University of the Faroe Islands

5025.20 KT-verkætlan

Project

Petur Harald í Kálvalíð & Ragnar Kruse

January 5, 2021

Contents

Chapter 1

Introduction

In this project we will take a look at growing plants in more challenging environments. We will be looking at what makes a plant grow, and what we can do to to make this happen in a controlled environment.

We will make a proof of concept experiment in the form of a practical component, which simulates a limited lighting environment, with controlled lighting and controlled watering for a plant.

Chapter 2

Aim of project

The aim of the project is to find a way to automate the cultivation of plants in a controlled environment, using off-the-shelf components, electronics and a bit of programming. We will pursue this objective with the construction of a practical component. When assembled, we will log data and observe how the experiment unfolds. Based on the data and calculations we word a conclusion if we succeeded to build the plant cultivator and what we managed to achieve.

Chapter 3

Objectives

In order to complete this assignment, we made a few objectives. These objectives are:

- build a main controller to control the plant cultivator as a unit
- build a sensor readout unit to give the main controller sensor data via serial over USB.
- build a box for the plant cultivator. This should contain the plant, the LED lights, the ventilation and all the electrical components.
- build a ventilation system.
- build a web-service to visualize collected data
- assemble everything
- analyze data and write conclusion

3.1 Main Controller Unit

We are going to use a Rasbperry Pi 4 as our MCU. The MCU will read sensor data coming from the Sensor Readout Unit (SRU). The MCU should read sensor data continuously and based on the sensor data and what configuration is loaded during startup it should control the box by turning on/off lights, ventilation and water pump.

The MCU also hosts a local server where a user can retrieve the sensor data being sent from the SRU. Basically we can see how the box is doing. We should also have the option to change schedule of the lights/ventilation and water pump. This should all be done via a browser for ease of use.

3.2 Sensor Readout Unit

For sensor data we are going to use an Arduino Uno. We can very easily read several sensors in a loop and send the data out to the raspberry Pi. The idea is to use a USB cable between the Raspberry Pi 4 and the Arduino Uno board. Not only will the RP4 power the Uno board but it will also be able to receive data via serial through the USB cable. The idea is to have sensor data sent from the Arduino to the RP4 via serial every 2 seconds.

3.3 The Box

We need a way to mount the plant, LEDs, ventilation and all the units and cables. We are going to use a single cupboard, which is going to contain the plant and LEDs. On the back of the cupboard we are going to mount cables, RP4, Arduino and ventilation fans. The idea is to have everything as one unit.

3.4 Ventilation

We need a ventilation system to be able to control the humidity and temperature inside the cupboard, to give the cultivar a preferable environment. We want to mount two 120mm 12V DC fans on the back of the cupboard. We are going to drill two 120mm holes in the back of the cupboard, to allow the fans to pull air through. The fans should be connected in series to a 12V DC power supply, which in turn should supply 6V each, such that the fans would not make undesirable noise.

3.5 Web Service

We need a way for the user to operate the box in a way that is simple and accessible. We want a web user interface that easily runs on any device, and would easily provide the user with the controls that they need.

All the functions that the cupboard has should have a control in the user interface, such that the user may configure the system to their liking and to suite the environmental preference of their cultivar.

We also want a dashboard in which the user may see the current status of the system, in terms of temperature, humidity, etc.

3.6 Assembly

Each separate unit should be individually built and tested first on breadboards and finally soldered to a PCB to eliminate errors. Each part is then connected together via specially made cables that are similar to jumper cables. Sensors are to be stored inside the cupboard and wires soldered and lead through holes in the cupboard to be connected to the Arduino. The raspberry pi should have cables going from the GPIO pins to relays that control the LED/fans and water pump.

3.7 Data Analysis and Conclusion

After finishing assembly we are going to log data for around a day. We are going to log data while the lights are on and while the lights are off. When sufficient data has been collected we will determine if we managed to build a plant cultivation box or not. We can know this by analyzing the data and writing a conclusion.

Chapter 4

Theory

In the theory chapter, we will be looking at what makes our approach to the experiment tick.

We are first going to look at photosynthesis, which is the basis for all life on Earth, then we are going to look at the computer technologies that may be of interest for the experiment.

4.1 Photosynthesis

Photosynthesis is a process which turns light energy into chemical energy. For plants this energy is sugar, which the plant uses to fuel itself, regenerate cells and more.

For photosynthesis to take place for a plant, there has to be a source of water, light, CO_2 and nutrients. When we are growing plants outside there is usually an abundance of these four.

We get water from the rain directly, or later from rivers or clean ground water.

We get light from the sun, which is the byproduct of a fusion reactor.

We get CO_2 from the atmosphere, which is released by all organisms.

We get nutrients from soil, which is a composition of organisms, minerals and liquids.

4.2 Water supply systems

To then grow plants indoor, we loose some of the earth given plant growth automation, namely the water supply.

This then becomes a problem. Traditionally we would use a water container to intermittently supply the plant with. This soon becomes tedious as the number of grown plants increases. More modernly we would consider an automated water supply system.

4.2.1 Rain

The rain is the simplest and easiest way of watering our plants, for most parts of the world. For the more arid regions, rain may not be the best way of relying on water for our plants.

The rain has provided the Earth with a reliable water supply for as long as humans have been on Earth and even longer back than so, and hopefully will continue to do so far into the future.

The Problem with rain is that, it is impractical to control with current technology, and will provide water when it sees fit. Across the globe, there are varying volumes of water that comes from the skies. With different plants requiring different amounts of water, this makes it impractical to grow select plants in a region which may be considered to be too humid for that plant, or where the inverse may also true. This is where a more controlled water supply system may be of greater interest to use than the rain.

4.2.2 Water container

A traditional system for a controlled water supply system is the simple manual water container system. This is the most commonly used system for indoor plant growers. This system requires a fresh water supply, which technically comes from the rain.

The system is labor intensive in a per plant measure, where a small number of plants is feasible for a single person to water in a timely manner, however a green house with hundreds or thousands of plants becomes impractical for this water supply system.

4.2.3 Pump based system

A more modern approach to a controlled water supply system, this system uses an electronic motor driven pump and a computer to control the pump. The pump is used to move water from a container of water to the plant with plastic or metal piping. In this system we would only have a single high powered pump capable of delivering enough pressure to deliver water to an entire green house, since having a single pump per plant would easily brake the promise of scalability of this system.

