The Sous-gardener II ™

Nana Porter-Honicky

ME135 Design of Microprocessor-Based Mechanical Systems

Contents

1 Overview 1
1.1 Background & Motivation . . . 1
1.2 Description 1

2 GUI 2
2.1 Real-Time & Multitasking . . . 2

3 Highlights 2

4 Reflection 3

1 Overview

1.1 Background & Motivation

I am a big plant mom. I have over 80 plants in my apartment and sometimes that can be a bit overwhelming to handle as a full time engineering student. This is where the Sous-gardener II ™ comes in. It is called the Sous-Gardener II because this project is an upgrade to a project done for ME100 with the same name. The original Sous-Gardener ™ only read soil moisture data and then sent it to an MQTT plotter. The Sous-Gardener II ™ will not only plot moisture *and* light level data it will also actuate a pump and grow-light based on the sensor data as well as allow for toggling of both the light and the pump.

1.2 Description

The set up for this project is fairly simple. It uses two sensors, a moisture sensor and a light sensor, to actuate two things, a small pump and a grow light. The code governing the actuation has two general states: manual and automatic. There is a button in the GUI allowing you to switch between the two settings. In the automatic setting, two loops run asynchronously that read from the two sensors and then actuate the pump and light when predetermined thresholds are met. A third asynchronous loop checks for button inputs from the GUI. The three buttons in the GUI toggle the setting, toggle the light when in the manual setting, and toggle the pump when in the manual setting. There is a fourth asynchronous loop that reads the sensors every second so that regardless of which setting the system is in, the GUI can always plot the sensor readings. The time between sensor readings can be decreased but for the purposes of this project it wasn't necessary.

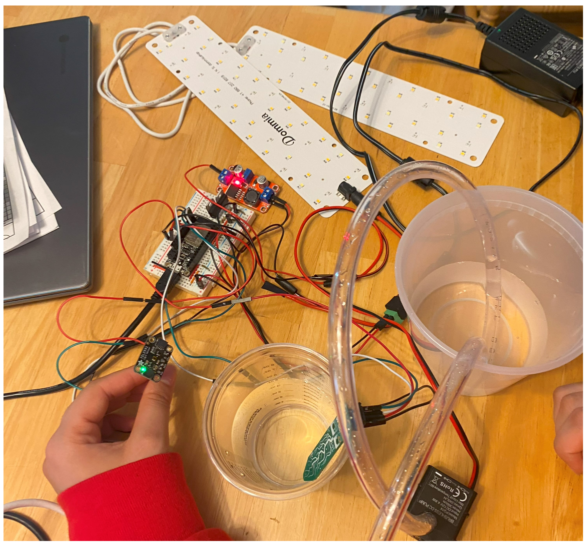


Figure 0: Full project assembly

In the automatic loop, a simple hysteresis control is used on the grow light based on the light sensor data to prevent flickering (no one wants a headache from this thing). The pump control is a bit more complex (only a bit); it is set to turn on the pump once the moisture threshold is met only if it has not been turned on in the last 20 seconds. The time of 20 seconds was chosen for the demo but for actual use, the time between watering would be closer to about two minutes.

