CSCI 43300
Final Project

Caleb Kirby, Jason Hampshire

Introduction & Background

Picture this:

- Got a plant from grandma
- She’s visiting next month
- Place it in bright place, water it, and forget about it
- Then before she visits you remember
- When you check on it, you are presented with this:

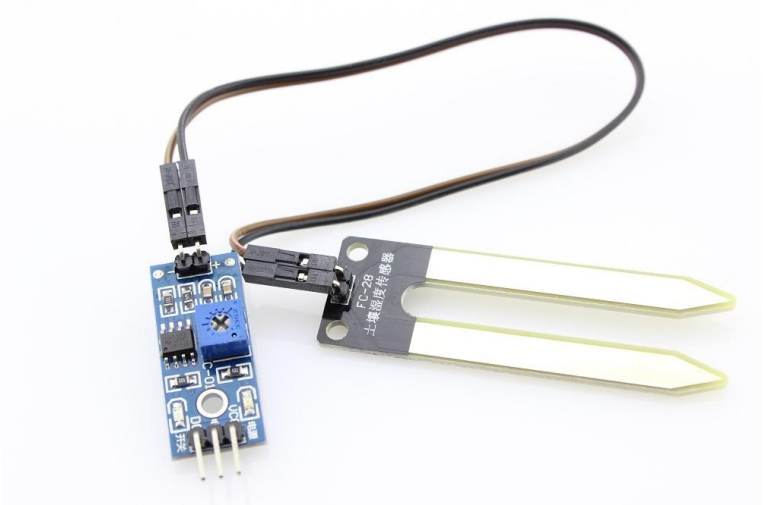
Problem & Solution

- Dead plant = Sad Grandma
- Healthy plant = Happy Grandma
- Should have found a way to keep the plant alive
- A solution:

  A device that senses vital information so the status of the plant can be monitored

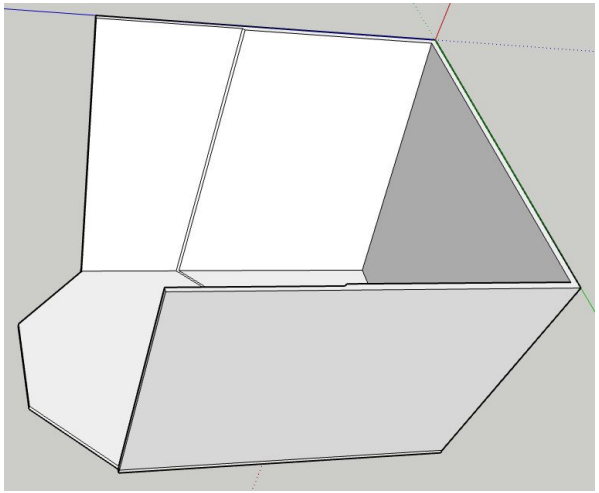
What should be monitored

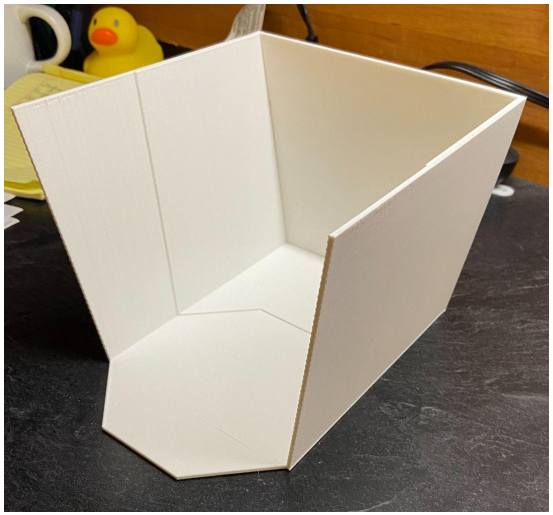
- A plant needs the right environment to stay alive and healthy
- Important statistics to monitor:
  - Soil Moisture
  - Gas/Co2
  - Temperature
  - Light

