

Final Project Report

CSCI 43300

Caleb Kirby / Jason Hamshire

kirbycm@iu.edu / jchampsh@iu.edu

0003607441 / 2000079027

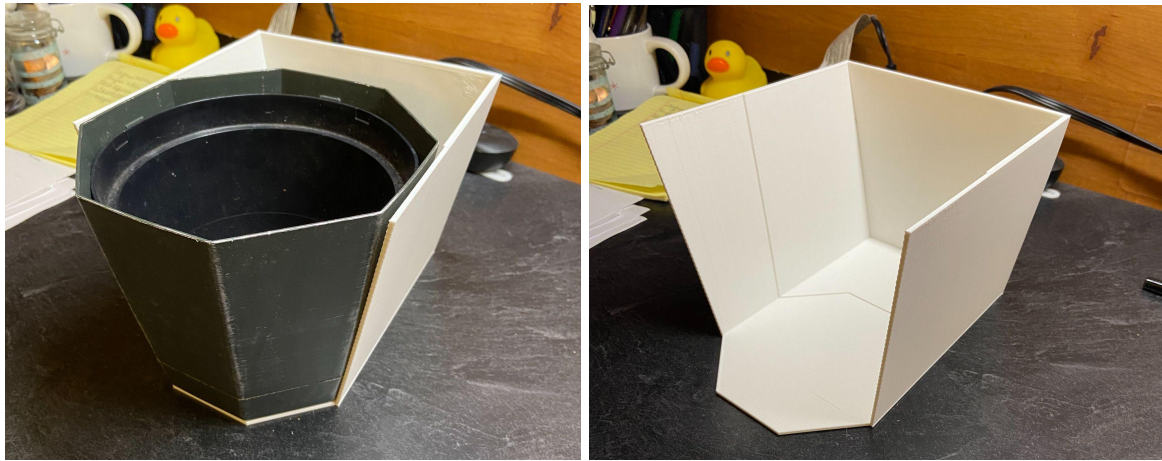
12/13/2021

Introduction :

For CSCI 43300, each group was tasked with creating a project that would be used as the final assessment of what we have learned this semester. For our project Caleb Kirby and Jason Hampshire created a plant monitoring system. This system is created by attaching a variety of sensors to a plant pot and a raspberry pi device. The Internet of Things device takes readings of the various sensors and uses this information to inform a user of the status of the connected plant.

Physical Construction:

The container for this project ended up being a 3D printed model that was attached to the pot with epoxy. Chemical epoxy was used instead of regular glue to be sure there was a good seal between the electronics and the potentially wet soil. The 3D print was modeled and exported to another student who then printed the model.



Sensors Used:

To work as the interface between the raspberry pi and the plant situated in the connected pot, a variety of sensors were used. These sensors were the following:

- Gas Sensor (CCS811)
- Temperature/Humidity Sensor (DHT11)
- Photoresistor
- Soil Moisture Sensor (HW-080)

The gas sensor would be used to monitor the CO₂ levels in the air. Since CO₂ is akin to what oxygen is to humans, an argument could be made that without it, the plant would effectively suffocate. The temperature sensor would be used to monitor just that, the temperature. Many plants are very picky about what temperature they will thrive in. The photoresistor is used to monitor the amount of light the plant receives. Since one of the main sources of energy many plants receive is from photosynthesis, light plays a pivotal role in the health of a plant. Finally, the soil moisture sensor is used to monitor how much water the plant has remaining in the surrounding soil. Without adequate water, a plant will shrivel up and die just like a human would.

Implementation (how it was put together and how it worked):

To implement this project, all of the sensors were connected to the raspberry pi and tested first. After the plant pot was attached to the 3D print, all of the sensors were placed in adequate locations for their corresponding tasks. The soil moisture sensor probe was placed in the soil, the photoresistor was placed where it could get the same sun as the plant, and the other two sensors were left in the protective outcrop with the raspberry pi. Note that since the raspberry pi section is only sealed off from the soil, not the open air, the temperature and gas sensor will work as intended.

Once the raspberry pi is connected to power and powered on, the python scripts that interact with the sensors through software are activated.

Code:

The software is divided up into 5 separate Python scripts. There are 4 scripts for the sensors, 1 for each, and 1 script for coap. co2.py, light.py, soil.py, and temp.py were all used as the sensor interfaces while coap.py was used as the primary user interface.

```
5 import time
6 import board
7 import adafruit_ccs811
8
9 i2c = board.I2C() # uses board.SCL and board.SDA
10 ccs811 = adafruit_ccs811.CCS811(i2c)
11
12 print("initializing co2 monitor")
13 # Wait for the sensor to be ready
14 while not ccs811.data_ready:
15     pass
16 print("done")
17
18 def get_co2():
19     return "{}".format(ccs811.eco2)
20
21 def get_tvoc():
22     return "{}".format(ccs811.tvoc)
23
```

The CO2 sensor was used by co2.py.

```
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")
```

The photoresistor was read by light.py. The soil moisture sensor was read by soil.py.

```
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)
```

```

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