Using a single pump for the entire green house has some limitations. If we are growing a singular type of produce, we can get away with synchronized watering cycles for all

the plants, but this means that it is impractical to grow multiple types of produce which may require differing watering cycles.

4.2.4 Valve based system

The valve based system fixes one of the crucial limitations with the pump based system, and that is of having a green house of multiple types of produce.

This system uses a per plant electronically motorized valve connected to a computer, with metal piping that runs across the green house connected directly to pressurised fresh water supply. This gives us the ability to control the water supply per plant, and eliminates the need for a pump to provide pressure to the water. This means that we can now grow multiple types of produce in the same green house, and automate varying watering cycles with a computer.

4.3 Lighting systems

Some regions have limited access to direct sunlight, especially during the winter times. This is a factor in the Faroe Islands, where more tropical produce is unable to grow. But for a more extreme example, in Svalbard during the entire winter there is no daylight at all [1].

For these use cases we may consider a more modern controllable lighting system, but if we can it would be preferable to use the sun.

4.3.1 Sun

The tried and tested lighting system that all life on Earth relies on. The sun provides a free source of light energy in the form of photon radiation which comes from the byproduct of the fusion reaction that takes place inside the sun [2].

Most plants thrive with direct sunlight, but some plants prefer a specific wavelength mixture and a specific light cycle to grow optimally.

The same also holds true for the sun as with the rain, the sun is impractical to control.

4.3.2 LED lighting

LEDs have improved greatly over the last few years. They are easy to come by and are very efficient. LEDs are excellent for growing plants since most of LEDs give a good mixture of the red, green and blue color spectrum, and ultraviolet LEDs are also an option. A mixture of multiple types of LEDs are also an option. Another point for the

LED is that they are usually operated with 12V, which gives a simple low-power relay or transistor driver a good interface to a computer.

NASA conducted an experiment [3] with growing plants in space using LEDS. According to their research they used blue and red color wavelengths.

4.4 Nutrient systems

The nutrients that are fed to the plant need not come from traditional soil. Modern agriculture has presented technologies such as the hydroponic and aeroponic nutrient systems. These modern systems allow for a more efficient use of water per kg of produce. According to Ziegler, "to produce a kilogram of tomatoes using traditional land cultivation requires 200 to 400 liters of water, hydroponics requires about 70 liters, aeroponics utilizes only about 20 liters" [4].

4.4.1 Soil

The tried and tested nutrient system is the soil. This system relies on the degradation of organic materials which decompose over a long time. This system is perfect for most plants, but when used for modern agriculture there are some drawbacks to this system.

The primary drawback is the amount of water needed to grow 1 kg of produce. This is a measure of water efficiency which may not be a factor if we are growing in a humid region, but this becomes a major limitation if we are growing in an arid region with limited water supply.

4.4.2 Hydroponic system

A modern approach to supply nutrients to the plants, this technology uses a water based solution including the nutrients that the cultivar in question requires. The roots of the cultivar are exposed directly to this solution. This removes the need for the traditional soil to supply nutrients to the plants.

The hydroponic system can get nutrient in the water based solution from byproduct of other produce, such as commercial fish farming where the excrement from the fish may be used as a nutrient.

This could be an option in the Faroe Islands, where there is large scale commercial fish farming, which could provide nutrients to a hydroponic farming strategy in the future.

Hydroponics are also much more efficient in water usage than traditional soil farming, but for the humid regions this may be a non factor.

4.4.3 Aeroponic system

Another modern approach to a nutrient system, this technology, much like hydroponics, also uses water more directly than the traditional methods.

The aeroponic system sprays a mist of water directly on the roots of the cultivar, where even less water is needed, than the hydroponic system, for the same amount of produce [5].

Aeroponic systems may be of interest for more extreme need of efficiency, where space exploration and colonization is one of the key areas that aeroponic systems are being considered [6].

One of the major drawbacks of the aeroponic system, is the fact that the root itself is suspended in mid air, above the nozzle of a pressurized water supply system. This means that if the water supply system were to fail due to a broken pipe, which also would be more likely to happen, due to the fact that the aeroponic system requires a pressurized water supply system, would render the system ineffective. A failure like this could make the root dry out in just about 2 hours, and would be a catastrophic failure for the whole aeroponic system, and potentially waste valuable resources which a colonization mission in space would potentially not tolerate [6].

4.5 Computer technologies

When we are building automated systems nowadays, we are usually using a computer. The computer will effortlessly control the electric motors that are present in modern water supply systems.

4.5.1 Controlling an electric motor

The simplest way to control an electric motor, is to connect it directly to the computer. But this is easier said than done.

A regular desktop or laptop computer that most everyone today uses, does not easily connect up to a electric motor. We need some sort of a interface.

A single board computer is a small computer where all components are on a single board, as the name suggests. This means that the computer can be extremely compact, and in fact most single board computers will fit in your pocket. What is interesting about this is that it may offer a cheaper alternative to a computer compared to your expensive laptop or desktop computers, and they also easily fit inside small enclosures and back of other electronics.

Why all this is relevant is the development board, a cheap single board computer that easily lends it self to develop electronics that rely on a computer.

An example of such a board is the Raspberry Pi, an inexpensive single board computer. The Raspberry Pi is a single board computer, which has an ARM CPU. It runs Raspbian, a Debian based Linux Distribution. This allows us to run all general purpose code on this small portable low powered computer. The Raspberry Pi is also an excellent development board for electronics, since it has an interface that we can connect our electric motors up to, the General Purpose Input Output (GPIO).

GPIO is a collection of electrical pins, that on the Raspberry Pi can do anything from carrying a serial signal, reading digital sensor data, and sending digital high signals to drive an electric motor.

But what happens when we now connect our electric motor up to our Pi, the motor does nothing. The reason behind this is that the Pi can only deliver about 3.3V 16mA from each pin [7], and for a typical pump electric motor we need upwards of 4.5V 180mA [8].

To make this work, we need a simple electrical circuit, called a driver. There are several types of drivers but the simplest is the relay driver. The relay driver has a control signal, which in this case will be 3.3V 16mA. The control signal opens a relay and lets a supply voltage flow through. The supply voltage can be anything that will carry the voltage and current that the specification demands, in this case 4.5V 180mA for a motorized pump. This could as an example be 3 AA 1.5V Alkaline batteries in series, which would give us 4.5V and about 1A [9].