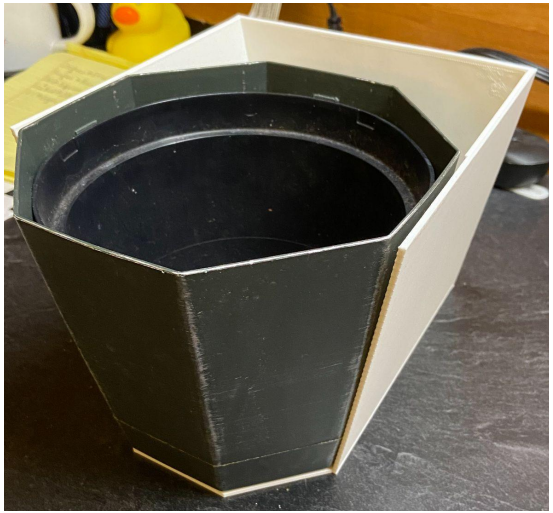


- Each of which will be monitored respectively by
  - HW-080 Soil Hygrometer
  - CCS811 Gas Sensor
  - DHT11 Temp/Humid Sensor
  - Photoresistor

Physical Construction

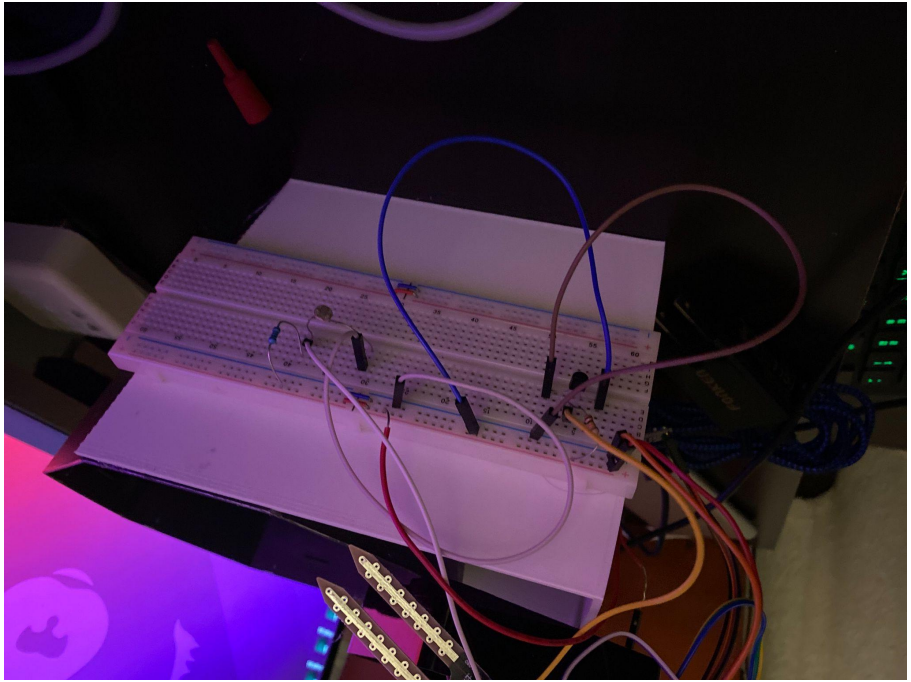


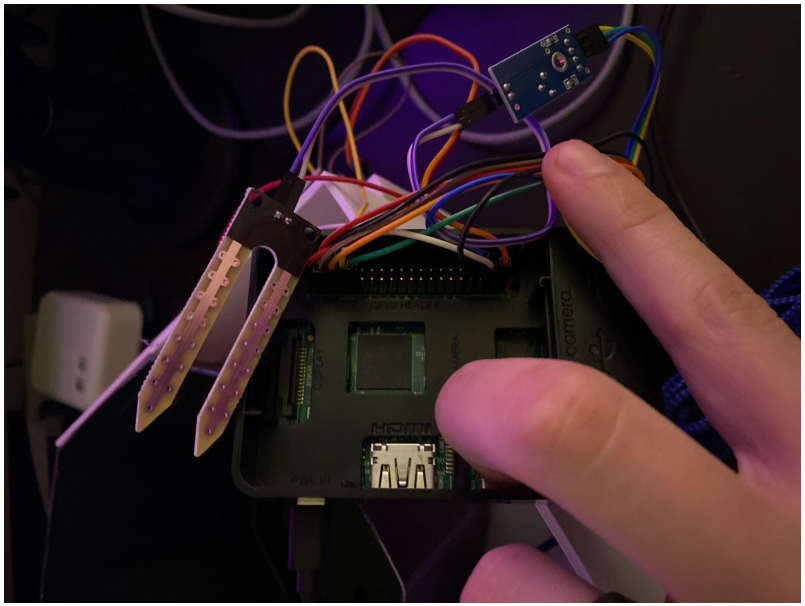


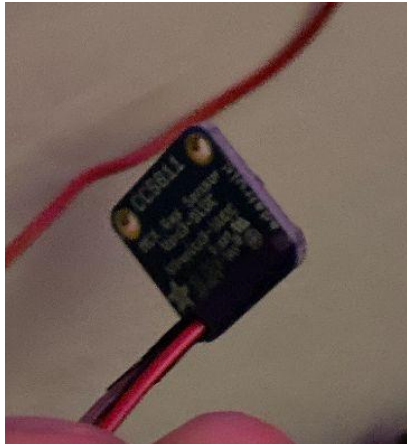


- 3D model created to house electronics
- Print was done by another student
- Pot slots into 3d print
- Pot is secured with waterproof epoxy

Images







Sensor Code

Gas Sensor

```
import time
import board
import adafruit_ccs811

i2c = board.I2C()  # uses board.SCL and board.SDA
ccs811 = adafruit_ccs811.CCS811(i2c)

print("initializing co2 monitor")
# Wait for the sensor to be ready
while not ccs811.data_ready:
    pass
print("done")

def get_co2():
    return "{}".format(ccs811.eco2)
def get_tvoc():
    return "{}".format(ccs811.tvoc)
```

Temperature Sensor

```
import glob
import time

base_dir = '/sys/bus/w1/devices/'
device_folder = glob.glob(base_dir + '28*')[0]
device_file = device_folder + '/w1_slave'

def read_temp_raw():
    f = open(device_file, 'r')
    lines = f.readlines()
    f.close()
    return lines

def read_temp():
    lines = read_temp_raw()
    while lines[0].strip()[-3:] != 'YES':