2 GUI

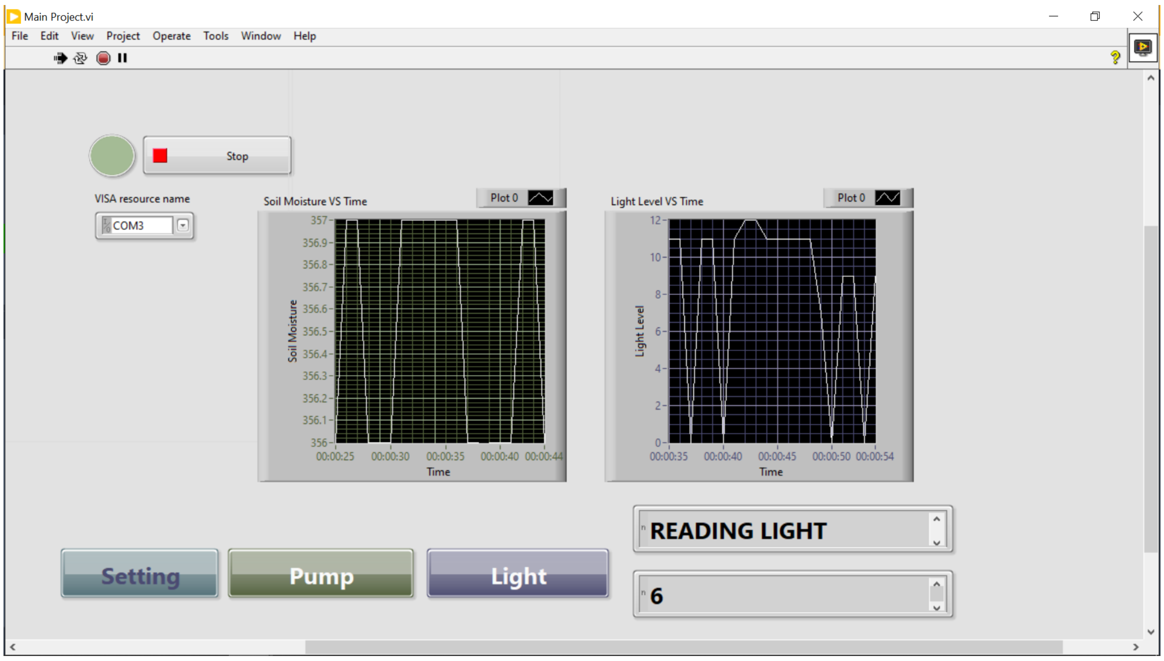


Figure 1: GUI screen capture

2.1 Real-Time & Multitasking

The primary real-time multitasking implementation comes from the use of the module **uasyncio**. This module allows us to (basically) simultaneously read the sensors, check for input events, and toggle the pump and light without any of these tasks interfering with each other. It achieves this by quickly switching between each of these tasks to allow each one an adequate portion of the CPU. What this means is the program is responsive event is **real-time** and can handle multiple events at once (**multitasking**).

3 Highlights

There were two main issues I encountered when trying to get the code to work smoothly; the first was getting the events to trigger *every* time there was a button press in the GUI. After many failed attempts at get ChatGPT's help and a very helpful suggestion from my father I decided to put all functions reading for input events into one loop. Previously, I had each function in a separate asynchronous loop, which was causing the code to miss inputs maybe 60% of the time. Although this is not the most impressive part this project, it was a big win for me when I figured it out.

The second big issue I encountered was getting the GUI to receive and plot the sensor data without commissioning it on the Labview end. I initially had Labview prompt the sensor data every time the front panel event structure went in to a timeout (every 25ms) but this resulted in the send-que being to full for button events to happen in a timely manor. My solution was to change the communication state machine to check for responses from the ESP32 that weren't prompted. The resulting state machine looked like this:

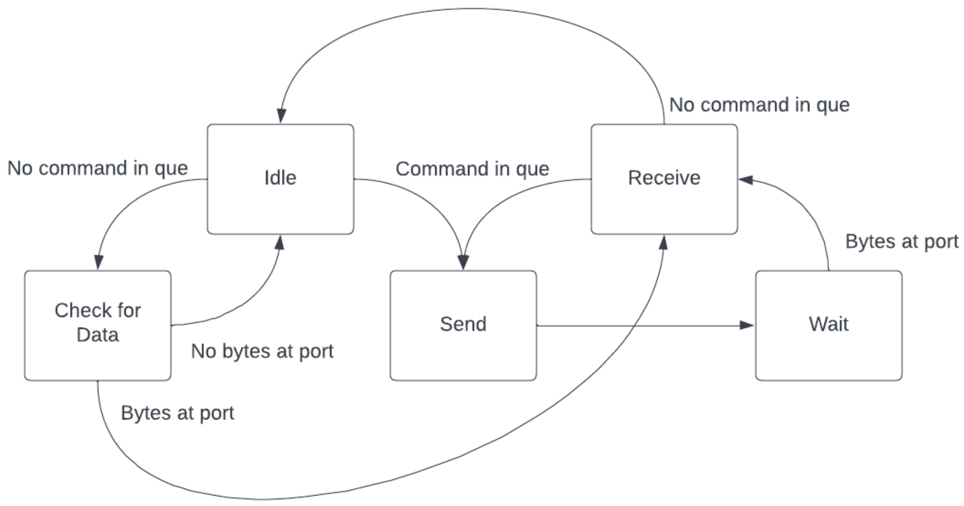


Figure 2: Changed communication state machine

This state machine switches between checking for responses from the ESP32 and commands from Labview so that the GUI can plot the sensor data without prompting it while still maintaining function of the buttons in the GUI. This state machine also stays in receive until there is a command from Labview instead of until there are no more bytes at the port. This ensures that buttons in the GUI will be read even when there are still responses coming in from the ESP32.