The end result of this circuit is that with a low powered 3.3v 16mA control signal from a GPIO pin on a Raspberry Pi, we are able to control a supply voltage that is capable of delivering 4.5v 180mA to the pump, and thus the pump will function and be controllable from the Raspberry Pi with general purpose programming languages.

4.5.2 Programming languages

The most popular general purpose programming languages according to the 2020 stackoverflow survey, are JavaScript and Python [10].

These programming languages fall into the category of high level modern programming languages, with an easy to learn and read syntax, dynamic typing, memory safety, garbage collection and an interpreter. The often fact of compiled languages, where the phrase: "but it works on my computer", is all but common speech, with these languages the simplicity of the interpreter makes it work equally across every major operating system that is in use today. This makes these programming languages ideal

for the learner, where the only effort the developer focuses on is solving the problem, not debugging a memory leak.

Both programming languages also future a robust package manager, that has contributed to a flourishing community of 3rd party libraries that solve almost all of the common reoccurring problems in modern software. The current culture is a greater focus on gluing these libraries together, rather than reinventing the wheel every time a new project is started.

These programming languages have in recent years made their way to the professional scene as well. With computers becoming more powerful and cheaper, memory management and requiring a compiled language is a thing of the past, and for the modern developer, this is usually no longer a concern. The modern developer is more concerned on shipping a product on time to the customer.

4.5.3 Python

Python is a house hold programming language that has a lot to offer the modern developer. Python is a great language to quickly realise a prototype, where time to market is of the utmost concern. Python has one of the wealthiest 3rd party library repositories of any programming languages, where the average quality is production ready. This makes it extremely fast to glue libraries together and ship a prototype.

Python is extensively used for controlling electronics, where libraries that can read serial data and libraries that can control GPIO pins are easy to come by. This makes Python an ideal choice for a controller of the drivers needed to operate electric motors.

One of the widely used serial libraries in python is the pyserial library [11].

an example of how to read a serial stream using pyserial,

```
ser = serial.Serial(path, 9600, timeout = 1)
ser.flush()
line = ser.readline().decode('utf-8').rstrip()
```

A Python library to work with GPIO is gpiozero [12]. gpiozero makes it easy to work with GPIO pins, where the pins can be controlled with a LED object, where the functions .on() and .off(), will send a 3.3v digital signal or a 0v digital signal respectively.

A Python library that serves as a programmable web server is the Flask library. Flask is a micro framework, where it is possible to create reasonably advanced web applications, but it also has a simpler functionality, where with just a few lines of code a

programmable web server is up and running. This makes it easy to connect a JavaScript front end up to a Python back end through a number of API endpoints, where data is then sent back and forth.

An example of how to expose an API endpoint using Flask,

```
@app.route('/api/obs')
def getData():
    return json.dumps(obs)
```

APScheduler is a Python library that allows us to schedule jobs, to be run at intervals. These jobs can be anything that fits inside a python function, where routines can be run at set intervals. This would allow us to set a schedule for a water pump to water the plant lets say once every 14 days at 16:00. This would be achived by configuring APScheduler to run a job using the UNIX crontab syntax.

An example of a crontab expression to run a task at 16:00 every 14 days,

```
0 16 */14 * *
```

Another example of a crontab expression to run a task every minute,

```
* * * * *
```

Threading allows us to run multiple tasks concurrently on the same CPU, thus effectively allowing us to run multiple things at the same time in the same program. This is essential when building controllers, since we want things to happen seemingly at the same time, where sensor data is being read, and that data is processed which then runs the motor on the pump based on this data that is processed. This would also allow us to run a web server while at the same time control the pump, and perhaps also in runtime adjust to a user supplied configuration that will change the profile of the watering schedule.

4.5.4 JavaScript

JavaScript is one of the most exciting trends in programming languages today. The reason being is that not quite so recently back in 2009, a new technology was introduced, Node.js [13]. Previously JavaScript had been a web browser client side exclusive programming language, but it was now possible to run that same code on a server. That in of it self is not exemplary, but what there was to come is.

The talk of full stack web development was just starting, previously unheard of. The reality that now it was possible to write websites in a single programming language both

on the server and client was slowly setting in. At the time one of the most used web technology stacks was, Linux Apache MySQL PHP (LAMP), where the programming language was PHP [14]. This meant that to develop websites back in 2009, it was usually necessary to have, a dedicated backend developer that knew PHP, a dedicated frontend developer that knew JavaScript and a dedicated database developer that knew SQL.

With full stack web development, there is a single programming language being used. Since JavaScript was already a requirement for web development, it was destined to become the singular programming language used. Today it is possible to write JavaScript on the server using Node.js, make queries against databases by writing JavaScript instead of SQL using a Object Relational Mapper (ORM), and finally write the same JavaScript on the client that we were already doing back in the day. One programming language for the entire web project. This of course means that today, instead of having close to a requirement on hiring 3 developers, we can get away with a single developer that does it all.

Another vision was slowly developing, a new approach to build web User Interfaces (UI), a focus on developing reusable component based UIs. Where all functionality of a web page were abstracted into a component, what could be thought of as a class in Object Oriented Programming (OOP). This allows a developer to develop components instead of web pages, and then compose these components together to then create a web page. This is a huge shift in paradigm, where before every web page was in of its own, where functionality was specific and baked thoroughly into that one web page. We could now implement the functionality once, and reuse this component across all the web pages that we were maintaining.

Another advancement that aimed to make web pages feel more like a native app on your phone, the Single Page Application (SPA). This is a design for web applications that only loads the page once, and then never reloads. When the user clicks on a link, that link would normally send the user off to a new page, triggering a reload, but in a SPA that link is captured in JavaScript and a router will determine where it is destined, and will render that page as if it functioned like a normal link, but the user never reloaded the page. This makes clicking a link so fast that the user will not notice that any loading took place, since all that loading took place when the user initially loaded the first page. This makes the web application feel like a native app, where everything happens in the snap of a finger.

A framework for building these web applications was released open source by facebook in mid 2013 [15]. Developers could now rely on a standardized way to develop their component based UIs for their SPA.

Further advancements in the full stack vision were later achieved in 2016, with a project

called Next.js [16]. Next.js is a meta framework for the react framework, that allows the developer to write react components, that can execute code on both the client and server. Next.js makes it effortless and highly coherent to program code that runs both on the client and server, in a single file. We can now preprocess the HTML on the server, where we fetch data from the database or deliver a temporary low resolution image, where we then process the client side specific logic after, where we incrementally fetch new data from the database or fetch a suitable resolution image for the client.

