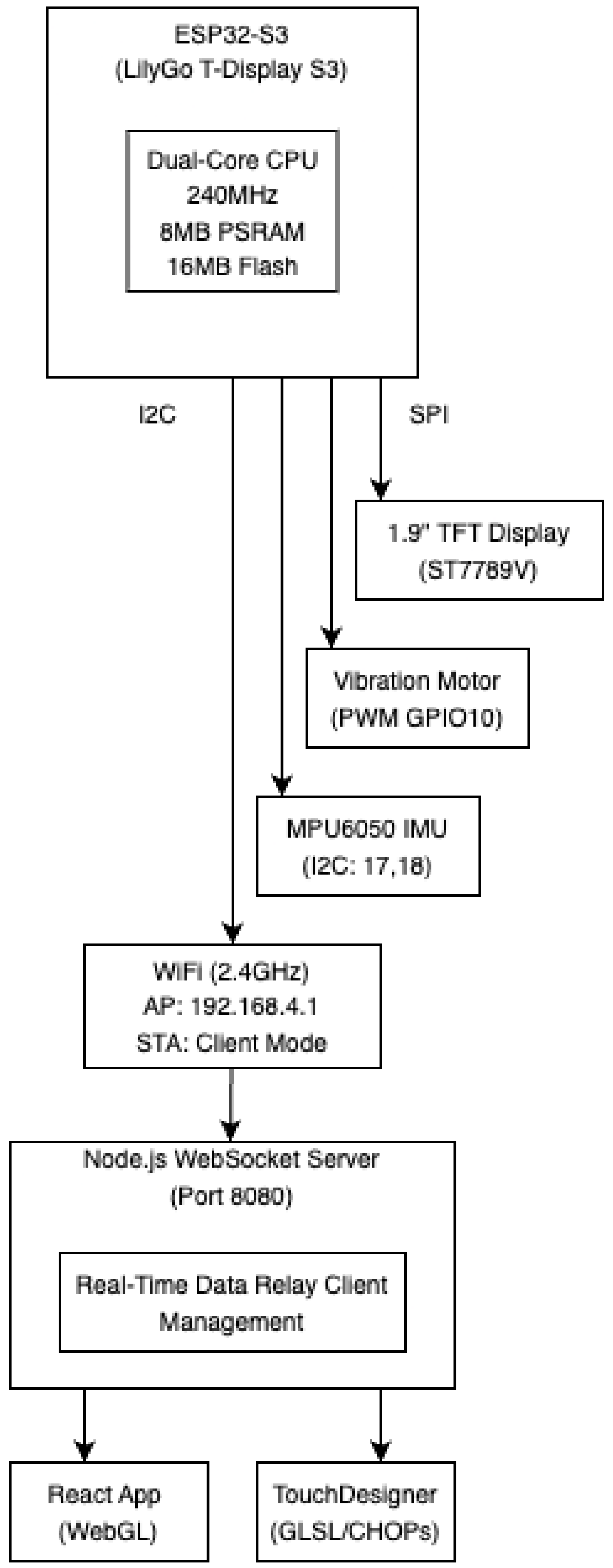
Vibration Motor

Eccentric rotating mass (ERM) type

Digital Control via GPIO10

3.3V logic compatible

1.1 Core Components



1.2 System Block Diagram

1. The ESP32 coordinates everything, reading sensor data from the MPU6050 over I²C.

2. The vibration motor is driven using a PWM signal for haptic feedback.

3. The TFT display is updated via SPI to show real-time readings or status messages.

4. Finally, the Wi-Fi interface allows the ESP32 to transmit motion data to a Node.js server and onward to client applications like React or TouchDesigner.

