Modular Front-End Framework for Agricultural Sensors

KIRAN M V
Roll No: 122101019
IIT Palakkad
122101019@smail.iitpkd.ac.in

Smart AgriTech Centre for Advanced Research and Development

This report submitted in partial fulfilment of the requirements for the Internship program, Smart AgriTech Centre for Advanced Research and Development. This project has been carried out with the guidance of Mr Narayanan P P and Mr Bavesh Ram.

July 17, 2024

Abstract

This project focuses on the development of a modular front-end framework for agricultural sensors, focusing on the integration of a capacitive-based leaf wetness sensor into a standalone system. Accurate monitoring of leaf wetness is essential for effective crop management and this system aims to provide reliable and autonomous data collection. The framework incorporates a Raspberry Pi Pico microcontroller, a solar-powered battery management system (including a solar panel, solar charge controller and Li-ion batteries). An OLED display, an SD card and an RTC module contribute in acquiring, processing, displaying and storing sensor data. The system addresses key challenges such as ensuring continuous power supply, handling real-time data efficiently and operating independently in diverse environmental conditions. Testing confirms the module's ability to function autonomously while providing accurate and reliable data, thus supporting improved decision-making in agricultural management.

Contents

1. Introduction

1.1 Background

In agri-tech, accurate and timely data collection is essential for effective crop management. Sensors play a crucial role in monitoring various environmental parameters such as leaf wetness, soil moisture, temperature and evaporation, which directly impact plant health and growth. Integrating the agricultural sensors into a standalone module presents several challenges, including ensuring a consistent power supply, processing and displaying data in real-time and storing data for future analysis. This project aims to address these challenges by developing a versatile, standalone module that integrates the sensor with a front-end processing and display system

1.2 Objective

The primary objective of this project is to develop a modular front-end framework for agricultural sensors that integrates a leaf wetness sensor with a standalone front-end setup. Leaf wetness is a critical factor, as it influences the likelihood of fungal diseases and pest infestations. The specific goals of the project are as follows:

- Sensor Data Acquisition
- Data Processing
- Data Display
- Data Storage
- Power Supply and Management
- Standalone Operation
- Testing

2. Methodology

The design for the project involves developing, testing and validating the system. Our requirement is a standalone module where a microcontroller processes the data sent from the sensor, displays it on an OLED screen, and saves it on an SD card. For this, a well-researched plan for implementation is required. After selecting the components and constructing a block diagram, designing a PCB for the circuitry will make the system more stable and robust.

2.1 System Architectural Design

2.1.1 List of components

The list of components used are as follows:

Table 2.1: List of Components

Electronic Components	Specification	Description
Raspberry Pi Pico	RP2040 Processor	Microcontroller
Solar Power Manager	5V 1A Output	Directs solar power to the system
Leaf Wetness	7 V	Capacitive Sensor
Lithium Ion Battery	3.7 V; 5000mAh	Backup storage
Solar Panel	6 W (6V, 1A)	Harvests solar energy
DC - DC Buck Convertor	5 V to 8 V	Provides higher voltage to components
I2C OLED Module	3 V	Displays processed data
SPI Micro SD Card Reader	5 V	Stores processed data
RTC Module	3 V	Provides real time timestamps

2.1.2 Block Diagram

The detailed block diagram of the project is given below:

- The entire setup is powered either by a solar panel or Li-ion batteries.
- The power from these sources is managed by a solar charge controller, which, with the help of an in-built linear regulator, provides a continuous and stable 5V, 1A output power supply. This ensures a reliable power source for the connected devices.
- The 5 V output is used to power up the Raspberry Pi Pico module.

