# ThermoView

## Smart Temperature Detector for Electronic Devices

Yongzhe Jiang

GitHub Link: https://github.com/cnzhe/TECHIN_514_Project

TECHIN 514 Final Project

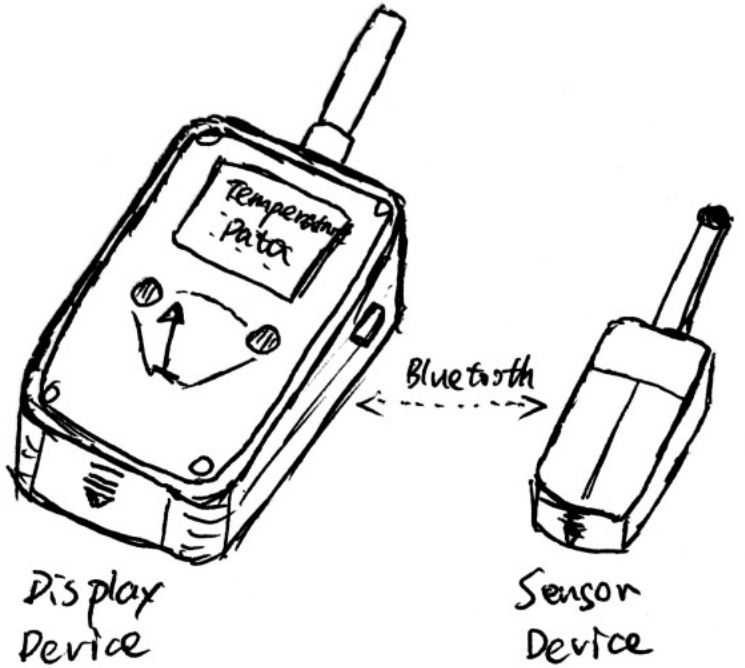
# Introduction

ThermoView - Smart Temperature Detector for Electronic Devices is designed to measure the temperature change of all electronic devices - whether they are in proper operating condition or not.

However, measuring temperature is not always convenient, for example in situations such as using a gaming laptop, where getting the temperature of the rear exhaust fan in real time and displaying it is challenging. This highlights the importance of modular sensor and display components.

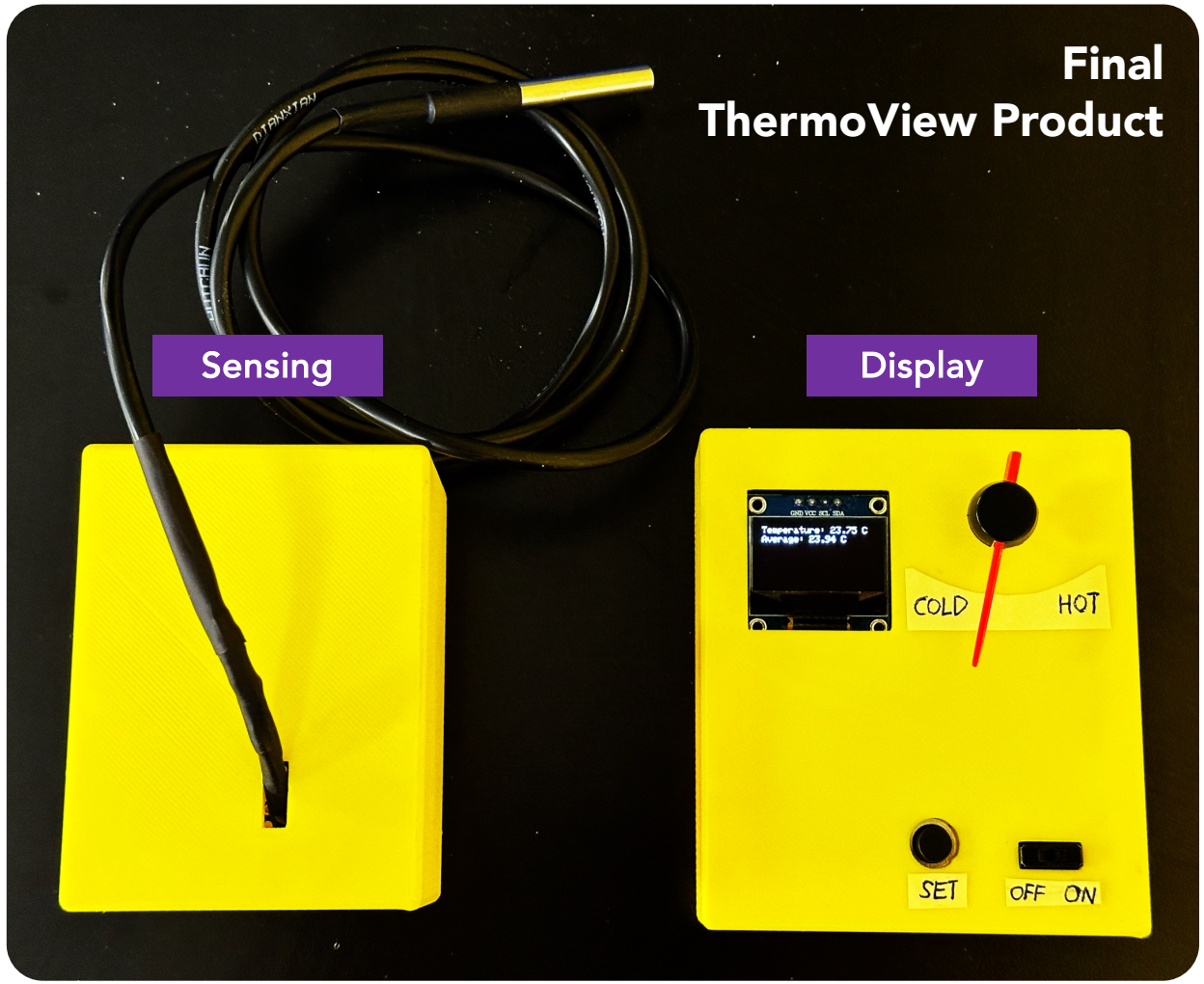
By separating the temperature sensing part and the display part and transmitting the data via Bluetooth signals, they are made into a set of connected devices. The sensor device can be conveniently placed near the device under test, while the display device is located within visual range. In addition, a DSP algorithm will be utilized to identify the normal temperature of the electronic device under operating conditions to avoid anomalies.