This concludes the full stack vision, where a single functionality of the web page can be encapsulated into a single component, and moved from project to project. The only thing that the developer need know is JavaScript, the paradigm of the component based design and the Next.js technology.

An example of a simple React component in Next.js,

index.js

```
import React from 'react';

class Demo extends React.Component
{
  render()
  {
    return (
      <h1>Demo {this.props.passData}</h1>
    );
  }
}

export default class Home extends React.Component
{
  render()
  {
    let first = 1;
    let second = 2 * first;

    return (
      <div>
        <Demo passData={first} />
        <Demo passData={second} />
      </div>
    );
  }
}
```

This would render the HTML,

```
<div>
  <h1>Demo 1</h1>
  <h1>Demo 2</h1>
</div>
```

Chapter 5

Experiment

For our experiment we decided to do a proof of concept, of a limited lighting environment with controlled lighting and controlled watering.

To simulate the environment we have repurposed an old cupboard. The cupboard has a door which when closed lets close to no light inside to the plant. Inside the cupboard we have installed LED lighting, and a pump system to water a single plant. We have also installed fans to keep the micro climate at a controlled temperature and humidity.

We use a Raspberry Pi 4 as our computer.

We use an Arduino UNO as our micro controller to collect all data from our analog and digital sensors, and then send this data through the serial connection over to the Raspberry Pi 4.

The data we are collecting are from a temperature and humidity sensor, to survey the micro climate inside the cupboard for our simulated use case of a limited lighting environment.

Peripheral sensors include a photoresistor to log when lighting is turned on.

All code used for our experiment is found in the appendix.

5.1 Cupboard

The whole plant cultivation box should be seen as one whole unit. There should be space for the plant, LEDs, fans, electronics, water pump, cables, Arduino Uno, Raspberry Pi 4 and all the sensors. The dimensions of the Cupboard is 36 x 28 x 68 cm and has two 120mm holes for ventilation. One at the lower back and one at the upper back of the cupboard.

5.1.1 Pump system

To pump water from the water tank to the plant itself we are going to be using a centrifuge pump. They are usually small and very cheap on aliexpress. We went for something that could run on 5 volts and the RP4 should be able to run it through the GPIO pins via a relay driver. Relays can allow a lot of voltage through, are cheap to come by and can switch using a very low voltage of 3.3V coming from the RP4. The pump is controlled by the scheduler of the MCU. The voltage supply to the pump that goes through the relay comes from the Arduino Uno 5v rails. This means that in reality we are using the 5volt USB power supply from the RP4 as our mean to power the water pump.

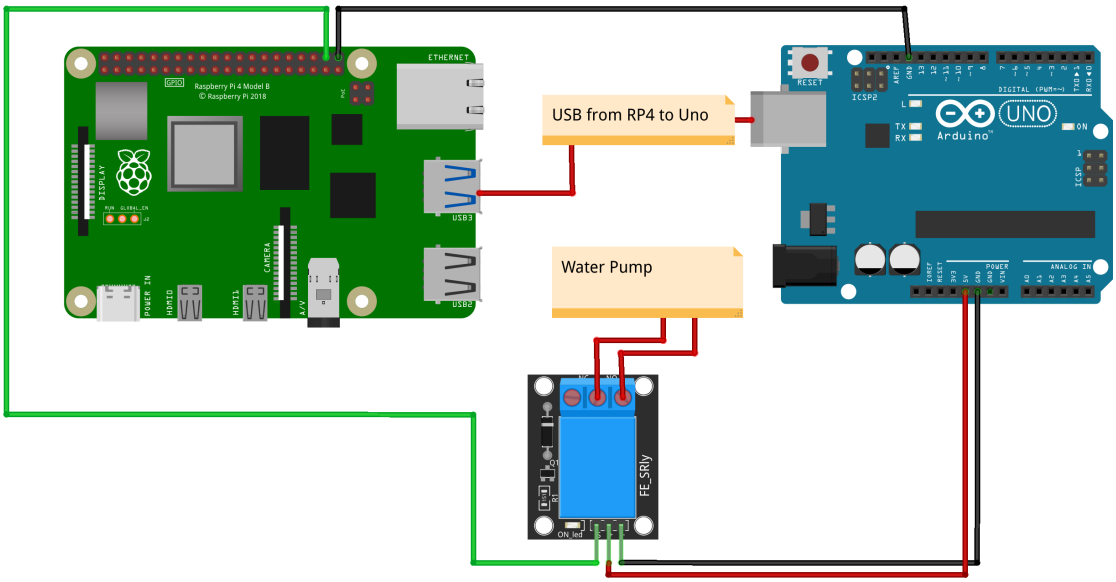

Figure 5.1: Relay connected to RP4 and Arduino Uno.

5.1.2 LEDs

The LEDs that we are using are of type 5050 SMD. The specifications of the LED are as follows:

- IP rating: IP65
- LED Type: SMD 5050
- LED Quantity: 60LED/m
- Color: Red and Blue (Red:Blue = 3:1)

- LED Wavelength: Red: 660nm; Blue: 450nm
- Voltage: DC 12V
- Power: 8W/m, 36W/5m

We are powering the LEDs using a 12 volt powersupply. We feed the power supply through a relay. The relay in turn gets turned on and off by the RP4 using a GPIO pin that is controlled by the scheduler of the MCU.