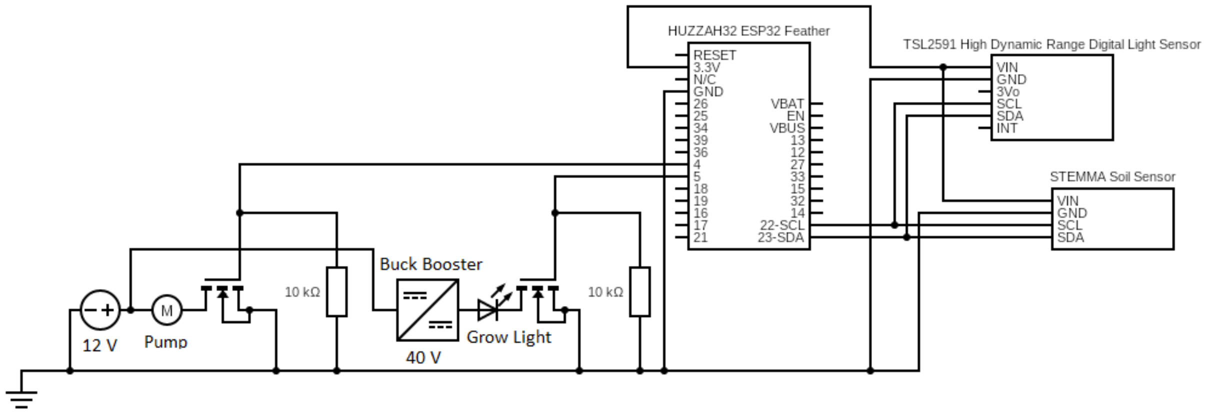
4 Reflection

The obvious next step for this project would be to build it out and see if it can actually keep a plant alive. Beyond that thought there are a couple minor improvements I would make. The first is that I think the communication state machine could be made a bit more efficient by having the state stay as receive until there is a command in the que instead of sending it back to idle. I also think using multiple sensors for both moisture and light and taking some aggregate of all of the data would make the system a lot more robust. That way if there is a momentary shadow on a light sensor or the moisture sensor is in a particularly dry or wet spot of soil, the light won't needlessly turn on and off and the plant is more likely to get watered properly. Another improvement would be to test out different plants preferred thresholds and have some way to change the thresholds on the GUI side of things depending on the plant. Another thing that may be cool is to make it wireless (use some of that ME100 knowledge) so that you can check in on your plants from afar.

Overall I'm pretty happy with how this project turned out given that I have pretty limited experience with MicroPython and I had never used Labview before this class. I actually plan on testing the Sous-gardener II on a few of my plants while I'm traveling this summer.

Appendix

Circuit Diagram



Parts List

	Amount	Name
1	x1	HUZZAH32 – ESP32 Feather Board
2	x1	12V Mini Submersible Water Pump
3	x1	Full Spectrum Grow Light
4	X1	STEMMA Soil Sensor
5	x1	TSL2591 High Dynamic Range Digital Light Sensor
6	x1	12V AC/DC Converter
7	x2	n-Channel Mosfet

MicroPython Code

```
import machine
from machine import Pin
import time
import seesaw
import stemma_soil_sensor as Soil
import tsl2591
import uasyncio as asyncio
import select
import sys
```

```
growLight = Pin(5, Pin.OUT)
pump = Pin(4, Pin.OUT)
SDA_PIN = 23
SCL_PIN = 22
light = False
water = False
lastWatered = None
manual = False

i2c = machine.SoftI2C(sda=Pin(SDA_PIN), scl=Pin(SCL_PIN), freq=400000)
moist = Soil.StemmaSoilSensor(i2c)
lightSensor = tsl2591.Tsl2591(i2c)
print('\r\nESP32 Ready to accept Commands\r\n')

def toggleLight():
    global light
    light = not light
    if light == True:
        print("Gturning light on\n")
    else:
        print("Gturning light off\n")
    growLight.value(light)

def togglePump():
    global water
    water = not water
    if water == True:
        print("Pturning water on...\n")
    else:
        print("Pturning water off...\n")
    pump.value(water)

def toggleSetting():
    global manual
    manual = not manual
    if manual == True:
        print("Tswitching from auto to manual\n")
    else:
        print("Tswitching from manual to auto\n")

async def readSensors():
    while True:
        mo = moist.get_moisture()
        full = lightSensor.get_luminosity("full")
        print("W"+str(mo))
        await asyncio.sleep(1)
        print("L"+str(full))

async def checkMoisture():
    global lastWatered
    global manual
    while True:
        if manual == False:
            mo = moist.get_moisture()
            if mo < 400:
                if not pump.value() and (lastWatered is None or ...
                time.time() - lastWatered > 20):
                    print("Pwatering plant...\n")
                    pump.value(True)
                    await asyncio.sleep(5)
                    pump.value(False)
                    lastWatered = time.time()
                    print("Pturning off water...\n")
#                 elif lastWatered is not None and time.time() ...
#                 - lastWatered <= 20:
#                     print("too soon to water again...")
            elif mo > 450:
                pump.value(False)
        await asyncio.sleep(0.5)

async def checkLight():
    global light
    global manual
    while True:
        if manual == False:
            full = lightSensor.get_luminosity("full")
            if full < 25:
                if not light:
                    print("Gturning on grow light...\n")
                light = True
            elif full > 35:
                if light:
                    print("Gturning off grow light...\n")
                light = False
            growLight.value(light)
        await asyncio.sleep(0.5)

async def manualSetting():
    global manual
    while True:
        rlist, _, _ = select.select([sys.stdin], [], [], 0)
        if rlist:
            command = sys.stdin.readline().strip()
            if command == "I":
                print('IESP32\r\n')
            elif command == "T":
                toggleSetting()
            elif command == "P" and manual:
                togglePump()
            elif command == "G" and manual:
                toggleLight()
        await asyncio.sleep(0.5)

loop = asyncio.get_event_loop()
loop.create_task(checkMoisture())
loop.create_task(checkLight())
loop.create_task(readSensors())
loop.create_task(manualSetting())
loop.run_forever()
```