The device is also visually impaired friendly and can avoid burns caused by high temperature devices. My solution improves the accessibility and usability of temperature measurement, ensuring a seamless monitoring experience for electronic devices.




Original Sketch

# Proposed Solution



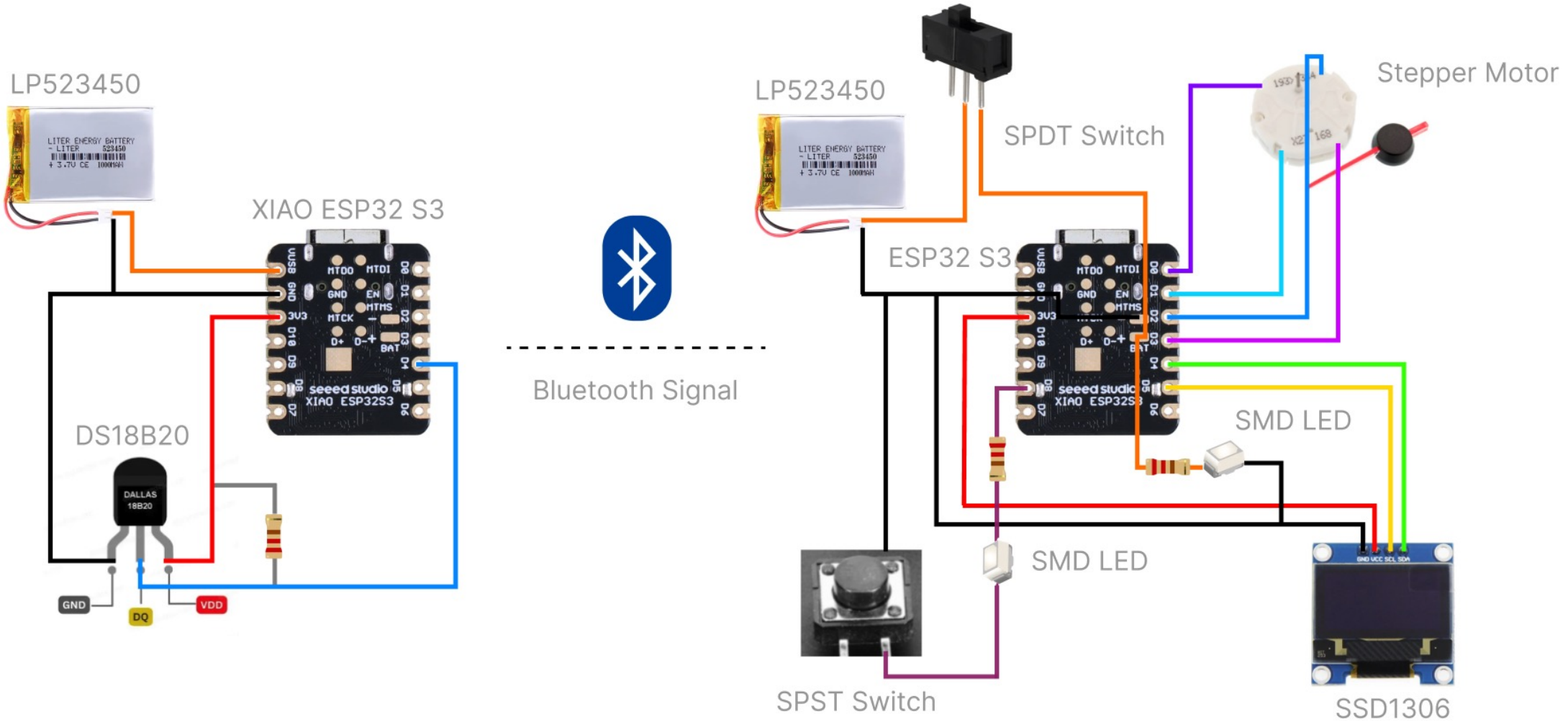

The sensing device has

- **a temperature sensor** (DS18B20)
- a **Bluetooth** wireless link to the display device
- **a sustainable power solution** (1000mAh rechargeable Lithium Polymer battery - LP523450)
- **a custom designed enclosure** (fit well with the PCB and interaction components using 3d modeling and printing)

The display device has

- **a stepper-motor-driven gauge needle** (Indicates the current state of the device temperature, the current setting is that the temperature is higher than 30 degrees will be marked as HOT, less than or equal to 30 degrees will be marked as COLD)
- **two LEDs** (One LED is used to show that the power supply can power the MCU; the other LED shows that the MCU can power other components)
- **two button/switch** (SPDT switch is to change the power mode, and SPST button is to allow the LED to light up, proving that the MCU can power other components)
- **a custom-designed enclosure** (fit well with the PCB and interaction components, using 3D modeling and printing)
- **a sustainable power solution** (1000mAh rechargeable Lithium Polymer battery - LP523450)

