Final Project Report

CSCI 43300

Caleb Kirby / Jason Hamshire

kirbycm@iu.edu / jchampsh@iu.edu

0003607441 / 2000079027

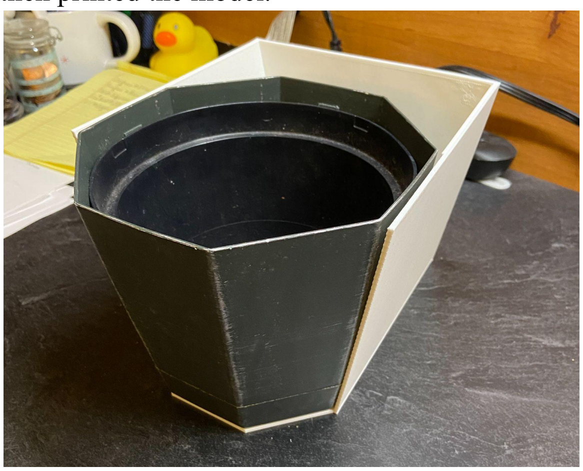
12/13/2021

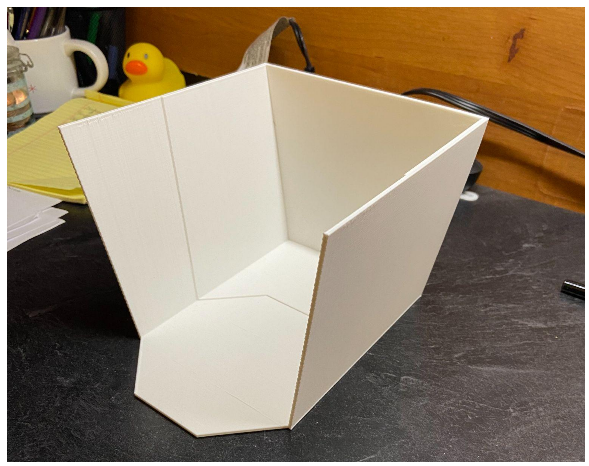
Introduction :

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 CO_2 levels in the air. Since CO_2 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.

```python
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)

```

The CO2 sensor was used by co2.py.

```python
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.

```python
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)
```

The soil moisture sensor was read by soil.py.

```python
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.

```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)
```

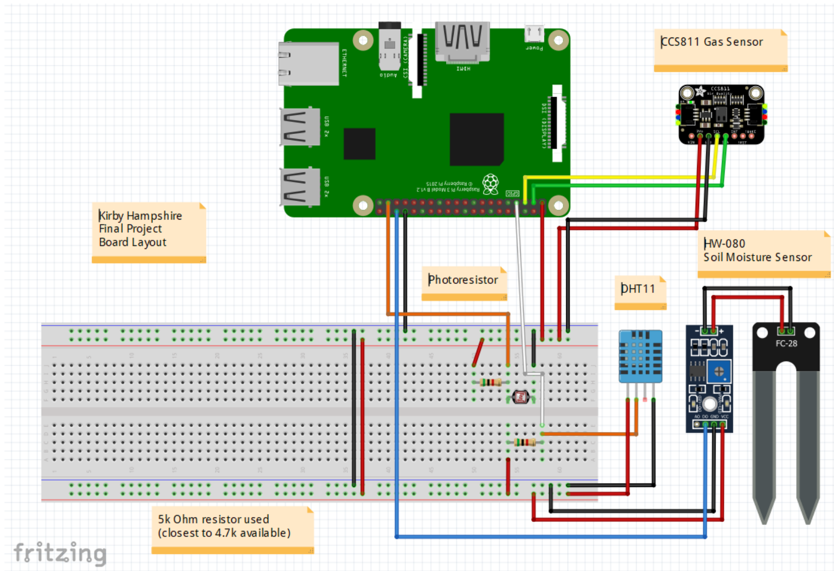
```python
class SeparateLargeResource(resource.Resource):
    """Example resource which supports the GET method. It uses asyncio.sleep to
    simulate a long-running operation, and thus forces the protocol to send
    empty ACK first. """

    def get_link_description(self):
        # Publish additional data in .well-known/core
        return dict(**super().get_link_description(), title="A large resource")

    async def render_get(self, request):
        await asyncio.sleep(3)

        payload = "Three rings for the elven kings under the sky, seven rings "\
                "for dwarven lords in their halls of stone, nine rings for "\
                "mortal men doomed to die, one ring for the dark lord on his "\
                "dark throne.".encode('ascii')
        return aiocoap.Message(payload=payload)
```

```python
class TimeResource(resource.ObservableResource):
    """Example resource that can be observed. The `notify` method keeps
    scheduling itself, and calles `update_state` to trigger sending
    notifications."""

    def __init__(self):
        super().__init__()

        self.handle = None

    def notify(self):
        self.updated_state()
        self.reschedule()

    def reschedule(self):
        self.handle = asyncio.get_event_loop().call_later(5, self.notify)

    def update_observation_count(self, count):
        if count and self.handle is None:
            print("Starting the clock")
            self.reschedule()
        if count == 0 and self.handle:
            print("Stopping the clock")
            self.handle.cancel()
            self.handle = None

    async def render_get(self, request):
        payload = datetime.datetime.now().\
                strftime("%Y-%m-%d %H:%M").encode('ascii')
        return aiocoap.Message(payload=payload)
```

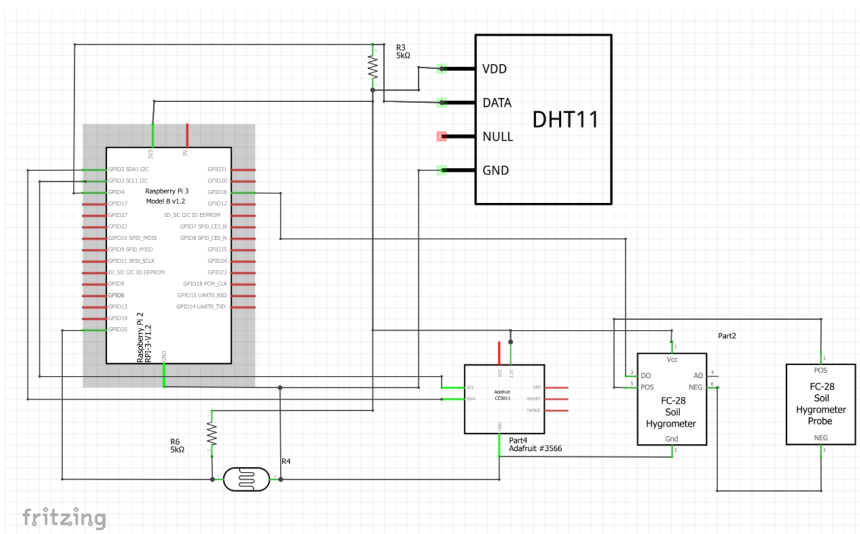
```python
class WhoAmI(resource.Resource):
    async def render_get(self, request):
        text = ["Used protocol: %s." % request.remote.scheme]

        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())
```

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.