02 Core Firmware

Programming the LilyGo T-Display S3 - ESP32 using the Arduino IDE

In the file MPU6050Sensor.cpp, we have these key features:

Hardware Abstraction

```cpp
bool MPU6050Sensor::begin(int sda_pin, int scl_pin) {
    Wire.begin(sda_pin, scl_pin);
    if (!_mpu.begin()) return false;

    _mpu.setAccelerometerRange(MPU6050_RANGE_4_G);
    _mpu.setFilterBandwidth(MPU6050_BAND_21_HZ);
    return true;
}
```

Calibration Logic

```cpp
void MPU6050Sensor::calibrate(unsigned long duration) {
    SensorData sum = {0};
    unsigned long start = millis();
    unsigned samples = 0;

    while(millis() - start < duration) {
        sensors_event_t a, g, temp;
        _mpu.getEvent(&a, &g, &temp);
        sum.x += a.acceleration.x;
        sum.y += a.acceleration.y;
        sum.z += a.acceleration.z;
        samples++;
        delay(10);
    }

    _offset.x = sum.x / samples;
    _offset.y = sum.y / samples;
    _offset.z = (sum.z / samples) - 9.81; // Subtract gravity
    _calibrated = true;
}
```

Signal Filtering

```cpp
SensorData MPU6050Sensor::readData() {
    sensors_event_t a, g, temp;
    _mpu.getEvent(&a, &g, &temp);

    // Apply calibration
    SensorData raw = {
        a.acceleration.x - _offset.x,
        a.acceleration.y - _offset.y,
        a.acceleration.z - _offset.z
    };

    // Low-pass filter (adjust alpha for smoothing)
    static SensorData filtered = {0};
    const float alpha = 0.2; // 0 < alpha < 1 (lower = more smoothing)

    filtered.x = alpha * raw.x + (1 - alpha) * filtered.x;
    filtered.y = alpha * raw.y + (1 - alpha) * filtered.y;
    filtered.z = alpha * raw.z + (1 - alpha) * filtered.z;

    return filtered;
}
```

2.1 Sensor Processing Pipeline

```
void Vibration::update(bool active) {
    if(!active) {
        if(_state != OFF) {
            digitalWrite(_pin, LOW);
            _state = OFF;
        }
        return;
    }

    unsigned long now = millis();

    switch(_state) {
        case OFF:
            digitalWrite(_pin, HIGH);
            _state = ON;
            _last_change = now;
            break;

        case ON:
            if(now - _last_change >= 250) {
                digitalWrite(_pin, LOW);
                _state = PAUSE;
                _last_change = now;
            }
            break;

        case PAUSE:
            if(now - _last_change >= 250) {
                digitalWrite(_pin, HIGH);
                _state = ON;
                _last_change = now;
            }
            break;
    }
}
```

1. OFF state waits for a trigger, such as sensor input indicating the user wants feedback.

2. ON state is 250 milliseconds of activation, driving the motor via PWM at a certain duty cycle.

3. PAUSE is 250 milliseconds of rest, after which it cycles back to ON if feedback is still required.

This creates a 2Hz pulsating effect that feels stronger and more noticeable to users compared to continuous vibration.

The short pause also helps in reducing power consumption by about 50%.