Here is the diagram of the setup,

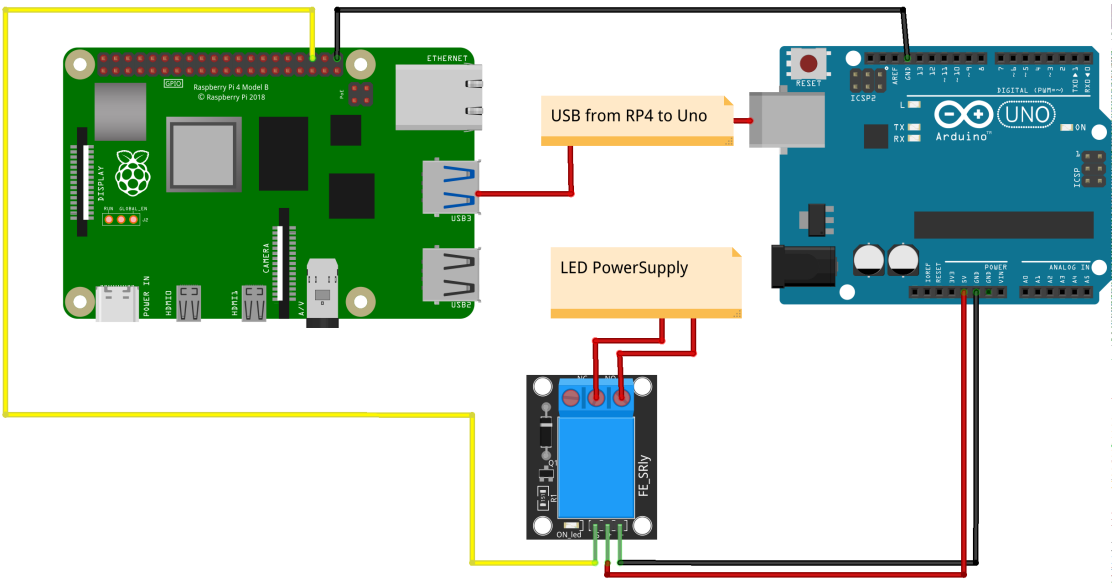

Figure 5.2: Relay connected to RP4 and Arduino Uno.

5.1.3 Fans

The fans are two Lian Li ST120 120mm 12V DC fans. They are connected in series such that each runs at 6V. The Fans get power from the 12V power supply just as the relay shown in figure 5.1 That means that when the LEDs turn on, so does the fans.

The fans spin at 800 and 1900 RPM and are rated to deliver a maximum airflow of 69.17 CFM. This should be more than sufficient for our needs.

5.2 Arduino UNO

The Ardunio UNO is connected to the Pi over a Universal Serial Bus (USB) connection. We use 5V USB from the Pi, to power the Arduino. Here we are using the Uno board as the main sensor readout unit (SRU). We have attached a temperature and humidity sensor, the DHT11 and also a photoresistor. All the data that we get from the sensors are sent to the RP4 over a serial protocol.

5.2.1 DHT11

The temperature and humidity sensor DHT11, gives a digital signal which is then decoded using the DHT Arduino library [17].

It is connected to the Arduino using diagram 5.3.

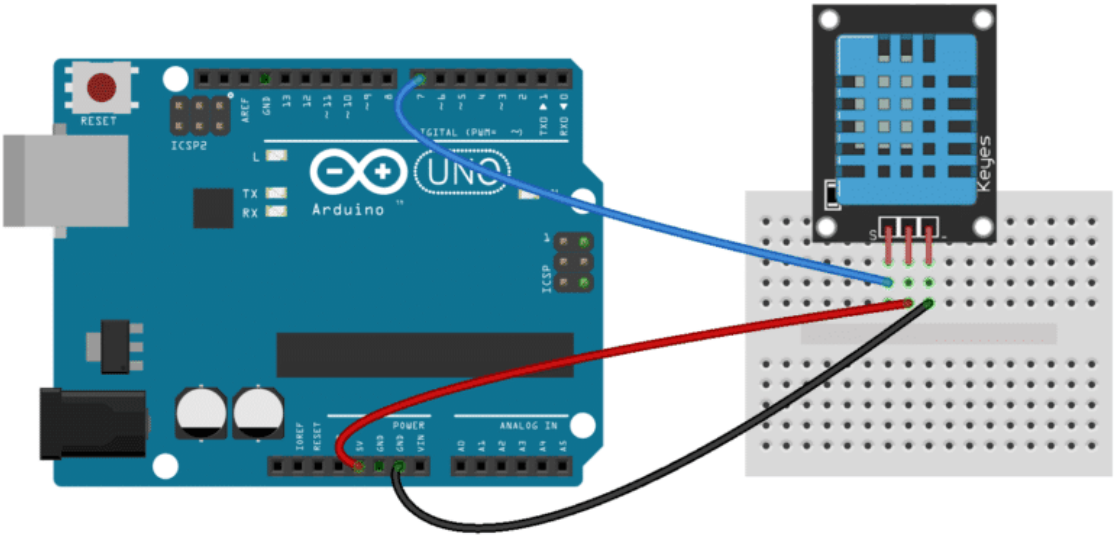

Figure 5.3: DHT11 connected to an Arduino UNO [17].

5.2.2 Photoresistor

The photo resistor changes resistance based on available light. This then gives an analog signal measurable as changes in voltage by the Analog to Digital Converter (ADC) in the Arduino.

It is connected to the Arduino using diagram 5.4.

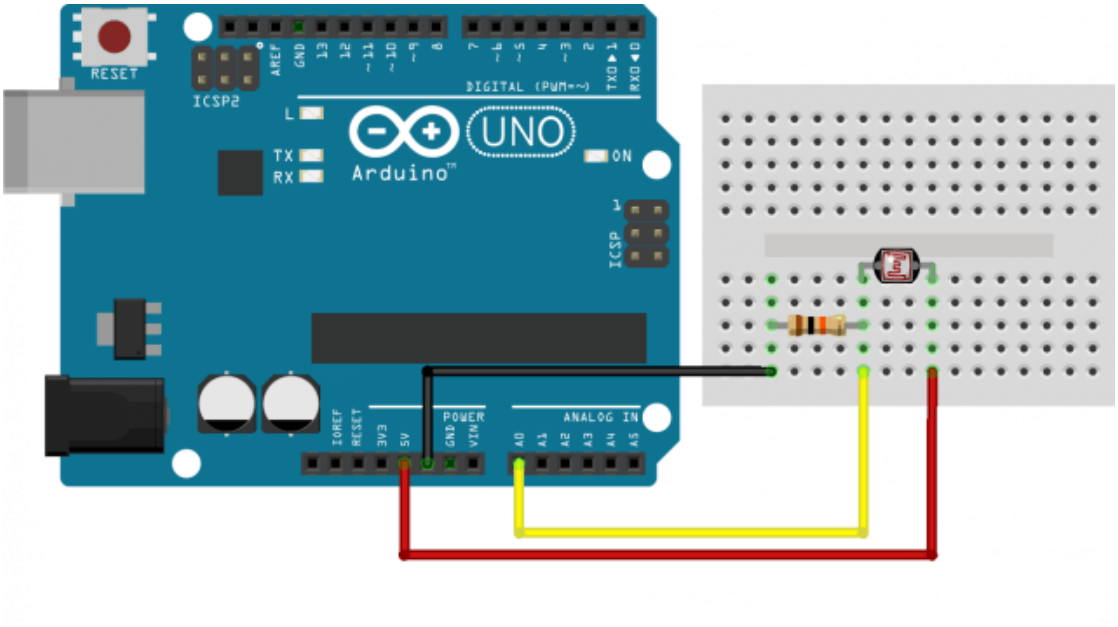

Figure 5.4: Photoresistor connected to an Arduino UNO [18]

5.2.3 Serial

The Arduino is capable of reading and writing data to the Serial object in the standard Arduino library, which then sends data over a serial protocol to the USB connection that is found on the back of the Arduino. This data is then sent to the Pi, as a single line CSV.