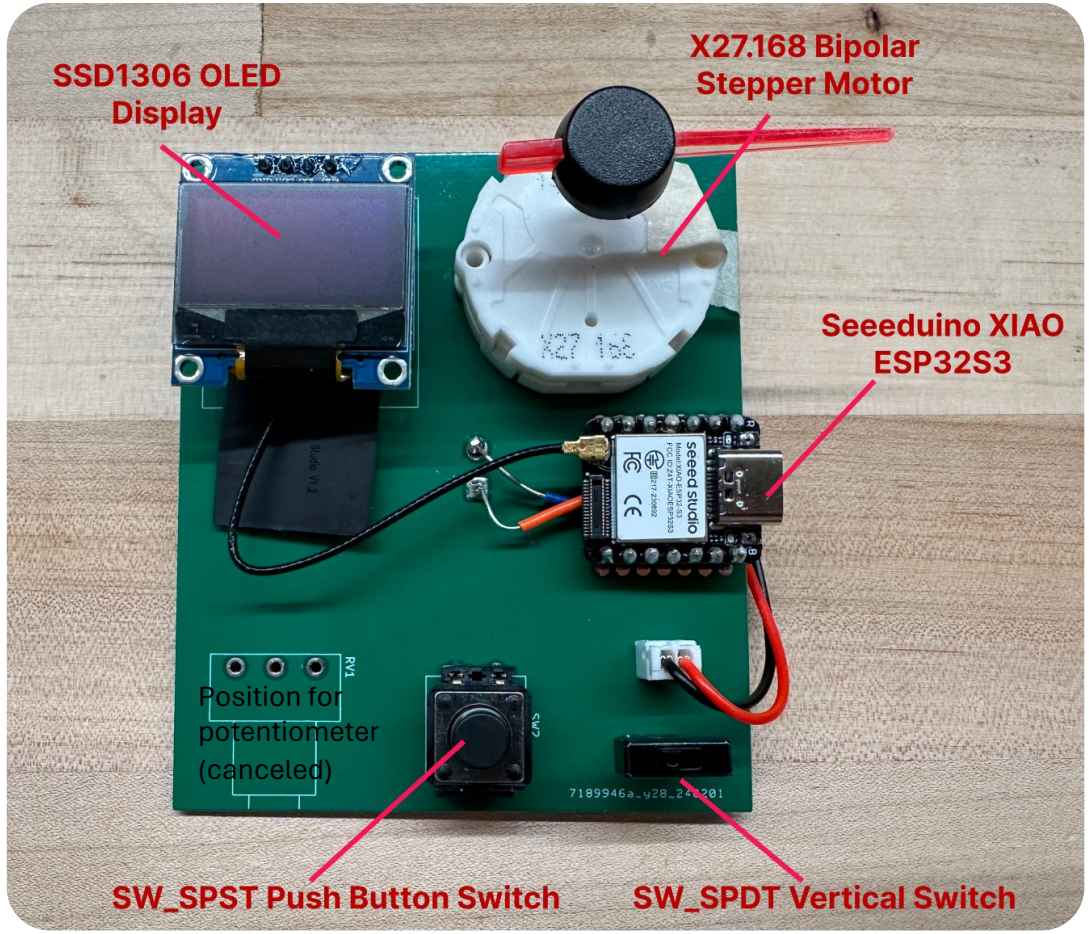
# System Architecture



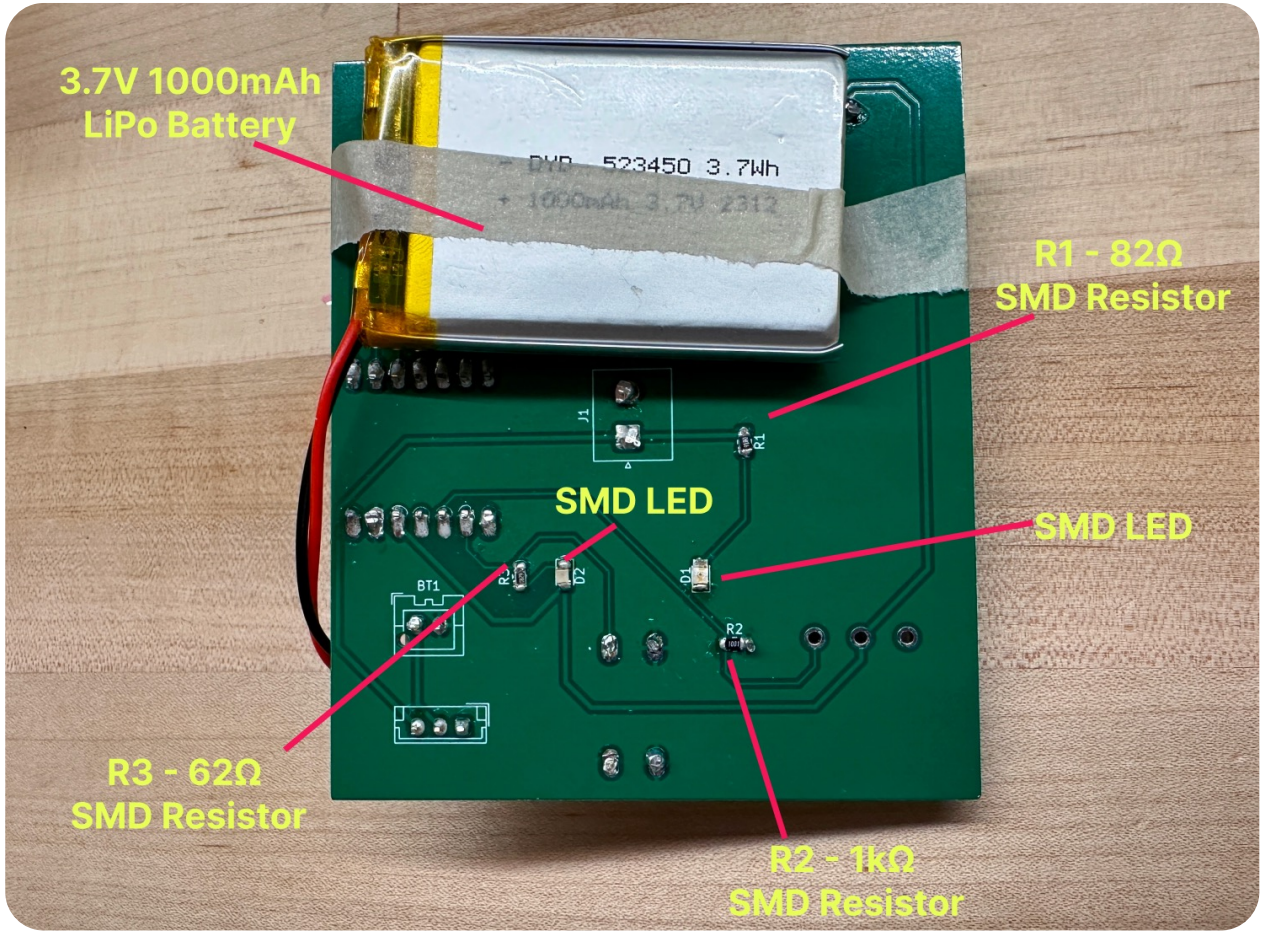

Sensing

Display

# Pictures of Critical Components – Display Device

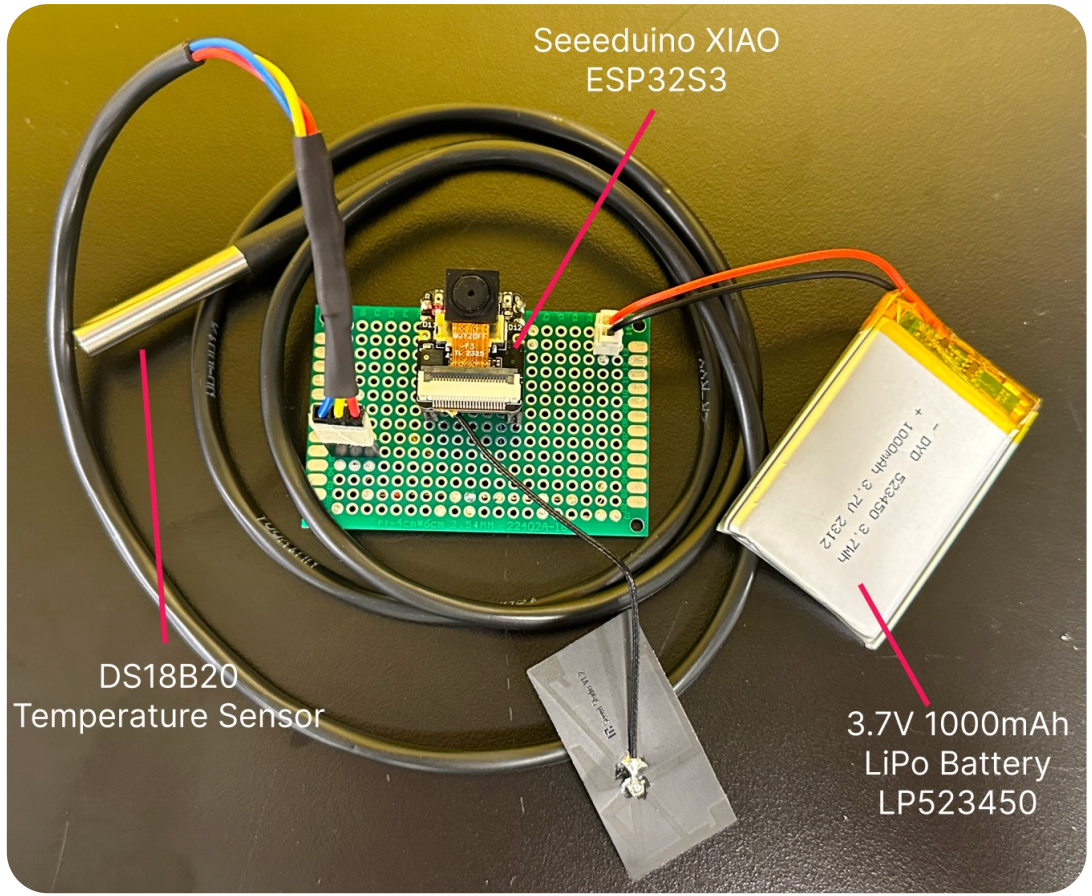



Front

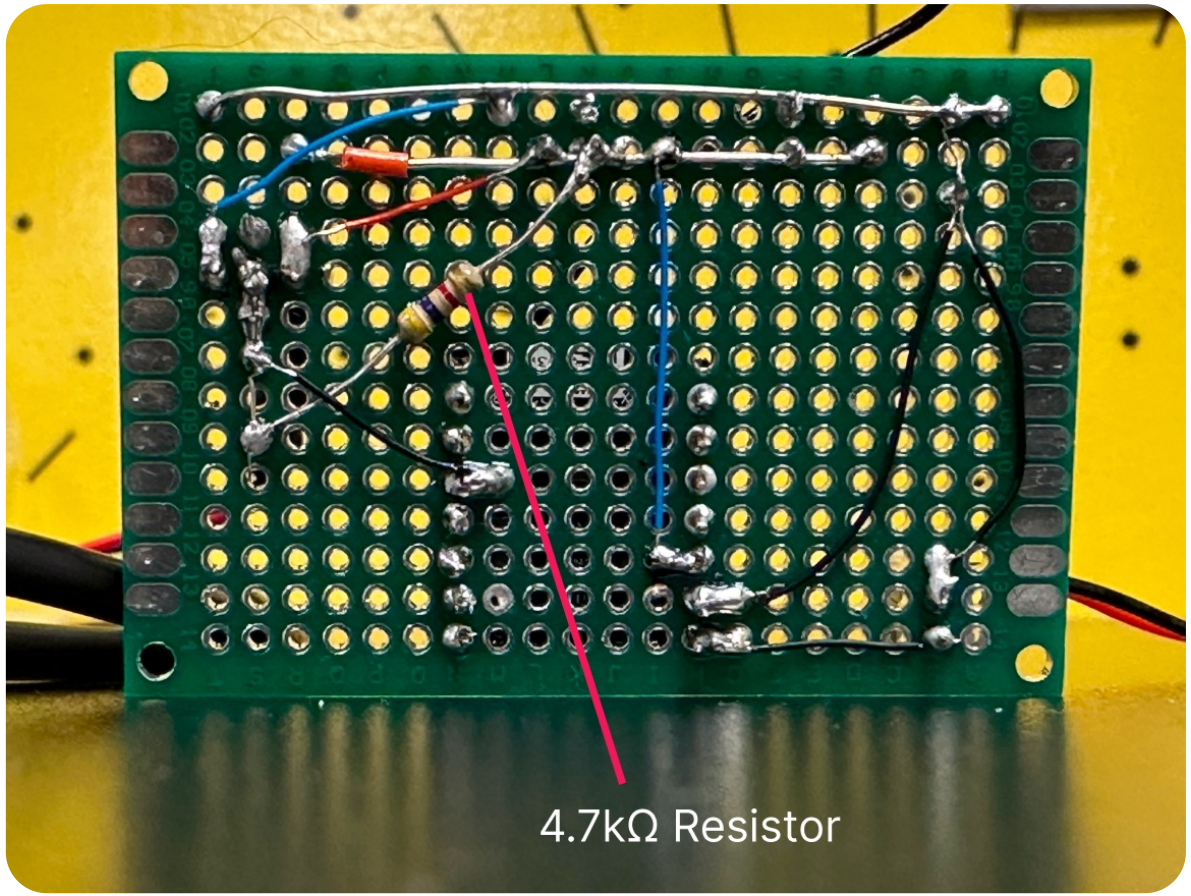



Back

# Pictures of Critical Components – Sensing Device



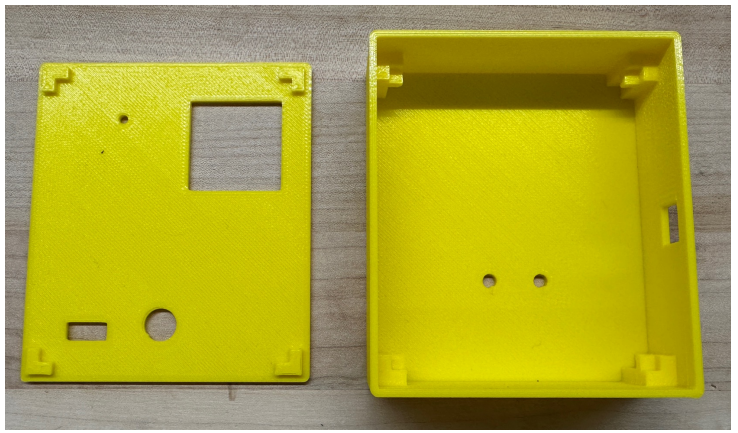

Front

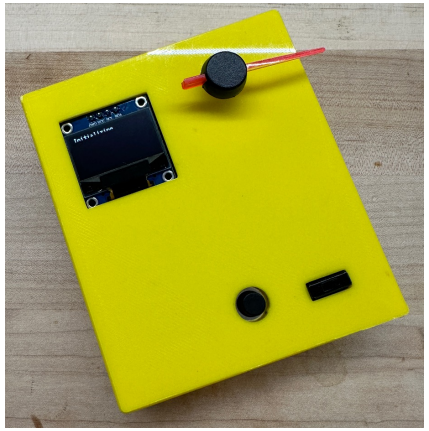



Back

# Pictures of Critical Components – Enclosure

Display



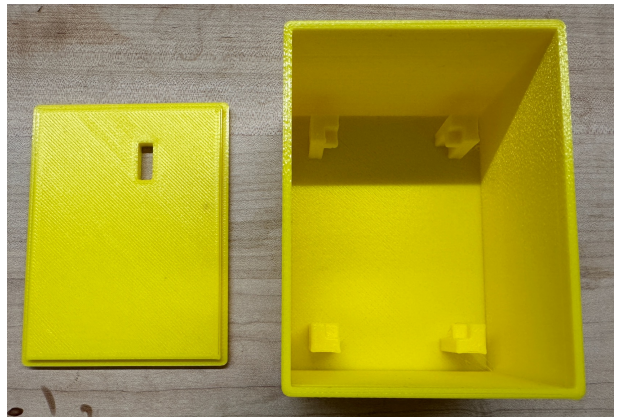


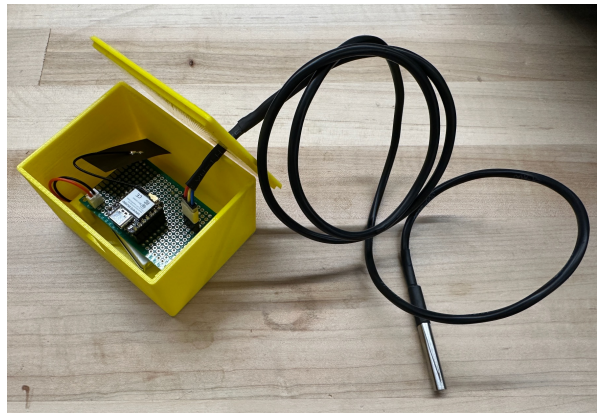
3D Modeling

3D Printing

Assembly Testing

Sensing





# Screenshots and Description of Signal Processing

## Key DSP Code in Sensing

```
// DSP Algorithm to calculate real-time average temperature