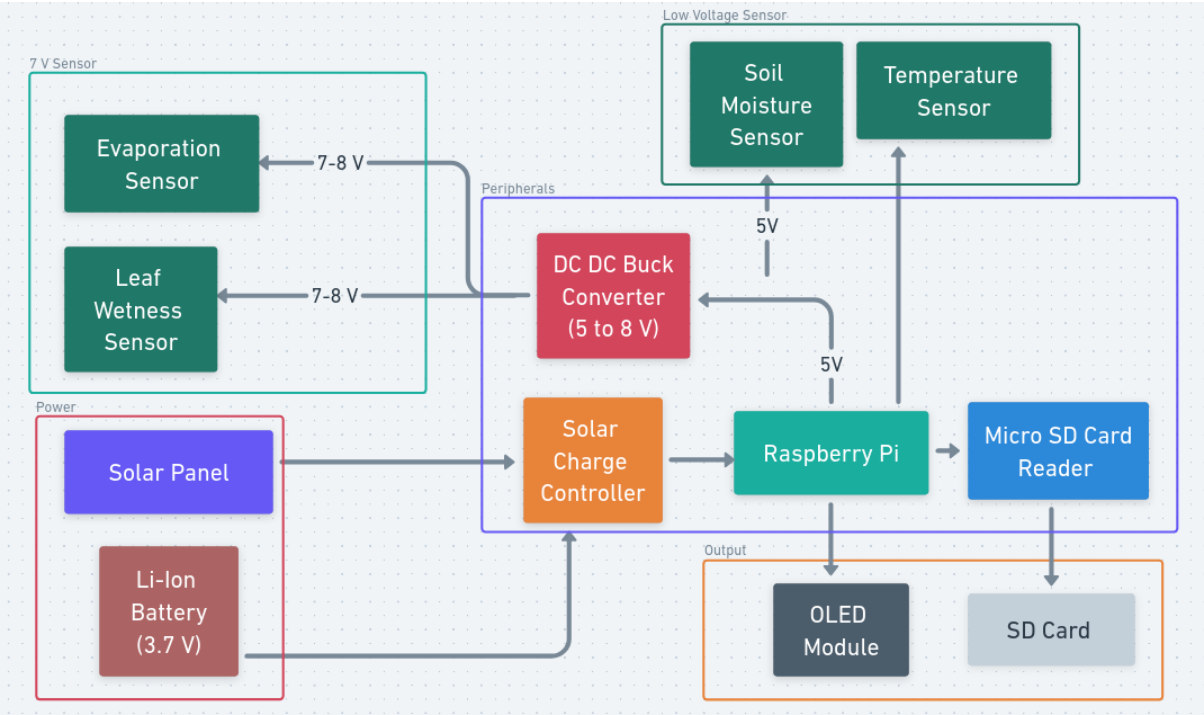

Figure 2.1: Block diagram of the setup

- We have got four sensors in the setup. Leaf wetness sensors and evaporation sensors are powered by 7V, soil moisture sensor uses 5 V and temperature sensor is just a resistive sensor, but uses 3.3 V for voltage divider.
- Since the Raspberry Pi Pico can only provide up to 5V, a DC-DC buck-boost converter is incorporated to convert 5V to 8V. It is then used to power sensors requiring 7V.
- An OLED display is used to show the data from the sensors
- An SD card, accessed through an SPI Micro SD Card Reader Module, is used to store the collected data with timestamps.

2.2 Software Development

First we tried and practiced different ways of processing data, in order to get in familiar with Raspberry Pi Pico. All of the signals used during coding is from digital storage oscilloscope.

2.2.1 Measuring the on-time of a PWM wave

The aim is to measure the on-time (high pulse duration) of a PWM (Pulse Width Modulation) signal on a pin 15 of a Raspberry Pi Pico. After importing required modules and initializing the PWM pin, a function is defined to measure the on time of the PWM signal. The function initializes on time to 0. It checks if the pin is at a zero logic level. If so, it waits until the pin goes high and records the start time. Then, it waits until

the pin goes low again and records the end time. The on time is calculated as the difference between end time and start time. For more precise results, the values of on time is averaged over a 1000 samples.

```
On_Time = 603
On_Time = 603
On_Time = 598
On_Time = 599
```

Figure 2.2: On-time for 1KHz at 60% duty cycle. Data in ms

2.2.2 Interfacing Raspberry Pico with OLED display, SD card reader and RTC

The next step was to start with the front end display and storage. Here we log the data, along with a timestamp, display the information on an OLED screen and save it to an SD card. SPI communication is set up for SD card handling and I2C communication is established for the OLED display and the real-time clock (RTC) is initialized for timestamping. The current date and time are obtained from the RTC and formatted for display and logging. Even if the system got powered off, the battery inside the RTC module will help in running the system clock inside the module. The OLED screen is updated with the current date and time. The data is appended to a file on the SD card, with a short delay before the next measurement cycle. So this code creates a basic data acquisition system.

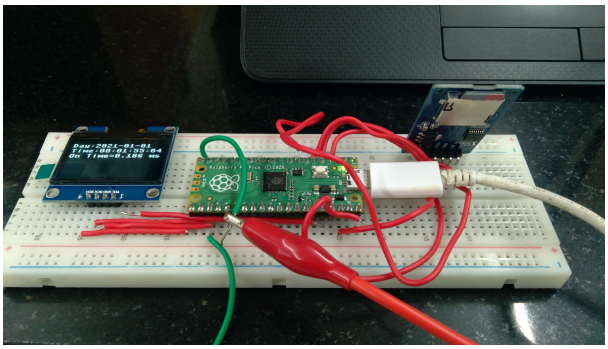

Figure 2.3: Raspberry Pico with OLED display and SD card reader

2.2.3 Measuring the voltage of a PWM signal

Here we read the analog voltage values from the signal from an ADC (Analog-to-Digital Converter) pin on the Raspberry Pi Pico. It is of 16 bits. A 16 bits ADC has 65536 quantization levels. So we will take the 16 bit data, averages it 100 times and will map it to 3.3 V.
Specifically for leaf wetness sensor, the data wire of the sensor is connected to ADC (2) pin of the Pico and the output is coming in between 0 to 65535 (0 to $2^{16}-1$). This bit data is then mapped to 0 to 3.3 V, as done in the code (Figure 2.4). So the output will range

```
def read_voltage():
    total = 0
    for _ in range(100):
        total += pin.read_u16()

    average = total / 100
    voltage = average * (3.3 / 65535)
    return voltage, average
```

Figure 2.4: Code for reading voltage from leaf wetness sensor

from 0 to 3.3 V, which can be easily calibrated and classified. This code provides real-time feedback on the sensor's voltage, continuously monitoring with a 1-second interval.

2.3 Battery Management System (BMS)

The Battery Management System (BMS) is a critical component of the framework system, ensuring efficient and reliable power management. It integrates three primary elements: a solar panel, a solar charge controller and Li-ion batteries. The solar panel captures energy from sunlight, which is regulated by the solar charge controller to safely charge the Li-ion batteries. These batteries store the harvested energy, providing a stable and continuous power supply to the system. This configuration enables autonomous and uninterrupted operation of the sensor module in remote agricultural environments, facilitating consistent data collection and processing.

2.3.1 Solar Power Manager

The solar power manager is one of the main component of the system. It incorporates an MPPT (Maximum Power Point Tracking) function that adjusts the load to extract the maximum available power from the connected solar panel, helping to maximize the efficiency of the generation. The module can charge a 3.7V Li-ion battery at up to 900mA of current, allowing for efficient storage of the generated solar power.