An example of this data is the following,

`23.00,41.00,0`

Where the first value is the temprature, second the humidity and lastly the photoresistor.

5.3 Raspberry Pi 4

The Pi handles all the processing of the data that is delivered from the Arduino, and also handles controlling the lighting and the pump. The Pi also is the web server that delivers the web UI.

We use a 5V power supply for the Raspberry PI 4.

5.3.1 Python

For control of our digital electric components from the Pi, we use Python as the programming language. This includes reading serial data from our Arduino, and sending control signals to our lighting and pump system. This also includes sending the data from the Arduino to the web UI.

We use the Flask python library as a programmable web server, which enables us to specify API endpoints that handle HTTP requests [19]. We use Flask to expose a number of endpoints to our Web UI, to pass sensor data and to allow the web UI to control the pump and lighting.

We use the gpiozero library to send 3.3v control signals to our relay drivers, which supply 5V to the pump, and 12V to the LEDs respectively. The relay driver for the LEDs is connected to the GPIO pin 19, and the relay driver for the pump is connected to the GPIO pin 26.

We use APScheduler as a job scheduling library which allows us to run tasks at set intervals, using the common UNIX crontab syntax to define these intervals [20]. Flask exposes 4 endpoints which allows the UI to control the crontab and the second interval for both the lighting and the pump.

We use threads to allow us to control the pump and lighting systems, while simultaneously reading data over the serial connection and sending that data to the web UI, and allowing the UI to reconfigure jobs.

5.3.2 Web user interface

The web user interface (UI) is a simple proof of concept for the controls needed to operate our simulated use case scenario.

The UI serves as a dashboard for the cupboard, where live graphs for temprature and humidity is shown for the last 5 minutes. The UI also lets the user operate the schedules and intervals of both the pump and lighting system. This is achieved through 4 simple text input boxes, with input validation to ensure that the user will provide a good input.

For our UI we use JavaScript as the programming language.

This gives us huge flexibility in terms of running code both on the client and the server in a single React component using Next.js.

We use the JavaScript library Chart.js, to render our charts in a usable and presentable fashion.

We use the Fetch API as the HTTP client to fetch data from our python APIs to our Next.js backend, and then preprocess that data to deliver it with the first http request to the client, and the client then later uses the Fetch API to incrementally fetch new data from the Next.js backend API.

Chapter 6

Data Analysis

To record the data for the data analysis we logged 40 minutes of data. About 20 minutes each where the LEDs where turned on and off respectively.

We chose to only look at the temperature and humidity, to observe if the cupboard was functioning as intended. We want to observe the effects of running without the LEDs and fans active to see if the humidity and temprature inside the cupboard are stable. We also want to observe the effects of then turning on the LEDs and fans. We want to have concrete data to test if the hypothesis, that the fans will negate the extra heating caused by the LEDs, and in turn give a stable micro climate inside the cupboard, holds true or not.

6.1 Temperature

Figure 6.1 shows us the temperature while the LEDs are on and off and as we can see the temperature is not affected by the LEDs at all. This should mean that the airflow is sufficient enough to combat the heat dissipation of the LEDs so that the temperature is stable throughout the light cycle.

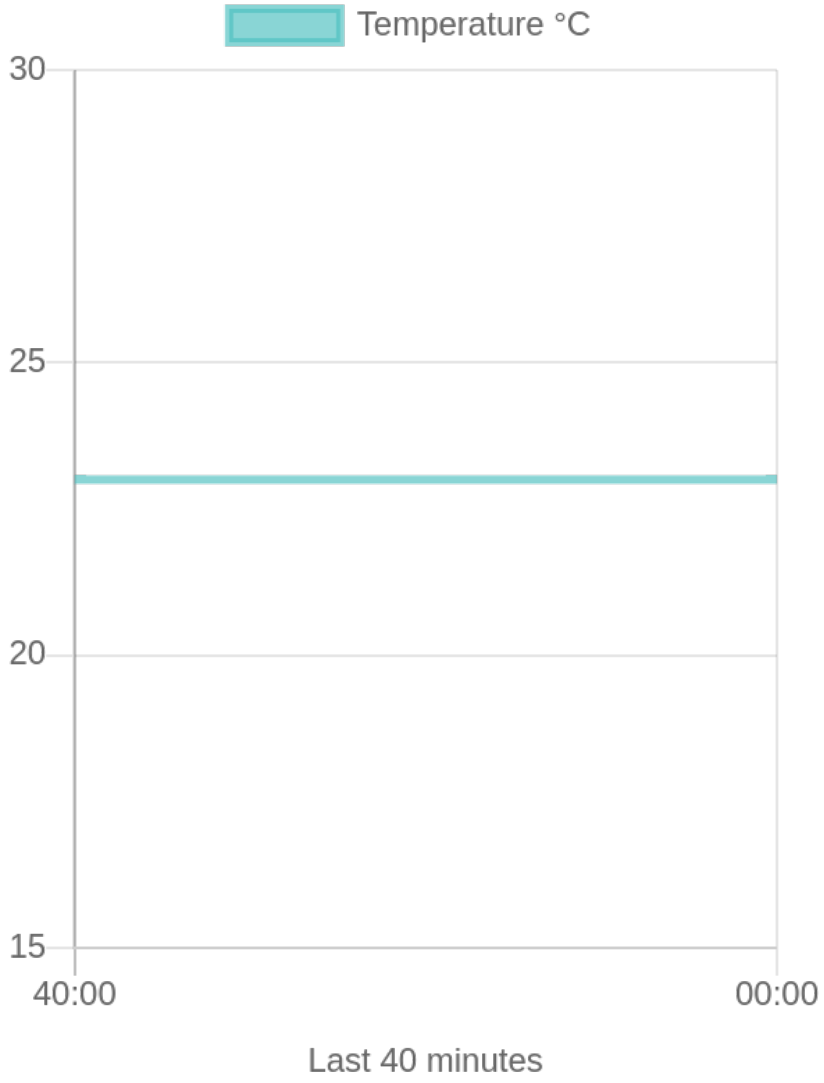

Figure 6.1: 40 minute temperature log.