        time.sleep(0.2)
        lines = read_temp_raw()
    equals_pos = lines[1].find('t=')
    if equals_pos != -1:
        temp_string = lines[1][equals_pos+2:]
        temp_c = float(temp_string) / 1000.0
        temp_f = temp_c * 9.0 / 5.0 + 32.0
        return temp_c, temp_f
```

Sensor Code

Light (Photoresistor)

```
import RPi.GPIO as GPIO
from time import sleep

GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)

Button = 26

GPIO.setup(Button, GPIO.IN, pull_up_down=GPIO.PUD_UP)

def get_light():
    button_state = GPIO.input(Button)
    if button_state == 0:
        return "Light".encode("UTF-8")
    else:
        return "Dark".encode("UTF-8")
```

Moisture Sensor

```
import RPi.GPIO as GPIO
import time

#GPIO SETUP
channel = 16
GPIO.setmode(GPIO.BCM)
GPIO.setup(channel, GPIO.IN)


def get_watered(channel):
    if GPIO.input(channel):
        watered = False
    else:
        watered = True


watered = get_watered(channel)
def coap():
    return "Thirsty!".encode("UTF-8")
    if not watered else "Satiated!".encode("UTF-8")

# let us know when the pin goes HIGH or LOW
GPIO.add_event_detect(channel, GPIO.BOTH, bouncetime=300)
# assign function to GPIO PIN, Run function on change
GPIO.add_event_callback(channel, get_watered)
```