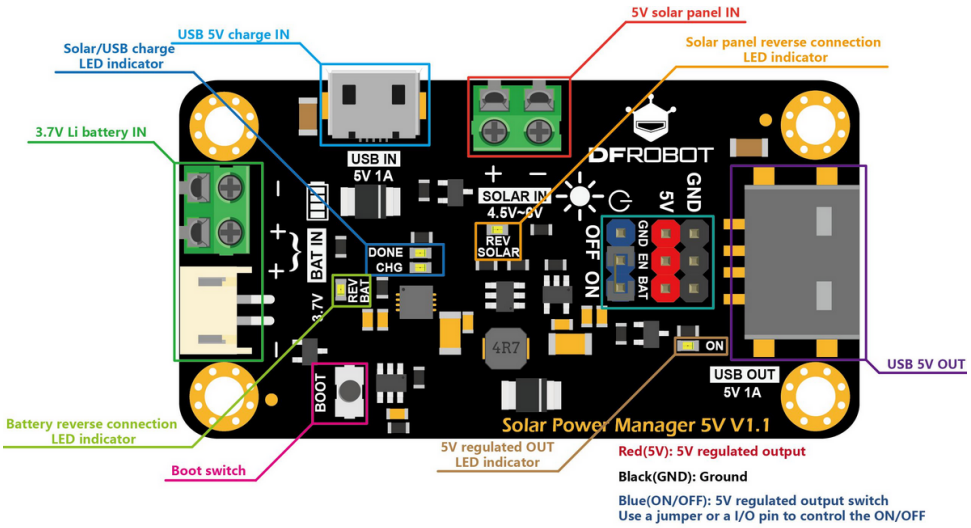

Figure 2.5: Solar Power Manager

The solar power manager also provides a regulated 5V, 1A output with help of an in-built linear regulator. It can be used to power various low-power components and peripherals in the project. This ensures a stable and reliable power supply for the connected devices. Furthermore, the module includes various protection features, such as over-charge, over-discharge, and short-circuit protection. It was found that the solar power manager is a well-designed and reliable component that integrated seamlessly into the project, playing a key role in the successful implementation of the solar power management system. The datasheet of the solar power manager is here

2.3.2 Solar Panel

The solar panel chosen is featuring an operating voltage of 5V and a maximum power output of 6W. This panel is capable of delivering up to 1A of current. Its flexible PET package and monocrystalline silicon construction ensure durability and performance in various lighting conditions. With dimensions of 27.5cm x 16cm x 0.2cm and a weight of 90g, it is an ideal choice for portable and remote applications. The datasheet of the solar panel is here

2.3.3 Li-Ion Batteries

3.7 V Li-ion batteries of various capacities have been used throughout this project for various studies. These rechargeable batteries are known for their high energy density, lightweight construction and long lifespan. They efficiently store energy harvested by the solar panel, which is then regulated by a solar charge controller. We have done our studies using 1200mAh, 2500 mAh and 3000 mAh Li-ion batteries. Our studies indicate that increasing the battery capacity results in extended operational life of the system.

2.4 Hardware Integration

2.4.1 Raspberry Pi Pico

The Raspberry Pi Pico is a low-cost, high-performance microcontroller board based on the RP2040 chip. It features a dual-core ARM Cortex-M0+ processor running at 133MHz, 264KB of SRAM and 2MB of onboard flash memory.

With 26 multi-function GPIO pins, including I2C, SPI and UART interfaces, the Pico offers versatile connectivity options. Its small form factor, energy efficiency and robust performance make it ideal for a variety of applications, including sensor integration and data processing in the project.

2.4.2 Leaf Wetness Sensor

The leaf wetness sensor is a capacitive-based device designed to measure moisture levels on leaf surfaces. It provides an analog voltage output that corresponding to the amount of moisture detected. The sensor operates using a 3-core wire configuration, with connections for power, ground and data. It requires a 7V power supply for proper operation. Whenever there is some wetness or water in the surface of the sensor, there will be change in the dielectric constant in the surface. This change in dielectric results in change in

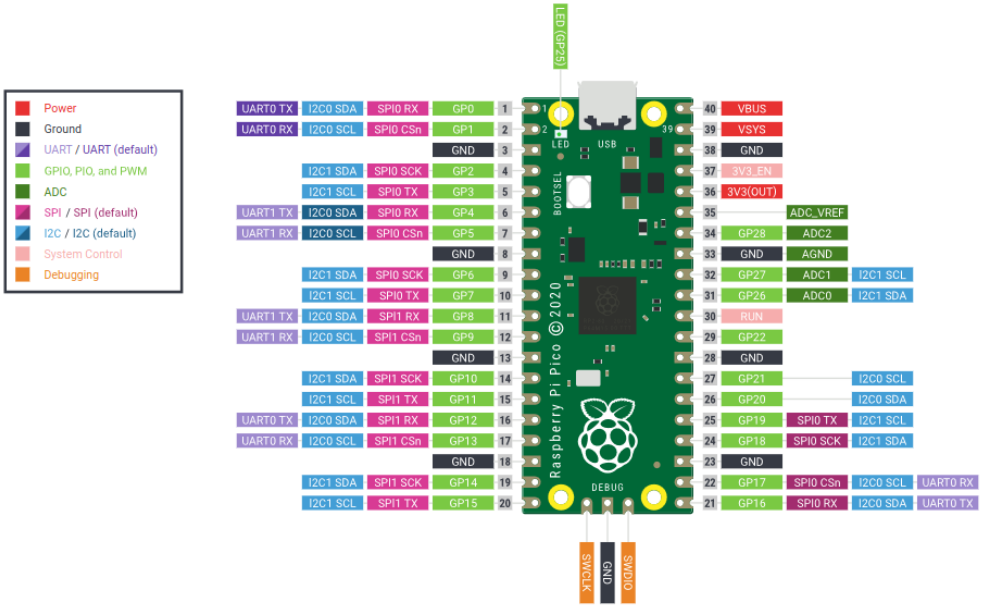

Figure 2.6: Raspberry Pi Pico Pinout. Source: raspberrypi.com

capacitive. With the help of an internal oscillator circuit, the capacitance change is outputted as a voltage change.