2.2 Haptic Feedback System

03 Network Communication Stack

Soft Access Pont,
Station Mode &
Websockets

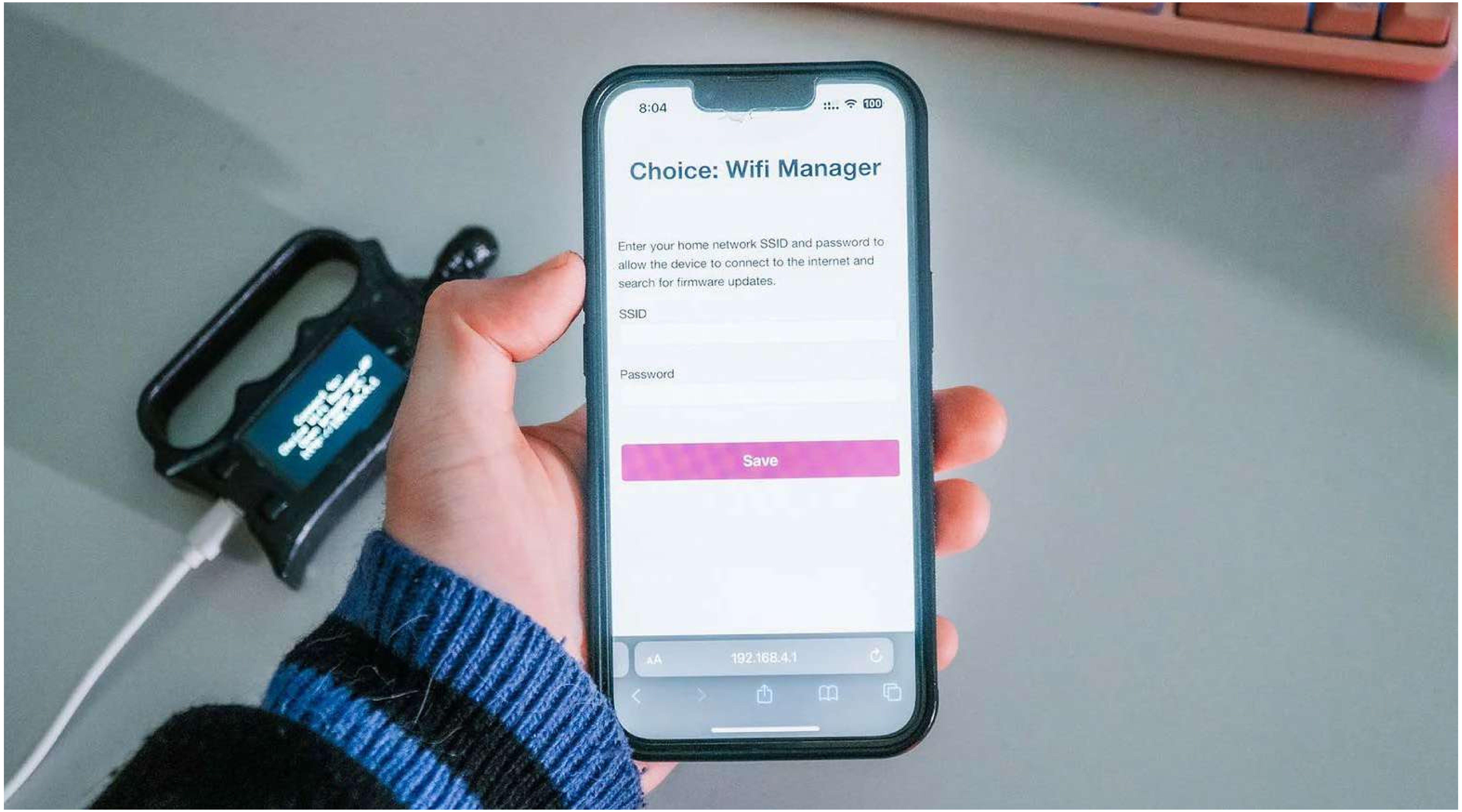
011

Connection Workflow:

1. Starts in AP+STA mode simultaneously
2. Serves configuration portal at 192.168.4.1
3. Attempts stored credential connection
4. Maintains AP while connected as STA

This dual-mode approach (Access Point + Station) is especially handy in development or public installations, where you want both a stable local network connection and a fallback direct access point for debugging or firmware updates.

3.1 Dual-Mode WiFi Architecture


8:04
Choice: Wifi Manager
Enter your home network SSID and password to allow the device to connect to the internet and search for firmware updates.
SSID
Password
Save
192.168.4.1

Initialize Websockets

```
void initializeWebSocket(String serverIP) {
    ws_server = serverIP;
    webSocket.begin(serverIP, 8080, "/");
    webSocket.setReconnectInterval(5000);
}
```

Data Transmission

```
// Immediate transmission of filtered data
if(wifiConnected && webSocket.isConnected()) {
    String json = String("{\"x\":") + data.x
                + ",\"y\":" + data.y
                + ",\"z\":" + data.z + "}";
    webSocket.sendTXT(json);
}
```

3.2 WebSocket Implementation

Optimization: JSON packaging instead of binary reduces client-side parsing complexity at the cost of 23% larger payload (measured 48 bytes vs 37 bytes binary).