float calculateAverageTemperature(float newTemperature) {
    temperatureSum += newTemperature;
    readingsCount++;
    return temperatureSum / readingsCount;
}
```

```
void loop() {
    if (deviceConnected) {
        // Read temperature from DS18B20 sensor
        sensors.requestTemperatures();
        float temperatureCelsius = sensors.getTempCByIndex(0);

        // Calculate real-time average temperature using DSP algorithm
        float averageTemperature = calculateAverageTemperature(temperatureCelsius);

        // Convert temperatures to strings and update BLE characteristic
        char tempString[15];  // Buffer to store current and average temperatures as a string
        snprintf(tempString, sizeof(tempString), "%.2f, %.2f", temperatureCelsius, averageTemperature);
        pCharacteristic->setValue(tempString);
        pCharacteristic->notify();

        Serial.print("Notify values: ");
        Serial.println(tempString);
    }
```

In the signal processing section, I process the real-time acquired temperature data on the sensing device. The goal is to obtain a temperature average. In this way the average temperature of the detected electronic device in a long term operating condition can be obtained at the same time.

**Algorithm:** The DSP algorithm accumulates temperature readings and counts, updating the running sum and count with each new reading. The average temperature is then computed by dividing the sum by the count.

**Data Merge to Send:** After obtaining the current temperature from the DS18B20 sensor, the algorithm formats it as a string. This string is then