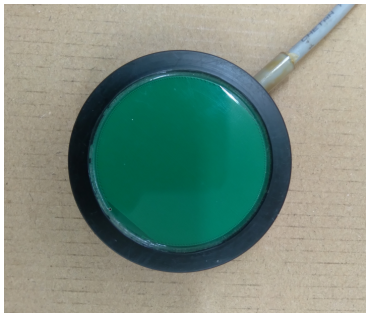

Figure 2.7: Leaf Wetness Sensor

Calibrating the sensor

To ensure accurate calibration of the capacitive-based leaf wetness sensor, it is crucial to measure the input voltage and map it to appropriate threshold values. For simulation, I created a wetness environment using a water sprayer.
The voltage values are recorded for corresponding increases in wetness and a plot of voltage versus percentage wetness is created to visualize the relationship.

As you can see from the graph (Figure 2.9), we have got a good linear relationship between the voltage vs wetness percentage.
Threshold levels are established to classify the voltage readings into corresponding wetness percentages. The threshold levels were found out by the manual inspection of wetness in over the sensor. The dry and wet values in the code are calibration constants representing the ADC values for completely dry and wet conditions. Based on the threshold values, the wetness percentage is calculated by comparing the raw sensor value to the calibrated dry and wet values.

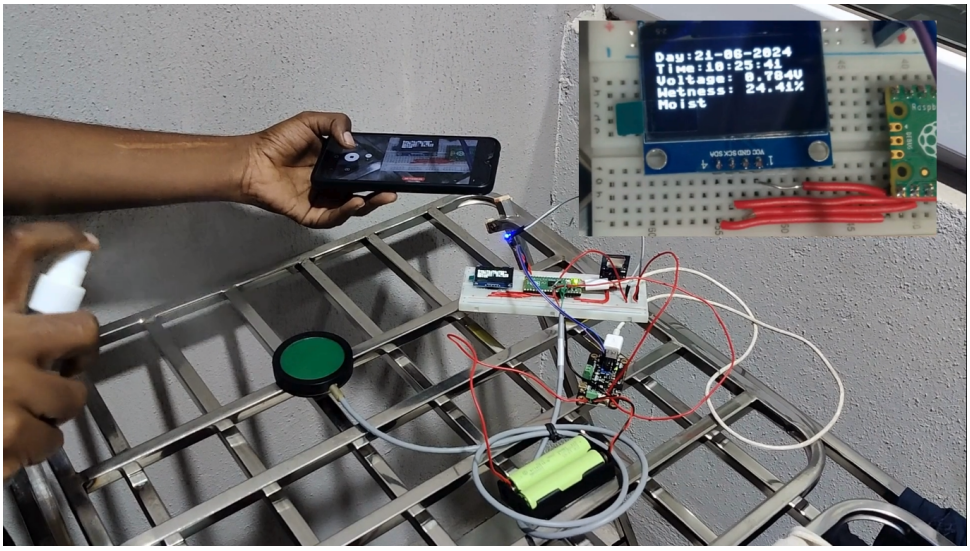

Figure 2.8: Measuring wetness from simulated environment

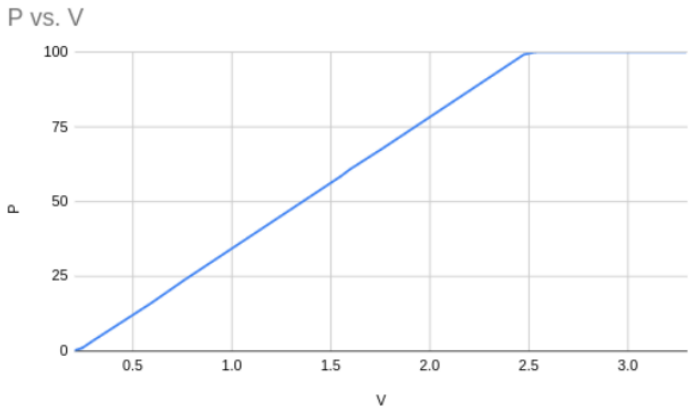

Figure 2.9: Voltage vs Percentage

Then the wetness percentage is categorized into levels such as ”dry”, ”slight moist”, ”moist”, ”wet” and ”saturated”, helping in interpreting the sensor data practically.

2.5 PCB Designing

For the standalone operation, we need a PCB to hold and connect all the components. The PCB design for the whole system (including all 4 sensors) is designed in KiCad. It is centered around the Raspberry Pi Pico microcontroller.

2.5.1 Schematic

It incorporates various connections to peripheral components through sockets. Components include:

- Raspberry Pi Pico with two 20-pin connectors
- SD Card Reader with 6 pin connectors

```
dry_value = 0.23* 65535/3.3
wet_value =  2.5 *65535/3.3

def calibrate_sensor(dry, wet, raw_value):
    wetness_percentage = 100 * (raw_value - dry) / (wet - dry)
    wetness_percentage = max(0, wetness_percentage)
    wetness_percentage = min(wetness_percentage,100)
    return wetness_percentage

def classify_voltage(wetness):
    if wetness < 10:
        return "Dry"
    elif 10 <= wetness < 20:
        return "Slight Moist"
    elif 20 <= wetness < 40:
        return "Moist"
    elif 40 <= wetness < 80:
         return  "Wet"
    else:
        return "Saturated"
```