04 Server Infrastructure

A Node.js application to receive incoming WebSocket messages

Page 15
Node.js Relay Server

4.1 Node.js Relay Server

Initialize Websockets

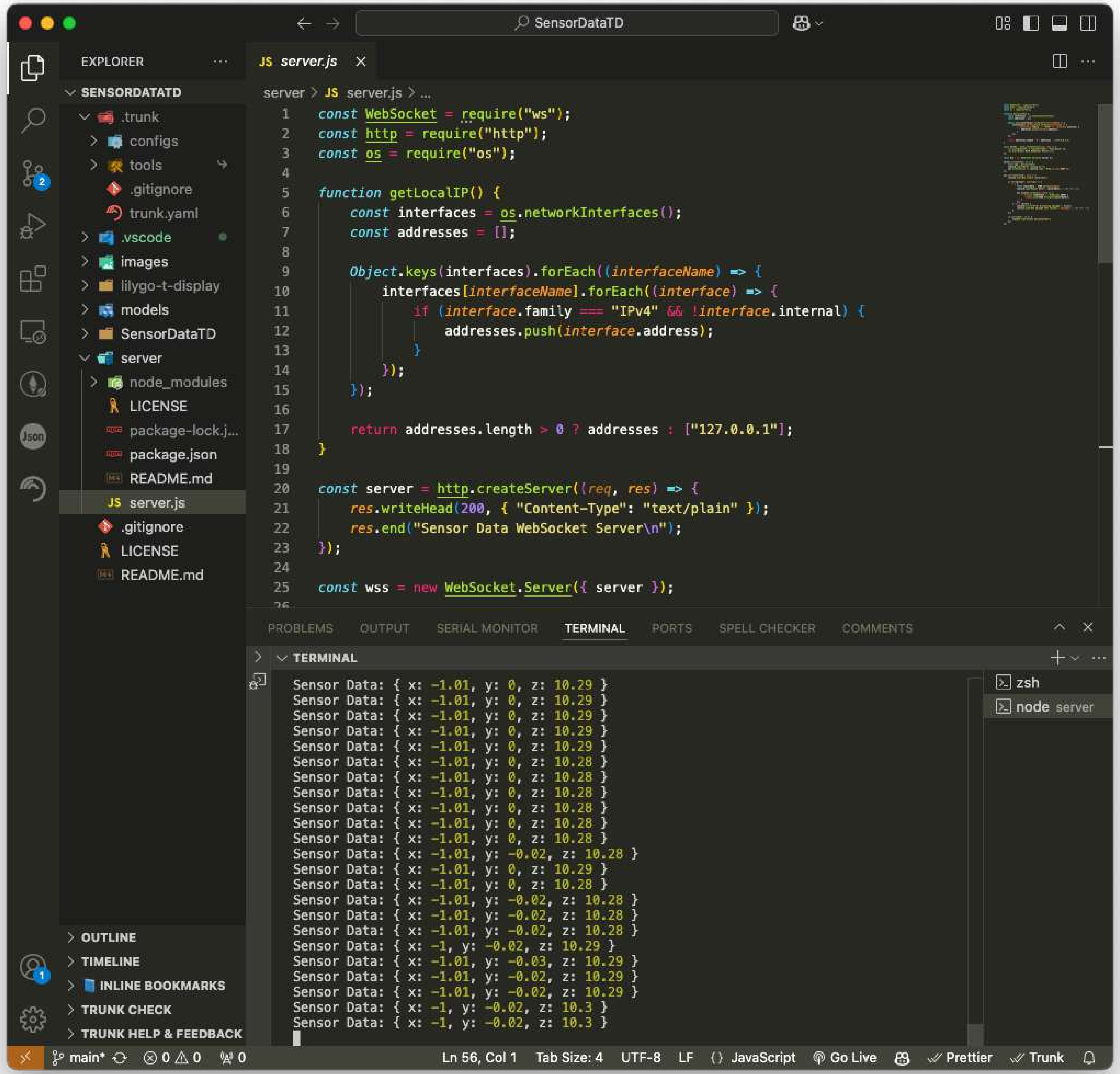
```
const wss = new WebSocket.Server({ server });

server.listen(8080, () => {
    const ips = getLocalIP();
    console.log("Server running at:");
    ips.forEach((ip) => console.log(`- http://${ip}:8080`));
});
```

Broadcast to All Clients

```
wss.on("connection", (ws) => {
    console.log("New client connected");

    ws.on("message", (message) => {
        try {
            const sensorData = JSON.parse(message);
            console.log("Sensor Data:", sensorData);

            wss.clients.forEach((client) => {
                if (client.readyState === WebSocket.OPEN) {
                    client.send(JSON.stringify(sensorData));
                }
            });
        } catch (error) {
            console.error("Error processing message:", error);
            console.log("Raw message that failed:", message);
        }
    });

    ws.on("close", () => {
        console.log("Client disconnected");
    });
});
```