combined with the average temperature (calculated by the DSP algorithm) into a single string. The combined string is sent as a parameter to the display device.

# Screenshots and Description of Signal Processing

## Interpretation of signal-processed data in Display

```
void notifyCallback(
    BLERemoteCharacteristic* pBLERemoteCharacteristic,
    uint8_t* pData,
    size_t length,
    bool isNotify) {
    // Extract current and average temperatures from the received data
    float currentTemperature = 0.0;
    float avgTemperature = 0.0;
    if (sscanf((char*)pData, "%f, %f", &currentTemperature, &avgTemperature) == 2) {
        updateDisplayAndMotor(currentTemperature, avgTemperature);
    } else {
        Serial.println("Invalid data format");
    }
}
```

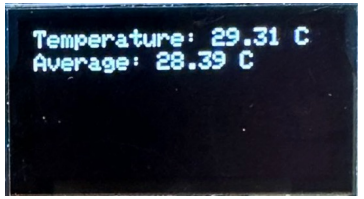
```
void updateDisplayAndMotor(float currentTemp, float avgTemp) {
    display.clearDisplay();
    display.setTextSize(1);
    display.setCursor(0, 0);
    display.print("Temperature: ");
    display.print(currentTemp, 2);
    display.print(" C");

    display.setCursor(0, 10);
    display.print("Average: ");
    display.print(avgTemp, 2);
    display.print(" C");
```

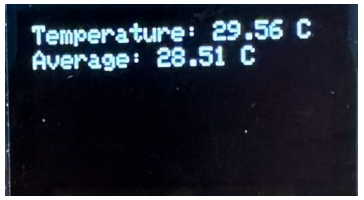
Data Interpretation: On the display device, the *notifyCallback* function interprets the received data. It extracts the current temperature and the average temperature in the combined string. The extracted temperatures are then used to update the SSD1306 display and control the motor.

For accuracy, the average results were very precise and exactly as expected.