Figure 2.10: Code for Calibration and Classification of Leaf Wetness Sensor Data

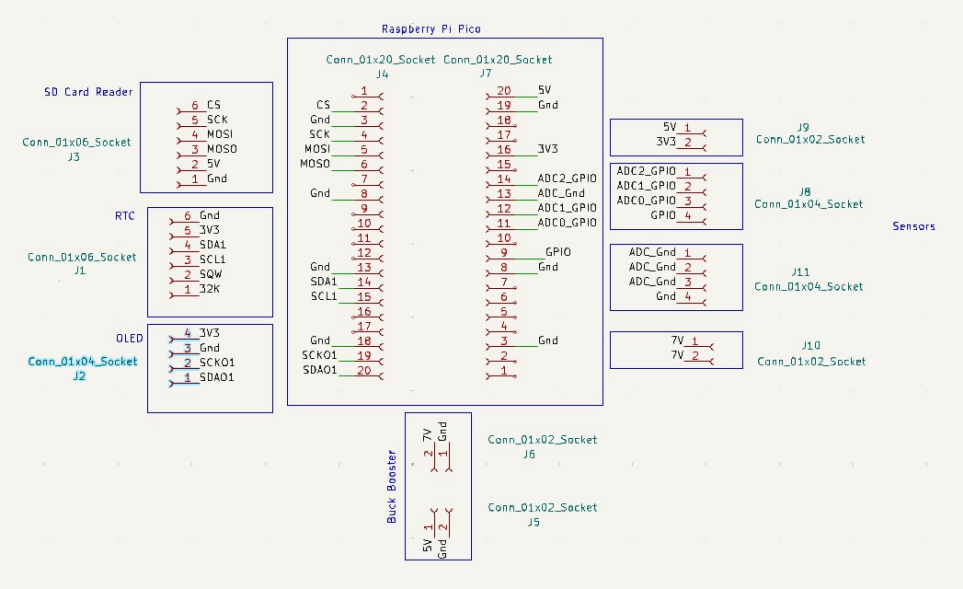

Figure 2.11: Schematic of PCB Design

  - CS (Chip Select)
  - SCK (Serial Clock)
  - MOSI (Master Out Slave In)
  - MOSO (Master In Slave Out)
  - 5V
  - GND (Ground)

- RTC Module with 4-pin connectors
  - 3V3 (3.3V power supply)
  - GND (Ground)

  - SDA1 (I2C data line)
  - SCL1 (I2C clock line)

- OLED Display with 4-pin connectors
  - 3V3 (3.3V power supply)
  - GND (Ground)
  - SCK01 (Serial Clock for I2C)
  - SDA01 (Serial Data for I2C)

- Buck Booster with 4-pin connectors for 5 V, 7 V and 2 GND grounds
- Sensor Connections are ADC2 for evaporation sensor, ADC1 for leaf wetness sensor, ADC GND for sensor grounds and GPIO for soil mositure sensor.
- Power and Ground Pins for 5V, 3V3 (3.3V power supply), 7V and GND (Ground)

This schematic outlines the connections and pin assignments for integrating an SD card reader, RTC module, OLED display, buck booster and sensors with the Raspberry Pi Pico for a modular front-end framework.

2.5.2 PCB Layout

After drawing the schematic, the components are positioned correctly in the layout. The layout includes footprints and socket connectors for the components.

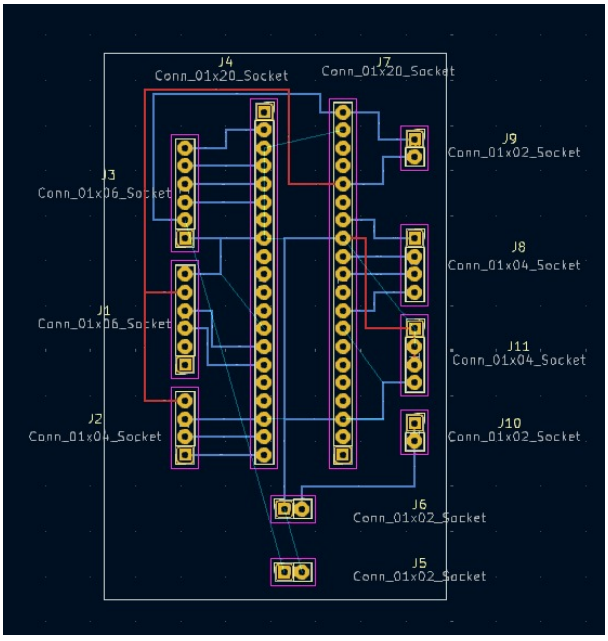

Figure 2.12: PCB Layout of the setup

The blue and red lines represent electrical traces connecting these components. This step is essential to ensure all components are correctly positioned and connected, allowing for soldering and the creation of a the standalone module. After assigning footprint and observing the 3D design, the components are kept on the holes and soldered.

3. Stand-alone Operation

The project is designed for stand-alone operation. The integrated BMS facilitates the self-sustaining capability. This configuration enables continuous and autonomous data collection, processing and storage, making the system ideal for remote agricultural environments.

3.1 Setup

Each component is properly positioned and soldered onto the board, ensuring that all connections are made correctly. The wires from the sensors are given from the outside to the box.