Sensor Code

CoAP

```python
#!/usr/bin/env python3

import datetime
import logging

import asyncio

import aiocoap.resource as resource
import aiocoap

import temp
import co2
import soil
import light

class TemperatureResource(resource.Resource):
    def __init__(self):
        super().__init__()

    def get_temp(self):
        return bytes(str(temp.read_temp()[1]).encode('UTF-8'))

    async def render_get(self, request):
        return aiocoap.Message(payload=self.get_temp())

```

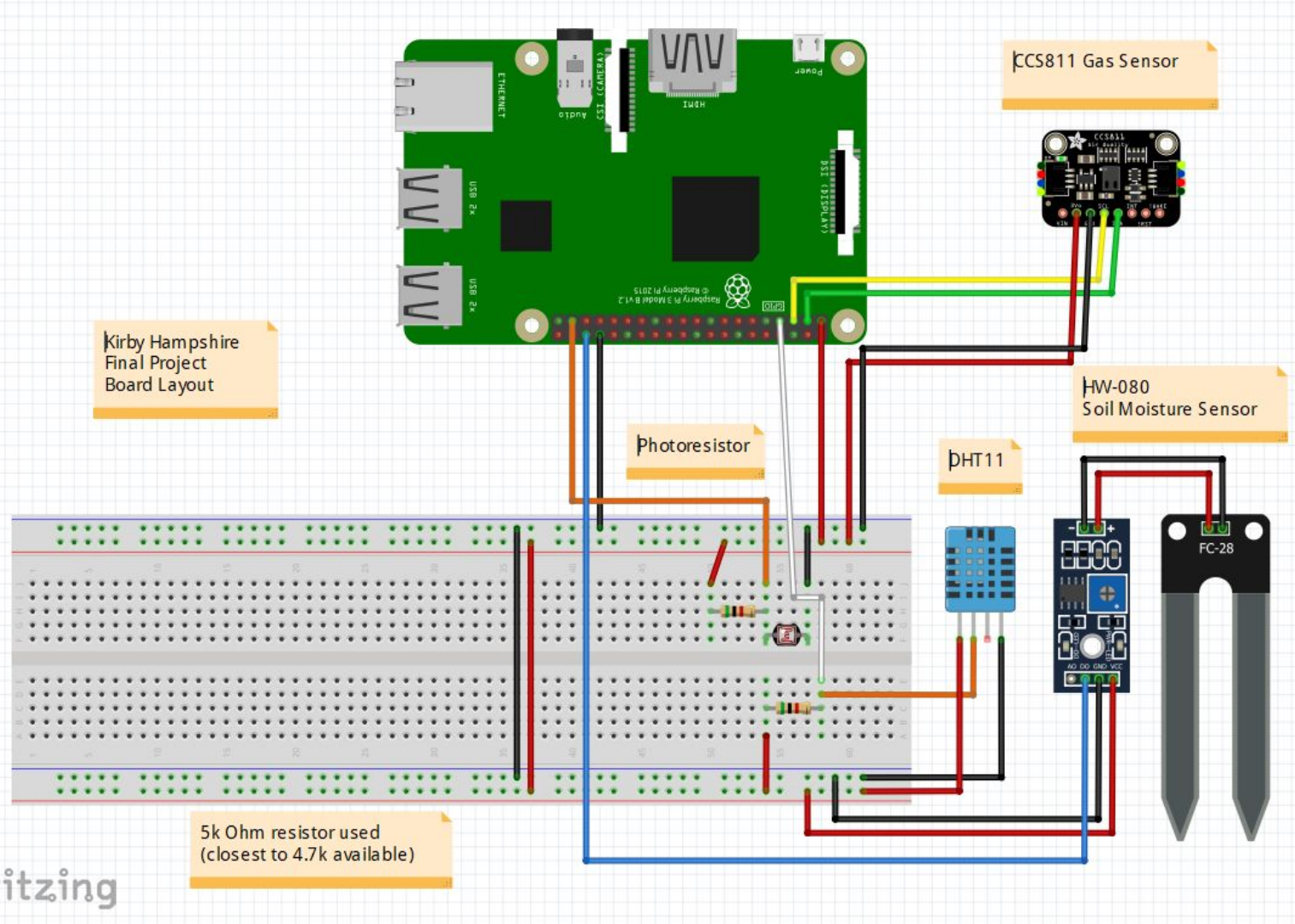
```python
class AQResource(resource.Resource):
    def __init__(self):
        super().__init__()

    async def render_get(self, request):
        return aiocoap.Message(payload=co2.get_co2().encode('UTF-8'))

class TVOCResource(resource.Resource):
    async def render_get(self, request):
        return aiocoap.Message(payload=co2.get_tvoc().encode('UTF-8'))

class SoilResource(resource.Resource):
    async def render_get(self, request):
        return aiocoap.Message(payload=soil.coap())

class LightResource(resource.Resource):
    async def render_get(self, request):
        return aiocoap.Message(payload=light.get_light())

class BlockResource(resource.Resource):
    """Example resource which supports the GET and PUT methods. It sends large
    responses, which trigger blockwise transfer."""

    def __init__(self):
        super().__init__()
        self.set_content(b"This is the resource's default content. It is padded "
                         b"with numbers to be large enough to trigger blockwise "
                         b"transfer.\n")

    def set_content(self, content):
        self.content = content
        while len(self.content) <= 1024:
            self.content = self.content + b"0123456789\n"

    async def render_get(self, request):
        return aiocoap.Message(payload=self.content)

    async def render_put(self, request):
        print('PUT payload: %s' % request.payload)
        self.set_content(request.payload)
        return aiocoap.Message(code=aiocoap.CHANGED, payload=self.content)
```