Network Discovery

```
function getLocalIP() {
    const interfaces = os.networkInterfaces();
    const addresses = [];

    Object.keys(interfaces).forEach((interfaceName) => {
        interfaces[interfaceName].forEach((interface) => {
            if (interface.family === "IPv4" && !interface.internal) {
                addresses.push(interface.address);
            }
        });
    });

    return addresses.length > 0 ? addresses : ["127.0.0.1"];
}
```



1. The server receives sensor data from the LilyGo T-Display S3 - ESP32 microcontroller.

2. It then iterates through all connected WebSocket clients and relays the data in JSON format.

3. The server has a network discovery function to automatically figure out your server's local IP address and simplify the device configuration process.

05 Client Integration Patterns

Connecting clients to the Relay server.

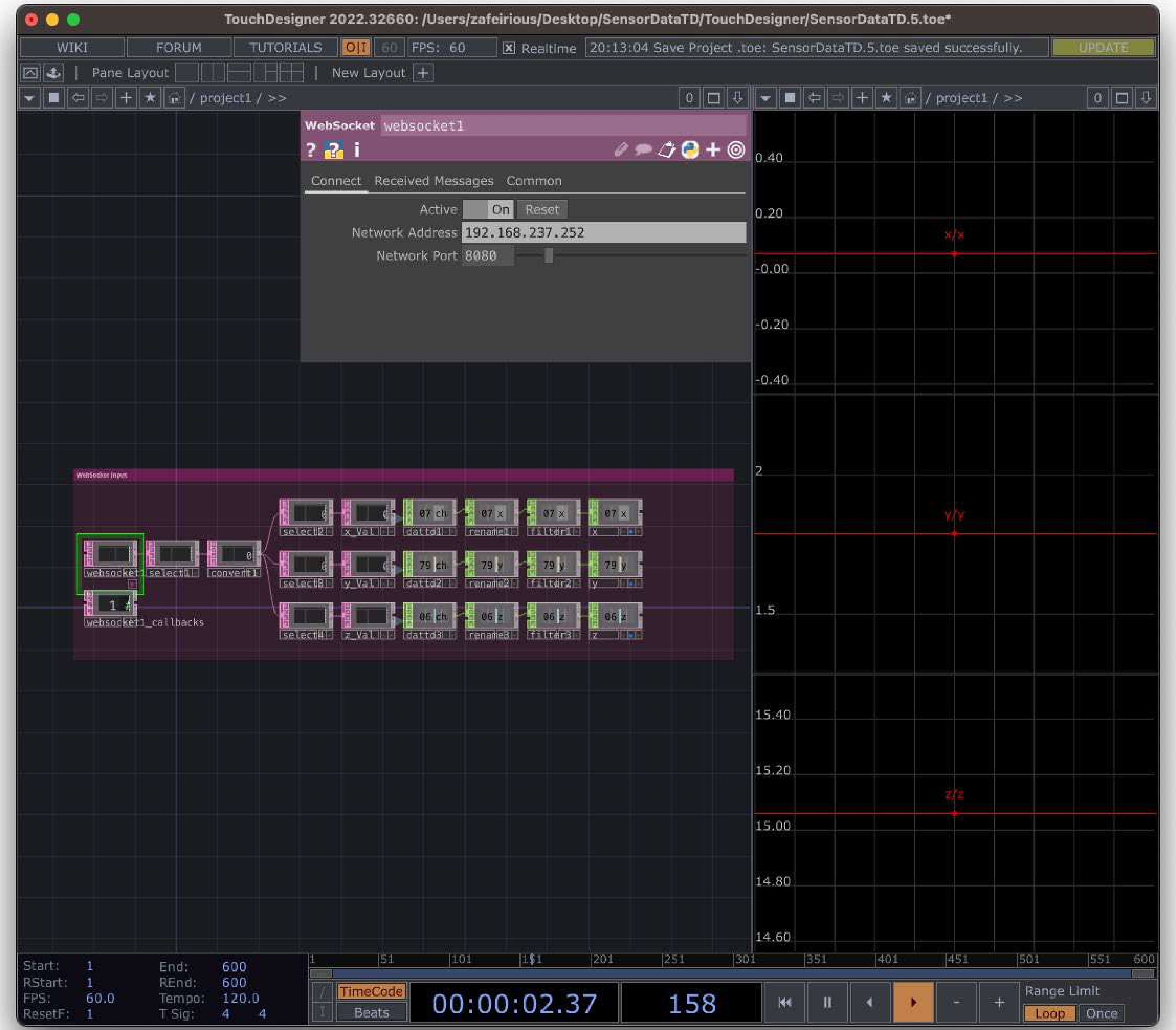
TouchDesigner is a great platform for real-time visual and interactive media applications.