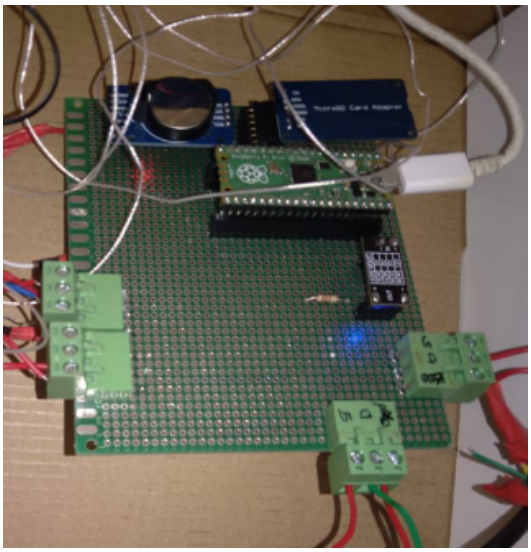

Figure 3.1: Soldered Setup with all components in it

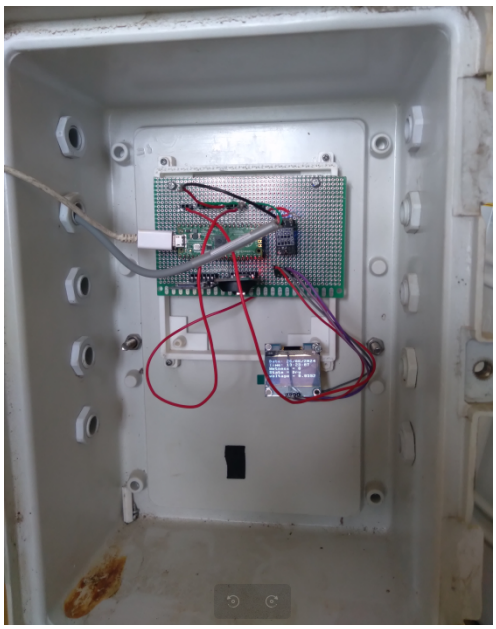

Figure 3.2: Soldered Setup inside Waterproof Box without BMS

The soldered PCB is then mounted onto a waterproof box to deploy into the field. The BMS is then attached to the soldered PCB board to power up the system to make it standalone.

3.2 Main Code

The MicroPython code for the complete standalone modular system is given below

```
from machine import Pin, I2C, ADC, SPI
from sh1106 import SH1106_I2C
import sdcard, uos, time
from ds3231_i2c import DS3231_I2C

led = Pin(25, Pin.OUT)
led.value(1)
# Initializations
wetness_in = ADC(1)
temp = ADC(0)
eva = ADC(2)

i2c = I2C(0, sda=Pin(16), scl=Pin(17), freq=400000)
w = 128
h = 64

oled = SH1106_I2C(w, h, i2c)

ds_i2c = I2C(1, sda=Pin(14), scl=Pin(15))
ds = DS3231_I2C(ds_i2c)
cs = Pin(1, Pin.OUT)
spi = SPI(0, baudrate=1000000, polarity=0,
          phase=0, bits=8, firstbit=SPI.MSB,
          sck=Pin(2), mosi=Pin(3), miso=Pin(4))
sd = sdcard.SDCard(spi, cs)
vfs = uos.VfsFat(sd)
uos.mount(vfs, "/sd")
file = open("/sd/sensor_data.csv", "a")

freq_pin = Pin(19, Pin.IN, machine.Pin.PULL_UP)
counter = 0
def pulse_counter(pin):
    global counter
    counter += 1
freq_pin.irq(trigger=Pin.IRQ_RISING, handler=pulse_counter)

dry_value = 0.23 * 65535 / 3.3
wet_value = 2.5 * 65535 / 3.3

def read_voltage():
    total = 0
    for _ in range(100):
        total += wetness_in.read_u16()

    average = total / 100
    voltage = average * (3.3 / 65535)
    return voltage, average

def calibrate_sensor(dry, wet, raw_value):
    wetness_percentage = 100 * (raw_value - dry) / (wet - dry)
    wetness_percentage = max(0, wetness_percentage)
```

```
    wetness_percentage = min(wetness_percentage, 100)
    return wetness_percentage

def classify_voltage(wetness):
    if wetness < 10:
        return "Dry"
    elif 10 <= wetness < 20:
        return "Slight Moist"
    elif 20 <= wetness < 40:
        return "Moist"
    elif 40 <= wetness < 80:
        return "Wet"
    else:
        return "Saturated"

def read_temperature_sensor():
    total_value = 0.0
    total_value_gnd = 0.0
    for _ in range(100):
        raw_value = temp.read_u16()
        total_value += raw_value
    avg_raw_value = float(total_value) / 100.0
    conv_rate = 3.3 / 65535.0
    voltage = (avg_raw_value) * conv_rate
    return voltage

NUM_SAMPLES = 10

def read_moisture_sensor():
    total_value = 0

    for _ in range(NUM_SAMPLES):
        raw_value = eva.read_u16()
        total_value += raw_value

    avg_raw_value = total_value // NUM_SAMPLES

    voltage = avg_raw_value * (3.3 / 65535) - 0.020
    return voltage

start_time = time.ticks_us()
while True:
    x = time.ticks_us()
    if time.ticks_diff(x, start_time) >= 1000000:
        voltage, raw_value = read_voltage()
        wetness = calibrate_sensor(dry_value, wet_value,
           raw_value)
        state = classify_voltage(wetness)

        voltage_temp = read_temperature_sensor()
        temp_val = 345.6276 * voltage_temp - 537.1619
```

```
102
103         voltage_eva = read_moisture_sensor()
104
105         t = ds.read_time()
106         timestring_day = "%02x/%02x/20%x" % (t[4], t[5], t[6])
107         timestring_time = "%02x:%02x:%02x" % (t[2], t[1], t[0])
108         with open("/sd/sd1.txt", "a") as file:
109             file.write(f"{timestring_day} {timestring_time},
                    Voltage = {voltage}, Wetness = {wetness}, state =
                    {state}, Temperature = {temp_val}, Evaporation = {
                    voltage_eva}, Frequency = {str(counter * 8)}\n")
110             file.flush()
111
112         oled.fill(0)
113         oled.text(f"Date: {timestring_day}", 0, 0)
114         oled.text(f"Time: {timestring_time}", 0, 10)
115         oled.text(f"Wetness: {wetness}", 0, 20)
116         oled.text(f"Frequency = {str(counter * 8)}", 0, 30)
117         oled.text(f"Temp = {temp_val}C", 0, 40)
118         oled.text(f"Evap = {voltage_eva}", 0, 50)
119         oled.show()
120         start_time = time.ticks_us()
121         counter = 0
```

3.3 Testing

For testing the standalone module and carrying out the power analysis, we decided to first test in lab and then put to the field for actual field testing.