```

The temperature sensor was read by temp.py.

```

1  #!/usr/bin/env python3
2
3  import datetime
4  import logging
5
6  import asyncio
7
8  import aiocoap.resource as resource
9  import aiocoap
10
11 import temp
12 import co2
13 import soil
14 import light
15
16 class TemperatureResource(resource.Resource):
17     def __init__(self):
18         super().__init__()
19
20     def get_temp(self):
21         return bytes(str(temp.read_temp()[1]).encode('UTF-8'))
22
23     async def render_get(self, request):
24         return aiocoap.Message(payload=self.get_temp())
25

```

```

26 class AQResource(resource.Resource):
27     def __init__(self):
28         super().__init__()
29
30     async def render_get(self, request):
31         return aiocoap.Message(payload=co2.get_co2().encode('UTF-8'))
32
33 class TVOCResource(resource.Resource):
34     async def render_get(self, request):
35         return aiocoap.Message(payload=co2.get_tvoc().encode('UTF-8'))
36
37 class SoilResource(resource.Resource):
38     async def render_get(self, request):
39         return aiocoap.Message(payload=soil.coap())
40
41 class LightResource(resource.Resource):
42     async def render_get(self, request):
43         return aiocoap.Message(payload=light.get_light())
44
45 class BlockResource(resource.Resource):
46     """Example resource which supports the GET and PUT methods. It sends large
47     responses, which trigger blockwise transfer."""
48
49     def __init__(self):
50         super().__init__()
51         self.set_content(b"This is the resource's default content. It is padded "
52                         b"with numbers to be large enough to trigger blockwise "
53                         b"transfer.\n")
54
55     def set_content(self, content):
56         self.content = content
57         while len(self.content) <= 1024:
58             self.content = self.content + b"0123456789\n"
59
60     async def render_get(self, request):
61         return aiocoap.Message(payload=self.content)
62
63     async def render_put(self, request):
64         print('PUT payload: %s' % request.payload)
65         self.set_content(request.payload)
66         return aiocoap.Message(code=aiocoap.CHANGED, payload=self.content)

```

```

69 class SeparateLargeResource(resource.Resource):
70     """Example resource which supports the GET method. It uses asyncio.sleep to
71     simulate a long-running operation, and thus forces the protocol to send
72     empty ACK first. """
73
74     def get_link_description(self):
75         # Publish additional data in .well-known/core
76         return dict(**super().get_link_description(), title="A large resource")
77
78     async def render_get(self, request):
79         await asyncio.sleep(3)
80
81         payload = "Three rings for the elven kings under the sky, seven rings "\
82                 "for dwarven lords in their halls of stone, nine rings for "\
83                 "mortal men doomed to die, one ring for the dark lord on his "\
84                 "dark throne.".encode('ascii')
85         return aiocoap.Message(payload=payload)
86

```

```

87 class TimeResource(resource.ObservableResource):
88     """Example resource that can be observed. The `notify` method keeps
89     scheduling itself, and calls `update_state` to trigger sending
90     notifications."""
91
92     def __init__(self):
93         super().__init__()
94
95         self.handle = None
96
97     def notify(self):
98         self.updated_state()
99         self.reschedule()
100
101     def reschedule(self):
102         self.handle = asyncio.get_event_loop().call_later(5, self.notify)
103
104     def update_observation_count(self, count):
105         if count and self.handle is None:
106             print("Starting the clock")
107             self.reschedule()
108         if count == 0 and self.handle:
109             print("Stopping the clock")
110             self.handle.cancel()
111             self.handle = None
112
113     async def render_get(self, request):
114         payload = datetime.datetime.now().\
115             strftime("%Y-%m-%d %H:%M").encode('ascii')
116         return aiocoap.Message(payload=payload)