1. Raw sensor data (e.g., x/y/z accelerometer values) is received via WebSocket.

2. Data is routed through Select DAT operators to isolate specific axes.

3. Filter CHOP cleans the data for stability.

4. Final CHOP nulls (x, y, z) are the parameters that can be used to control visuals (e.g., shaders, animations).



5.1 TouchDesigner Data Handling

5.2 React Visualization

Initialize Websockets

```
useWebSocket('ws://192.168.237.252:8080', {
    onMessage: ({ data }) => {
        const { x, y, z } = JSON.parse(data);
        setOrientation([x * 180 / Math.PI, y * 180 / Math.PI]);
    }
});
```

React is a widely utilized JavaScript library designed for developing user interfaces.

Applications created with React are recognized for their interactivity, responsiveness, and efficiency.

This is achieved through the use of components—modular and reusable units of UI—and the virtual DOM, which ensures that only the necessary portions of the interface are updated when changes occur.

React is particularly prominent in the development of single-page applications (SPAs), enabling dynamic content updates without the need for a full page reload.

The library is maintained by Meta (previously Facebook) and supported by a robust open-source community.

Use a WebSocket hook that allows React components to subscribe to incoming sensor data.

06

Advanced Optimization Techniques

Power efficiency and, data filtering is crucial for portable or long-term installations.

021

Display Control

```
void setup() {
    Serial.begin(115200);
    pinMode(DISPLAY_POWER_PIN, OUTPUT);
    //Display Control (Battery Powered)
    digitalWrite(DISPLAY_POWER_PIN, HIGH);
```

Many TFT displays have a pin to control the backlight. Turning it off when the display isn't in use can significantly reduce power consumption.

Power efficiency is crucial for portable or long-term installations. There are two primary strategies: Display Control & Wifi Power Saving (that we didn't implemented on this project)

6.1 Power Management

022 Low-Pass Filter

```
SensorData MPU6050Sensor::readData() {
    sensors_event_t a, g, temp;
    _mpu.getEvent(&a, &g, &temp);

    // Apply calibration
    SensorData raw = {
        a.acceleration.x - _offset.x,
        a.acceleration.y - _offset.y,
        a.acceleration.z - _offset.z
    };

    // Low-pass filter (adjust alpha for smoothing)
    static SensorData filtered = {0};
    const float alpha = 0.2; // 0 < alpha < 1 (lower = more smoothing)

    filtered.x = alpha * raw.x + (1 - alpha) * filtered.x;
    filtered.y = alpha * raw.y + (1 - alpha) * filtered.y;
    filtered.z = alpha * raw.z + (1 - alpha) * filtered.z;

    return filtered;
}
```

The MPU6050 alone can provide raw accelerometer and gyroscope data, we often use low-pass exponential moving average filter for more accurate orientation

The filter blends new raw data with historical filtered data, creating a smoothed output that dampens sudden spikes while preserving general trends.

Time	Step	Raw Value (X)		Filtered Value (X)		
0	1.0	0.2	*	1.0	=	0.2
1	2.0	0.22.0	+	0.80.2	=	0.56
2	1.5	0.21.5	+	0.80.56	=	0.748