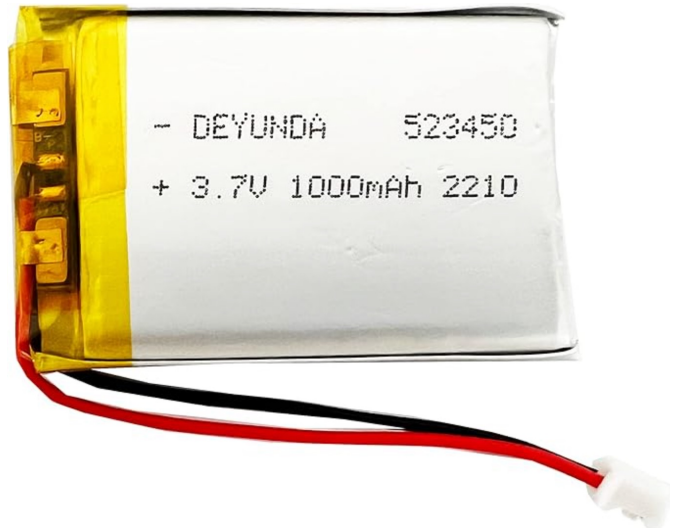
## Actual Results on SSD1306 Display

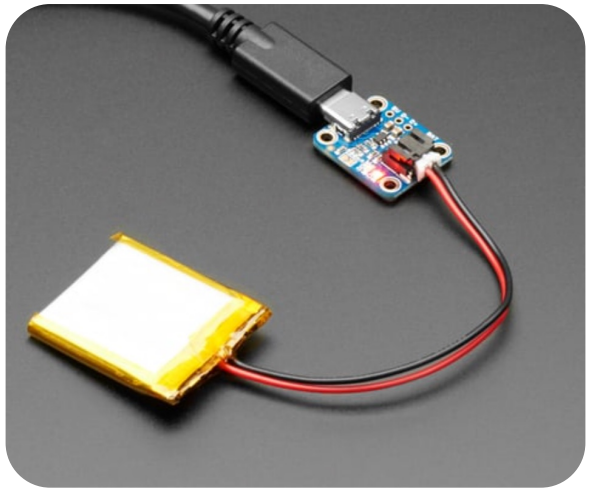
The current and average temperature data updates successfully and correctly.



# Description of Battery Considerations



## Rechargeable vs Disposable Batteries

I selected a 1000mAh rechargeable Lithium Polymer battery - LP523450 because it offers high energy density, low self-discharge, and up to 500 recharge cycles. This allows for smaller form factor and lower long-term costs compared to disposables.

## Capacity and Lifespan

The two devices use two 1000mAh batteries, enabling a very long time of operation. It provides a low battery indicator.

## Charging Method

Simply plug it via any USB C cable into a USB port and a 3.7V/4.2V lithium polymer or lithium ion rechargeable battery into the JST plug on the other end.

## Charging Method

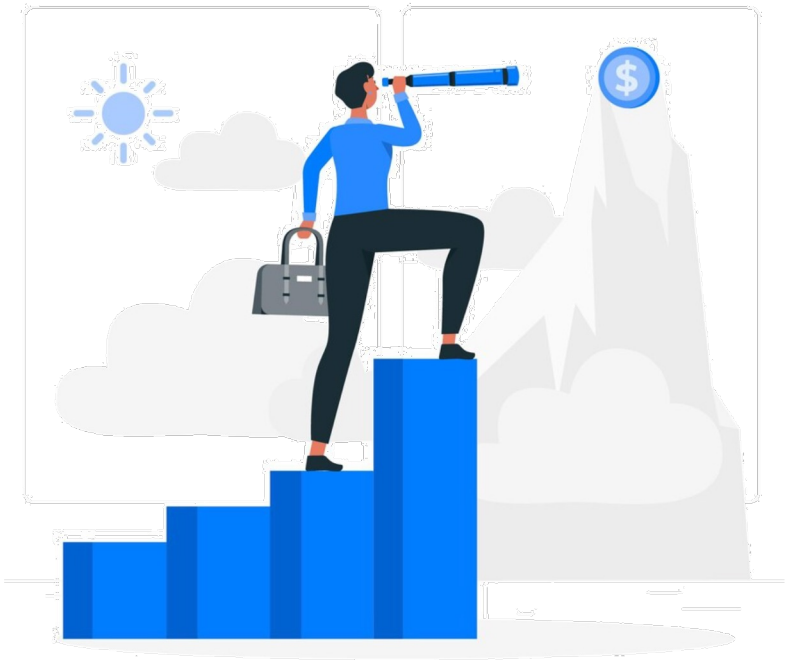
Components were selected for low power consumption. Software features like adjustable sample rate and sleep mode maximize battery efficiency.

# Budget Summary

| Item | Quantity | Unit Price | Total Price |
|---|---|---|---|
| SSD1306 OLED | 1 | $6.99 | $6.99 |
| Push switch B3F | 1 | $0.38 | $0.38 |
| SPDT Slide Switch | 1 | $0.95 | $0.95 |
| SEEED STUDIO XIAO ESP32S3 SENSE | 2 | $13.99 | $27.98 |
| LiPo Rechargeable Battery 1000mAh 3.7V | 2 | $8.99 | $17.98 |
| USB-C Data Transfer Cable (USB C to USB C) | 2 | $3.92 | $7.84 |
| DS18B20 Digital temperature sensor + extras | 1 | $3.95 | $3.95 |
| Waterproof 1-Wire DS18B20 Digital temperature sensor | 1 | $9.95 | $9.95 |
| Micro-Lipo Charger for LiPoly Batt with USB Type C Jack | 1 | $5.95 | $5.95 |
| **Total** | / | / | **$89.81** |

# Future Work

- Consider to add additional sensor units
- Experiment with thermal imaging modules
- Experiment with more enclosure manufacturing processes
- Software support for custom analytics and reports
- More iterations of the product version
- Add the camera/speaker to recognize the specific device



# Working Devices

Please see my successful live demo of my fully functional devices in class. Thank you!