3.3.1 Testing in lab

In the lab, we were able to control various conditions like changing battery capacities and systematically evaluate the performance of each component and the overall system. For in lab testing, we kept the waterproof box connected with sensors and all other components powered with batteries of different capacities. The system was tested to completely utilize the batteries and drain them fully to determine the runtime and compare it with our theoretical calculations. This involved continuously running the module until the batteries were completely depleted.
We monitored the sensor data displayed on the OLED screen and ensured it was accurately saved with a timestamp on the SD card. This controlled environment allows us to troubleshoot and make any necessary adjustments before exposing the module to real-world conditions. It also helped us optimize the power usage and enhance the overall reliability of the standalone module for field applications.

3.3.2 Testing in field

Once lab testing confirmed the module's functionality and reliability, we moved to field testing. In the field, we deployed the system in the agricultural field to observe its performance under varying weather, environmental factors and time.

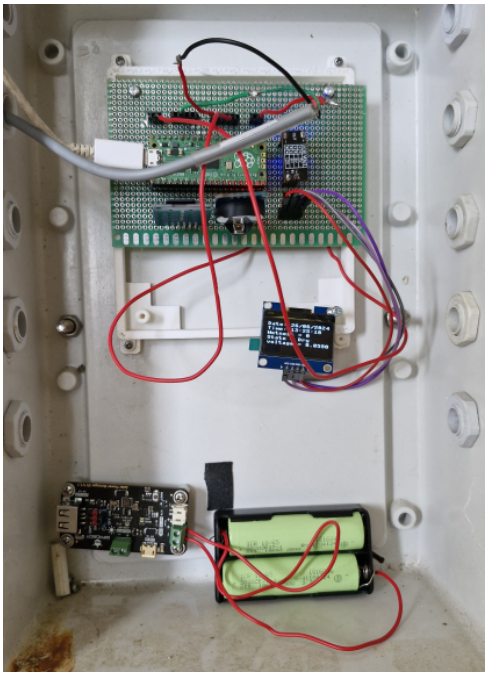

Figure 3.3: Soldered Setup inside Waterproof Box with BMS

We also incorporated a solar panel along with the batteries to provide additional power support during the daytime. By integrating a solar power manager/controller, we were able to charge the Li-ion batteries during the day using solar energy. This setup helps to ensure continuous operation and extends the battery life of the standalone module.

This testing was crucial for assessing the system's durability, power efficiency with the solar panel and Li-ion batteries and its ability to provide accurate and consistent data over time.

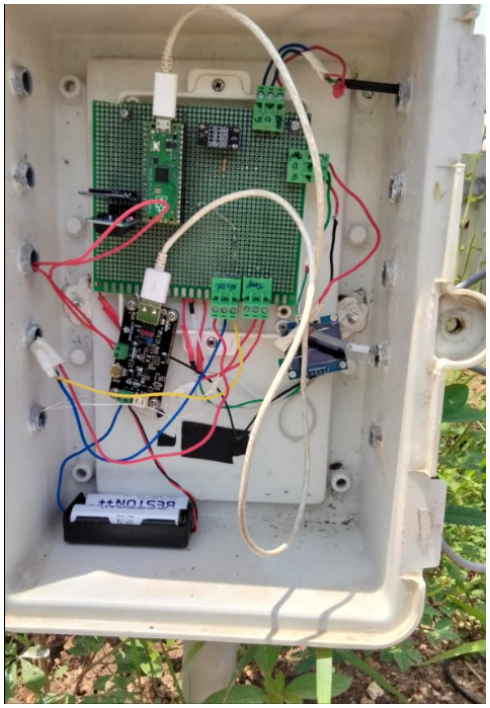

Figure 3.4: Setup with all four sensors and BMS in the field

Figure 3.5: Setup with solar panel in the field

Figure 3.6: Leaf Wetness Sensor with the setup

4. Power Analysis

4.1 Theoretical Analysis

First we calculated the current consumed by the system without adding sensors, i.e, with only Raspberry Pi Pico, hardware components and BMS. We found out that the Raspberry Pi Pico drawn about 100 mA of current. Individual current consumption of each sensor is found out to be 10 mA.

Current consumed by Microcontroller and Hardware components = $100mA$

Current consumed by all sensor = $4 \times 10mA = 40mA$

Current consumed by all system = $140mA$

Power consumed by all system = $5V \times 140mA = 0.7W$ The solar controller supplies power using a in-built linear regulator of 3.7 V to 5V. And the power transfer efficiency of the solar power manager is 86%

So, it implies the total current drawn from the batteries = $\frac{0.7W}{3.7 \times 0.86} = 0.21998A \approx 219.98mA$

If it draws 219.98 mA, and we use a 5000 mAh battery, we will get a runtime or battery life of $\frac{5000}{219.98} = 22.729hours \approx 22hours$

For a 3000 mAh battery, the runtime wold be = $\frac{3000}{219.98} = 13.638hours \approx 13hours$

4.2 Practical Analysis

4.2.1 In lab

We kept the setup inside the lab and allowed it to fully drain the 3000 mAh battery. It ran for precisely 13 hours, validating our theoretical calculations.