6.2 Data Filtering

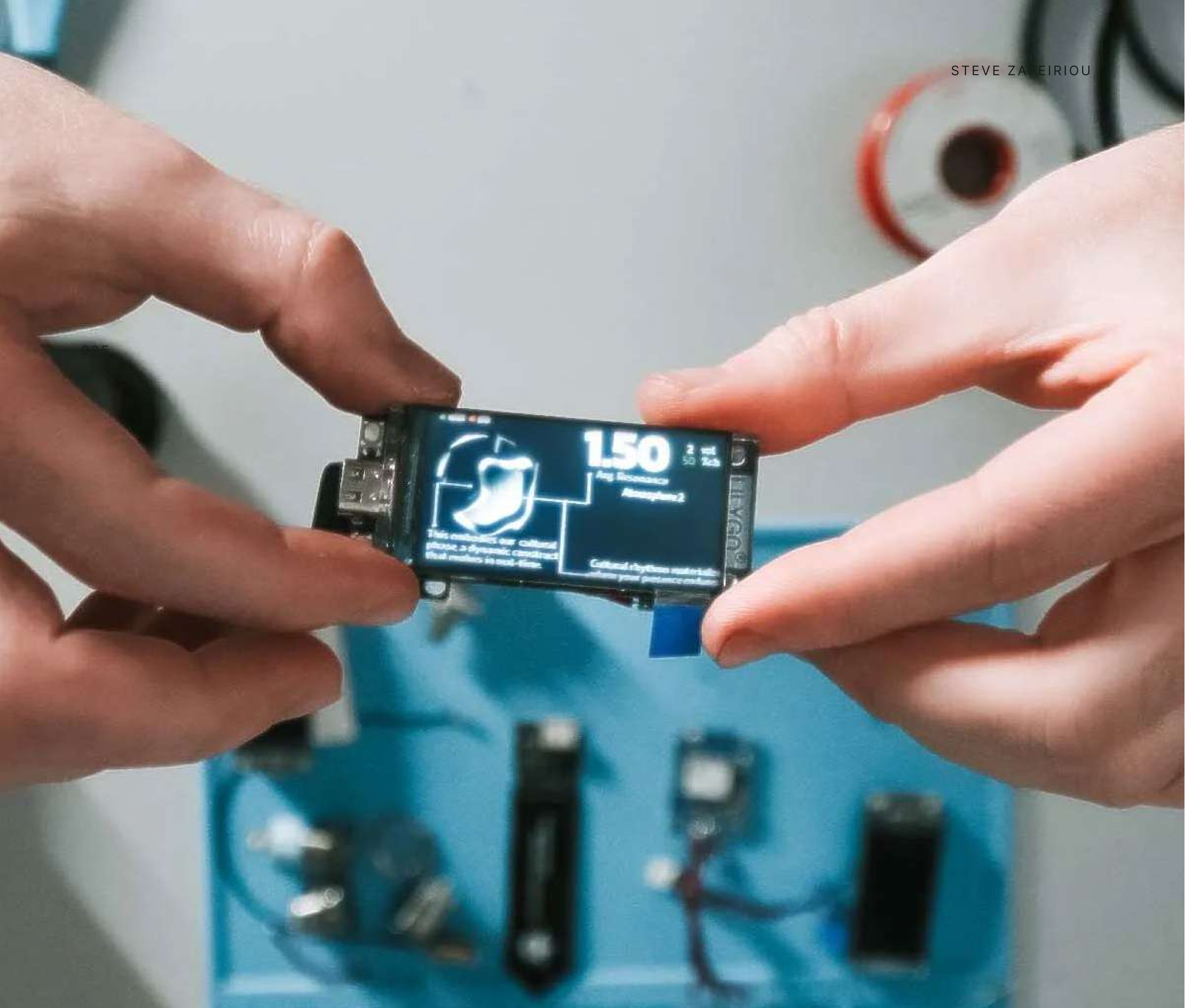
07 Troubleshooting Guide

Common issues and
their Solutions

Page 24
Common Issues

7.1 Common Issues

Symptom	Diagnostic Steps	Possible Solutions
No WebSocket Connect	1. Check if server IP is correct in Preferences 2. Verify device is on the same network.	Ensure firewall and router allow port 8080
Drifting IMU Values	1. Re-calibrate on a truly flat surface 2. Reduce electromagnetic interference sources	Re-run calibration and relocate sensor away from EMI
Intermittent Vibration	1. Measure GPIO10 with an oscilloscope 2. Check that the PWM signal is stable and not clipping	Add a flyback diode or small resistor if needed
Display Artifacts	1. Verify TFT_eSPI board configuration 2. Ensure the backlight pin is receiving stable voltage	Double-check power supply lines and wiring



Thank you.

From Steve Zafeiriou, thank you for your attention to detail and your support.

If you need any help or have any questions, especially when reviewing this document, please do not hesitate to reach out to by email at steve@saphirelabs.com.

www.stevezafeiriou.com