Sensor Code

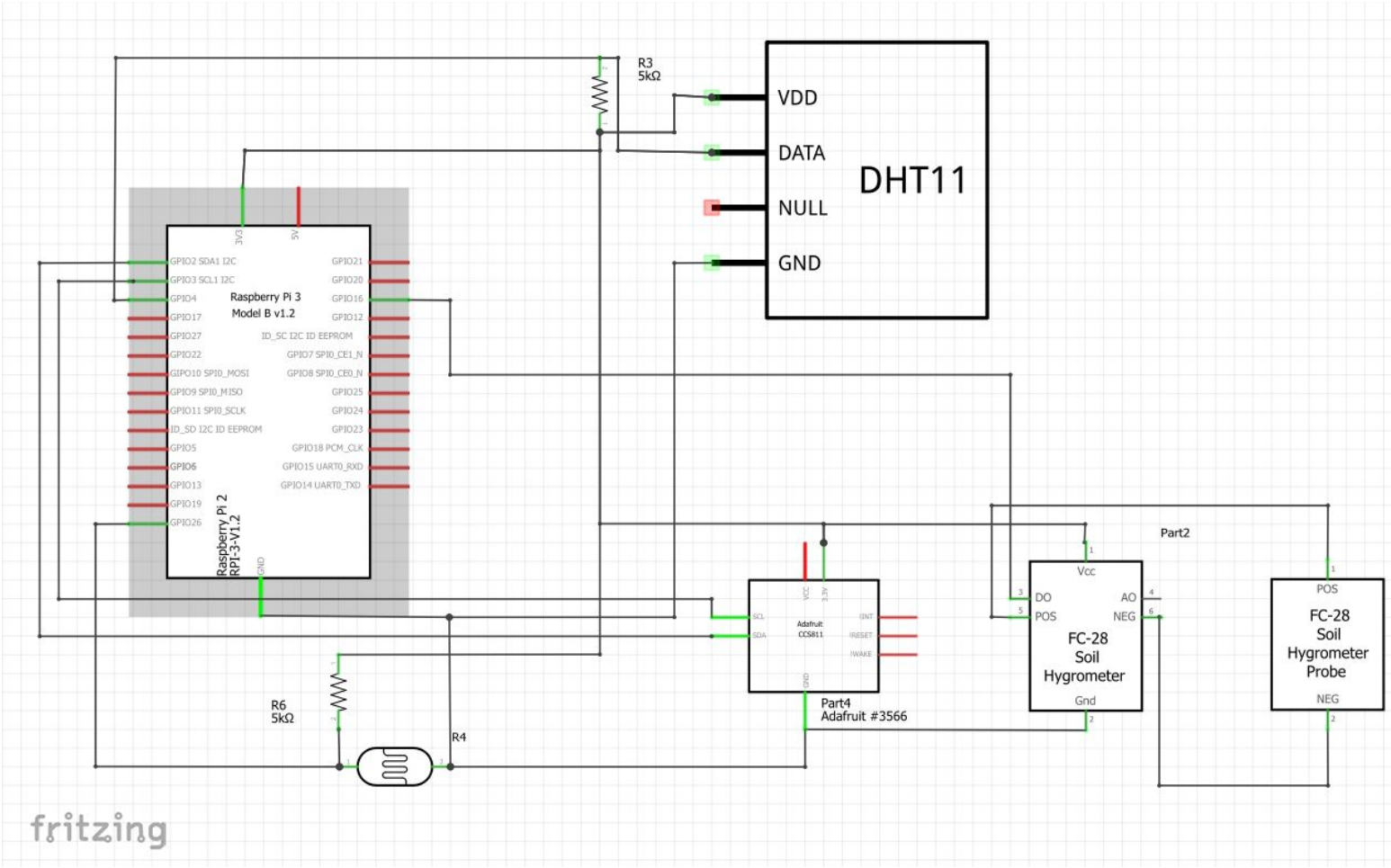
CoAP cont.

```
class WhoAmI(resource.Resource):
    async def render_get(self, request):
        text = ["Used protocol: %s." % request.remote.scheme]

        text.append("Request came from %s." % request.remote.hostinfo)
        text.append("The server address used %s." % request.remote.hostinfo_local)

        claims = list(request.remote.authenticated_claims)
        if claims:
            text.append("Authenticated claims of the client: %s." % ", ".join(repr(c) for c in claims))
        else:
            text.append("No claims authenticated.")

        return aiocoap.Message(content_format=0,
                payload="\n".join(text).encode('utf8'))

# logging setup

logging.basicConfig(level=logging.INFO)
logging.getLogger("coap-server").setLevel(logging.DEBUG)

async def main():
    # Resource tree creation
    root = resource.Site()

    root.add_resource(['.well-known', 'core'],
            resource.WKCResource(root.get_resources_as_linkheader))
    root.add_resource(['time'], TimeResource())
    root.add_resource(['other', 'block'], BlockResource())
    root.add_resource(['other', 'separate'], SeparateLargeResource())
    root.add_resource(['whoami'], WhoAmI())
    root.add_resource(['temp'], TemperatureResource())
    root.add_resource(['co2'], AQResource())
    root.add_resource(['tvoc'], TVOCResource())
    root.add_resource(['soil'], SoilResource())
    root.add_resource(['light'], LightResource())


    await aiocoap.Context.create_server_context(root)

    # Run forever
    await asyncio.get_running_loop().create_future()

if __name__ == "__main__":
    asyncio.run(main())
```