4.2.2 Solar Panel

The solar panel we have used is 6 W rated, which can supply 6 - 7 V with 1 A. It provided 6.3V, 0.268 A on gloomy day and 7V, 0.965 A on sunny day.

And the solar power manager has a solar power transfer efficiency of 73%. So to charge

battery of 3000 mAh capacity,

$$\text{On gloomy day, time time taken will be} = \frac{3000 \times 3.7}{6.3 \times 0.268 \times 0.73} = 9.01 hours$$
$$\text{On sunny day, time time taken will be} = \frac{3000 \times 3.7}{7 \times 0.965 \times 0.73} = 2.25 hours$$

4.2.3 In field

For the field setup, we added a solar panel to increase indepedency and battery life. We tested the setup with a 5000 mAh. The setup worked for 27 hours, which implies that it used power from solar during daytime and utilised battery power during evening, nights and early mornings.

5. Results

5.1 Data Analysis

The data collected throughout the project is given here. Analyzing this data helped in identifying trends and patterns in the data.
The plots for voltage vs time and wetness vs time from 05/07/202, 11:00 A.M to 06/07/2024, 7:00 A.M is given below

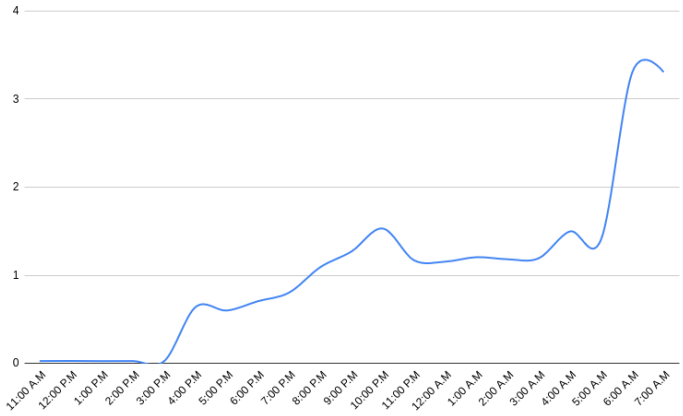

Figure 5.1: Voltage vs Time from 05/07/202, 11:00 A.M to 06/07/2024, 7:00 A.M

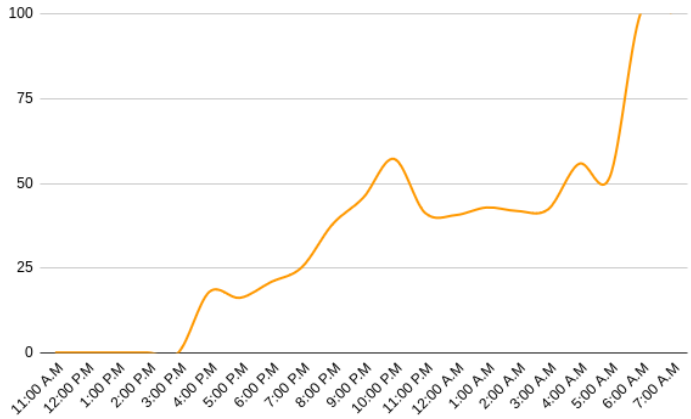

Figure 5.2: Wetness vs Time from 05/07/202, 11:00 A.M to 06/07/2024, 7:00 A.M

5.2 Findings

The data analysis revealed a correlation between sunlight, moisture and readings from the leaf wetness sensor:

- We can see that in afternoon, during peak sunlight, the leaf wetness sensor is showing 0% wetness and a very low voltage.
- As the day move towards evening, we can see that the voltage and wetness is gradually increasing. This is due to increase in the moisture and humidity in the air during evening and night.
- A significant rise in wetness coincides with the reported rain in Palakkad on the morning of 06/07/2024 (data here), demonstrating the sensor's ability to detect moisture changes.

6. Conclusion

6.1 Summary

- The development of a modular front-end framework for agricultural sensors has been successfully completed
- This project demonstrates the effective integration of agricultural sensors with a Raspberry Pi Pico microcontroller, supported by a robust battery management system powered by solar energy.
- The system's ability to autonomously collect, process, display and store data with precise timestamps ensures reliable and continuous monitoring of environmental conditions.
- The successful testing and validation of the module in both lab and field conditions confirm its effectiveness in providing accurate and timely environmental data
- Overall, the project achieves its objective of creating a reliable, standalone module for improved crop management.

6.2 Future Work

- Making it into a proper IoT device with remote monitoring
- Implementing timer algorithms for extended battery life

A. Annexure

A.1 Codes

The link for the codes are here

A.2 Datasheet

The link for the datasheets are here

A.3 Gallery

The link to the gallery is here

List of Figures

Bibliography

[1] Nguyen, B.H., Gilbert, G.S. and Rolandi, M. (2023), A Bio-Mimetic Leaf Wetness Sensor from Replica Molding of Leaves. Adv. Sensor Res., 2: 2200033. https://doi.org/10.1002/adsr.202200033

[2] Potratz, Kara J. and Gleason, Mark L. and Hockmuth, Melanie L. and Parker, Sharon K. and Pearston, Glenn A.(1994), Testing the Accuracy and Precision of Wetness Sensors in a Tomato Field and on Turfgrass, Journal of the Iowa Academy of Science., https://scholarworks.uni.edu/cgi/viewcontent.cgi?article=1398&context=jias

[3] Leaf Wetness Sensor, Decagon Devices, Inc. https://www.youtube.com/watch?v=caz1Mspxn-M