6.2 Humidity

Figure 6.2 shows us the humidity while the LEDs and fans are on and off. We can see that the humidity is affected by the state of the light cycle. While the LEDs and fans are off the humidity is higher. This can be explained by the fact that the fans are not active and therefore are not sucking any air out of the cupboard. While the LEDs and fans are active the humidity drops by about 5 percent.

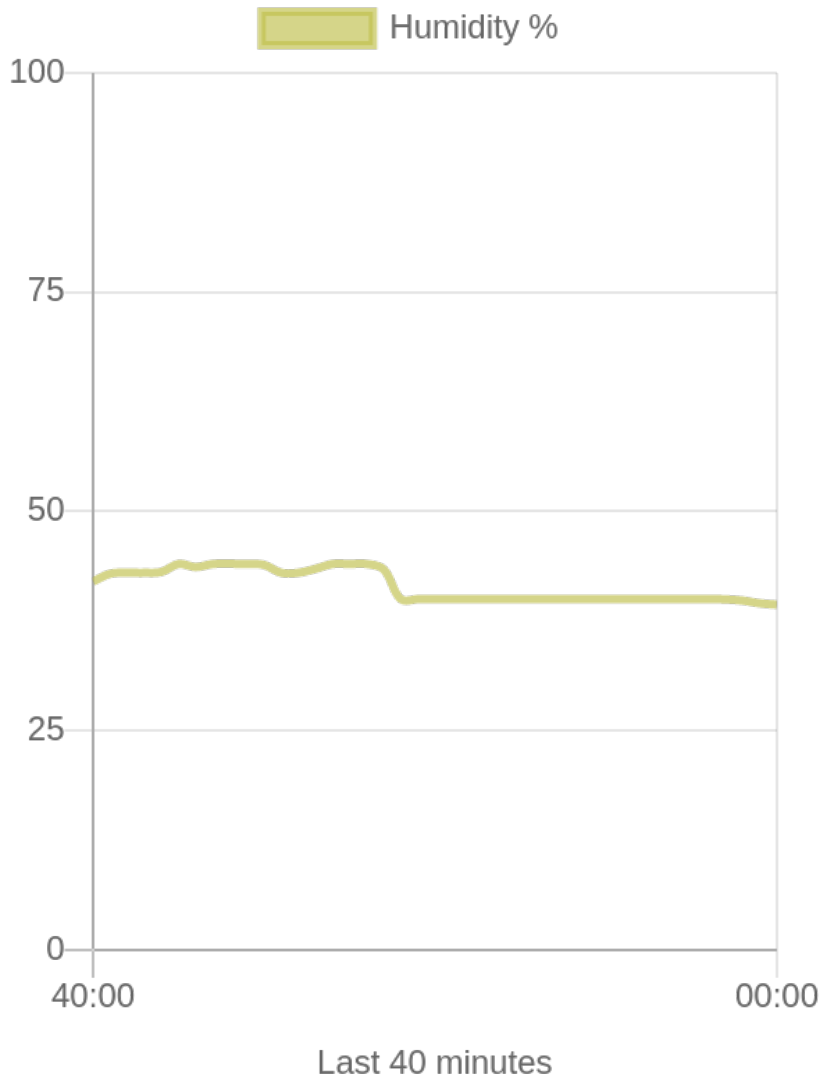

Figure 6.2: 40 minute humidity log.

6.3 Water pump system

We attempted to measure the volume that the water pump system delivered to the plant.

We used a simple scale with an empty container which was about 500ml.

We ran the pump at different intervals and noted if the tube was empty, and what the resulting weight of the water was, and since the weight of water for each gram is about 1 milliliter we do not have to rely on our eyes to measure volume from a more traditional measuring cup.

This is the resulting raw data,

```
time,empty,grams
2000,0,40
2000,0,41
2000,1,27
2000,0,40
2000,1,27
4000,0,89
4000,0,89
4000,0,89
4000,1,77
8000,0,177
8000,0,179
8000,1,168
8000,1,171
8000,1,161
8000,1,169
8000,0,303
```

6.4 Misc raw data

We observed in older logs that the system would occasionally measure an invalid value. It was observed to have happened about the same time as the pump was in an active interval.

an example of such a value is,


```
{
    "id": 135,
    "timestamp": "2020-12-30T15:15:02.724716",
    "temp": "-999.00",
    "humid": "-999.00",
    "light": "1"
}
```

In newer logs, this result is not seen but another behaviour was observed also when the pump was in an active interval.


```
{
    "id": 229,
    "timestamp": "2021-01-04T14:56:58.626226",
    "temp": "23.00",
    "humid": "41.00",
    "light": "0"
},
{
    "id": 230,
    "timestamp": "2021-01-04T14:57:00.632333",
    "temp": "23.00",
    "humid": "41.00",
    "light": "29"
},
{
    "id": 231,
    "timestamp": "2021-01-04T14:57:02.638861",
    "temp": "19.00",
    "humid": "34.00",
    "light": "111"
},
{
    "id": 232,
    "timestamp": "2021-01-04T14:57:04.644467",
    "temp": "23.00",
    "humid": "36.00",
    "light": "0"
},
{
    "id": 233,
    "timestamp": "2021-01-04T14:57:06.750502",
    "temp": "23.00",
    "humid": "41.00",
    "light": "0"
}
```

Chapter 7

Discussion

7.1 Humidity and Temperature

From what we can tell from the data analysis, the LED, fans and box are behaving like we anticipated. We guessed that the temperature would be stable because the fans run at the same time as the LEDs are active. We considered having one fan sucking air out of the box while the LEDs were inactive, this would create some air flow during the inactive light cycle. This could have caused the temperature to decrease somewhat while the LEDs were off. We decided to have both fans running while the LEDs are on so that the temperature would be more stable. This seems to have worked. What we were going for was to have a temperature between 20 and 25 degrees celsius and we have reached this temperature.