```

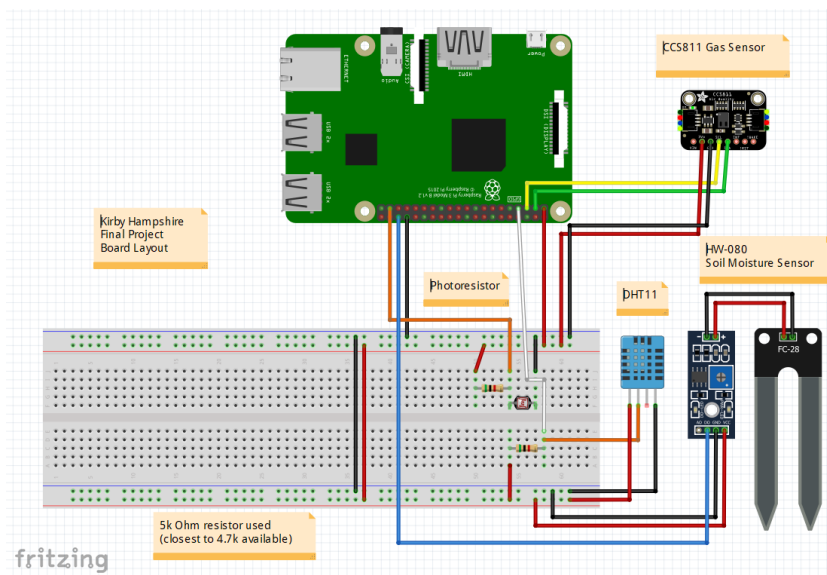
```

118 class WhoAmI(resource.Resource):
119     async def render_get(self, request):
120         text = ["Used protocol: %s." % request.remote.scheme]
121
122         text.append("Request came from %s." % request.remote.hostinfo)
123         text.append("The server address used %s." % request.remote.hostinfo_local)
124
125         claims = list(request.remote.authenticated_claims)
126         if claims:
127             text.append("Authenticated claims of the client: %s." % ", ".join(repr(c) for c in claims))
128         else:
129             text.append("No claims authenticated.")
130
131         return aiocoap.Message(content_format=0,
132                                payload="\n".join(text).encode('utf8'))
133
134 # logging setup
135
136 logging.basicConfig(level=logging.INFO)
137 logging.getLogger("coap-server").setLevel(logging.DEBUG)
138
139 async def main():
140     # Resource tree creation
141     root = resource.Site()
142
143     root.add_resource(['.well-known', 'core'],
144                      resource.WKCRResource(root.get_resources_as_linkheader))
145     root.add_resource(['time'], TimeResource())
146     root.add_resource(['other', 'block'], BlockResource())
147     root.add_resource(['other', 'separate'], SeparateLargeResource())
148     root.add_resource(['whoami'], WhoAmI())
149     root.add_resource(['temp'], TemperatureResource())
150     root.add_resource(['co2'], AQResource())
151     root.add_resource(['tvoc'], TVOCResource())
152     root.add_resource(['soil'], SoilResource())
153     root.add_resource(['light'], LightResource())
154
155
156     await aiocoap.Context.create_server_context(root)
157
158     # Run forever
159     await asyncio.get_running_loop().create_future()
160
161 if __name__ == "__main__":
162     asyncio.run(main())
163

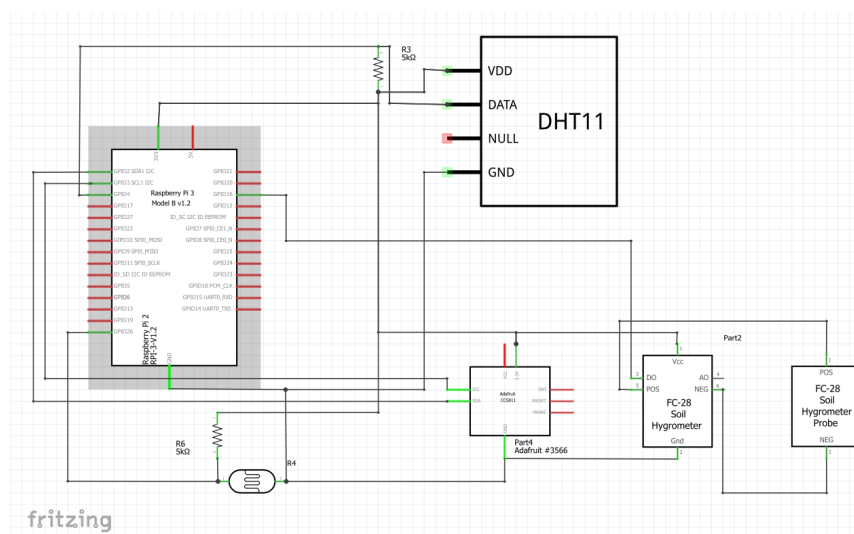
```

CoAP was used and was located inside of coap.py.

Board Diagram (Fritzing board diagram):



Circuit Diagram (Fritzing circuit diagram):



Results and Analysis (how it worked out after implementation):

We were able to successfully develop a system that could monitor plant stats by a remote request. This project was not able to go through very rigorous testing for a number of reasons. Firstly, The plant that was purchased to be the test plant was a Sago Palm plant. Sago Palms are known as strong plants that are hard to kill; however, they also require strong and consistent sunlight. Unfortunately for the Sago Palm, the window in which it was stored did not get very much direct sunlight and as a result, the Palm died much earlier than the project could finish. Due to the season, it was not possible to find a suitable replacement. Secondly is the time scale. For the device to be effective, it would need to be deployed for a long period of time and used to

monitor. Since there was not much time between the conception of this project to it's end, this type of long term testing was not possible.

On the subject of what could be done to continue and expand on this project, creating a system in which any conditions falling out of favorable zones could be automatically corrected. This could be done for each of the four different statistics that were read. For light, a grow light could be turned on when there is low light. For moisture, a servo could be used to open and close a valve that could rehydrate the soil when it is dry. For gas, a tank of CO₂ could be opened and closed in a similar way to a water tank but with a gas valve. For temperature, the system could be attached to a smart thermostat that could trigger the HVAC or Furnace to turn on based on the temperature of the environment.

References and Sources:

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>.