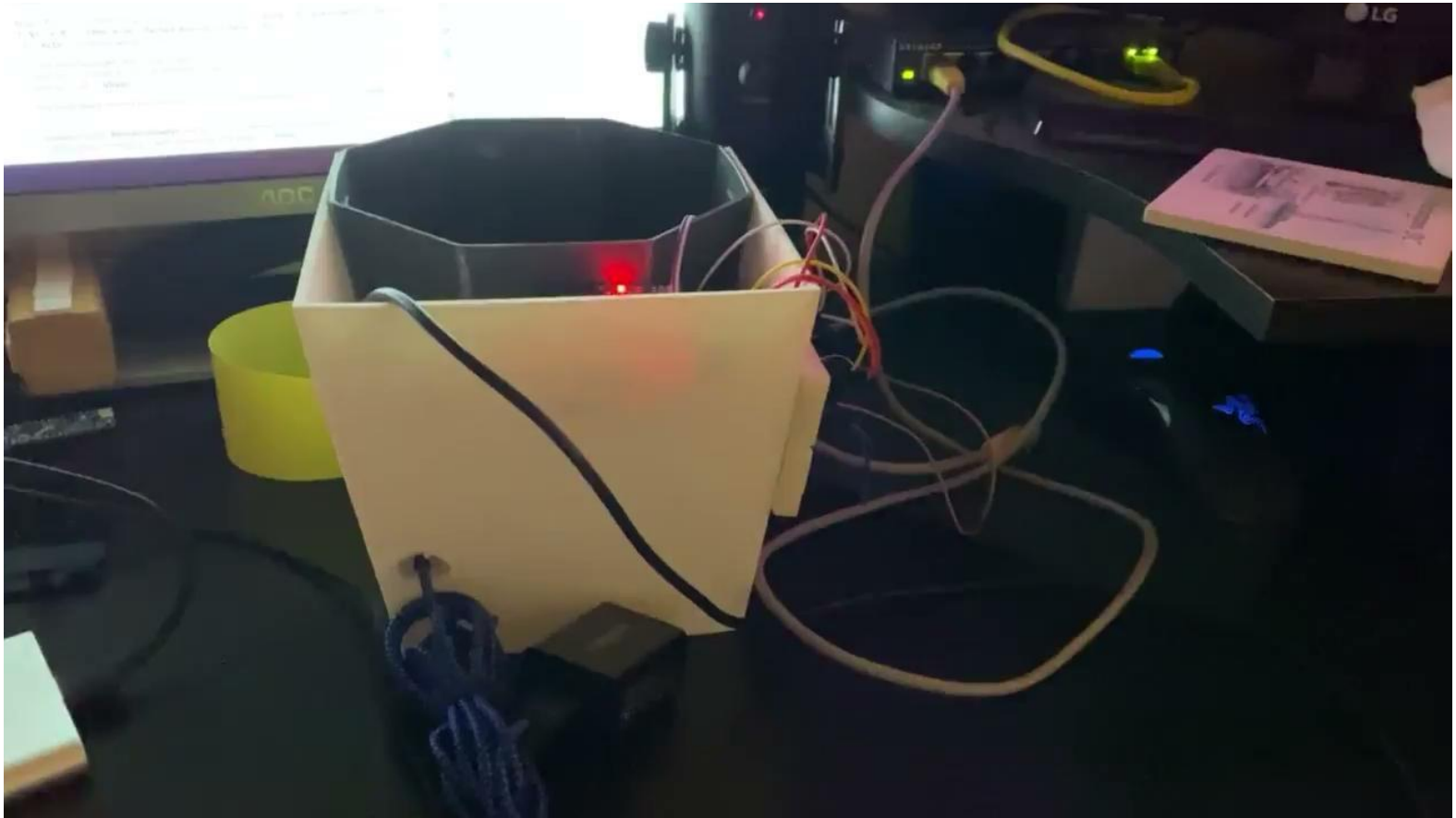
Board Layout



Circuit Diagram



Demo



Tragedy

The test plant was dying… Sago Palms need a lot of sun.

Results and conclusion

- We are able to successfully develop a system that could monitor plant stats by request
- Further steps would be to create a system that would help rectify any stats that were out of line
- A prolonged test over the course of a few months would give results on the effectiveness of the device

References

Photoresistor: Resistor types: Resistor guide. EEPower. (n.d.). Retrieved November 15, 2021, from https://eepower.com/resistor-guide/resistor-types/photo-resistor/#.

Dht11–temperature and humidity sensor. Components101. (n.d.). Retrieved November 15, 2021, from https://components101.com/sensors/dht11-temperature-sensor.

Alldatasheet.com. (n.d.). *HW-080080-10-9*. ALLDATASHEET. Retrieved November 15, 2021, from https://pdf1.alldatasheet.com/datasheet-pdf/view/458562/MACOM/HW-080080-10-9.html.