We can observe that the humidity is relatively low, and that it is expected to follow somewhat the surrounding environment outside the cupboard. We only have control over the humidity of the cupboard when we also want lighting at the same time. This could prove to be inconvenient when we would like a slightly higher humidity for a particular species of cultivar or the inverse.

7.2 Pump system

We ran some preliminary tests on the water volume of the pump, and a reasonable volume for our proof of concept was found to be about 80ml of water for 4 seconds of pump activation. This varies depending on if the tube from the pump to the plant was empty or not.

Pump activation below 3 seconds did not even push water to the plant, where just

the tube itself was filled, and then gravity pushed the water back down into the water container.

Pump activation for 8 seconds would deliver about 170ml, and as such the dataset is not linear. There is some curve to the dataset at the beginning where there is an extra cost to get the water into the tube, which is diminishing the further away from the threshold we go.

An outlier of interest is the last measurement, where we ran the pump for 8 seconds, and then had the empty container lower than the pump, such that we were working with rather than against gravity. This shows the impact that gravity has on the pump start cost, where we are mostly exclusively fighting gravity.

Pump activation approaching 28 seconds would over fill our test container, which was about 500ml, which caused a puddle of water on our measurement instrument, causing an instrument failure. A decision was then made to cancel the investigation of the water volume of the pump.

Further testing would be beneficial to the project and a curve fitness to a dataset of differing pump activation intervals would be of interest, but was decided to be of lower priority compared to other features of the system.

Other points of note are that the water volume required by the specific plant pot and cultivar used is also a factor which was not investigated in this project.

7.3 LEDs

The LEDs that we use aim to comply to the experiments that NASA have accomplished in space agriculture, both in brightness and a mixture of red and blue wavelengths.

7.4 DHT11 error when operating pump

Occasionally when the pump relay opens, the temperature and humidity sensor fails to read. The activation of the pump seems to affect the accuracy of the measurement of the DHT11 sensor.

At this point we speculate that some noise from the pump is interfering with the DHT11 sensor, or that the pump draws too much current from the Arduino, such that there is not enough current for the DHT11 sensor, but we do not have any data to back this hypothesis up.

This also seems to interfere with the light sensor, and thus the entire Arduino, suggesting that there may be some design flaw that we are not aware of.

The exact cause is unknown at this point, and will not be investigated further since the operation of the pump rarely exceeds a 10s interval. We will allow the temperature, humidity and light sensors to read errors for the duration when the pump is running.

The invalid value is filtered out of the data when the data is then displayed in the graphs on the web UI, so it has little to no effect on the end result of the system.

The only thing that affects the system is that there may be an inaccurate mean calculation where data may be misrepresented to the end user, but the system still functions as a proof of concept.

7.5 Design

We use the simpler single pump per plant water supply system, since we have a single plant. We could had used the more complex but arguably better design of the valve based system, where we need not worry about water pressure. But for this proof of concept the pump system was more than adequate.

We use LEDs, since they are 12V and thus are easily controlled by our computer over a relay driver with a 3.3v control signal and 12V supply for the LEDs.

7.6 DHT11

The temperature and humidity sensor is a digital signal and thus would be possible to connect directly to the Pi, but in our initial research we concluded that it was easiest to use the DHT arduino library, and read the sensor on the Arduino, and then send the data over serial to our Pi. If we were not using an analog photoresistor we would have pursued the possibility of running the DHT11 directly of the Pi, since then we could get away with only using the Pi and not using the Arduino.

7.7 React

The fundamental need for a UI framework in JavaScript is the need to build reusable component based UIs.

Perhaps in this case you could argue that we do not require the ability to build a component based UI, but then again establishing a consistent base for every project, where each project can borrow components from each other, makes it easy to jump start a new project.

There are a number of JavaScript UI frameworks including React, Angular and Vue.js. For our use case, where we need a frequently updating chart with data, we use React due to the built in functionality to only render what has been changed in the component state, which will give us much better performance for our charts.

We could absolutely have went with any of the other frameworks, or even use pure JavaScript, to have achieved the same functionality of the resulting UI.

But since we already have competency working with React, we would have used more time to achieve the same result with other options, and thus using React for our use case was a reasonable choice for us.

Chapter 8

Conclusion

In this project, we have shown that it is possible to create a simulated micro climate that resembles that of a limited lighting environment, with controlled lighting and controlled watering.

We have done research on what makes a plant grow, the essentials of life on earth, and we have used our previous knowledge and skills learned throughout the process of our university degree, to make an experiment, in the form of a practical component, which executes the functions that we intended it to do.

We have collected data and analyzed that data, and can conclude that it is possible to grow a plant in the micro climate that we have presented here.

Appendix A

Code

A.1 Python

raspberrypi/api/requirements.py

```
APScheduler==3.6.3
click==7.1.2
colorzero==1.1
Flask==1.1.2
gpiozero==1.5.1
itsdangerous==1.1.0
Jinja2==2.11.2
MarkupSafe==1.1.1
pigpio==1.44
pyserial==3.5
pytz==2020.4
RPi.GPIO==0.7.0
six==1.15.0
SQLAlchemy==1.3.20
tzlocal==2.1
Werkzeug==1.0.1
```

raspberrypi/api/data.py

```
import flask
import json

import threading
import serial

import gpiozero

import time
import datetime

import atexit
```

```
13 from apscheduler.schedulers.background import BackgroundScheduler
14 from apscheduler.triggers.cron import CronTrigger
15 from apscheduler.jobstores.base import JobLookupError
16
17
18 app = flask.Flask(__name__)
19
20 obs = []
21
22 class Scheduler:
23     def __init__(self, pin):
24         try:
25             self.pin = gpiozero.LED(pin)
26         except gpiozero.exc.BadPinFactory:
27             class MochPin():
28                 def on(self):
29                     pass
30
31                 def off(self):
32                     pass
33             self.pin = MochPin()
34
35         self.jobId = 0
36         self.crontab = ''
37         self.interval = 0
38
39         self.scheduler = BackgroundScheduler(daemon=True)
40         self.scheduler.start()
41         atexit.register(lambda: self.scheduler.shutdown())
42
43     def setInterval(self, interval):
44         if interval < 0.001:
45             raise ValueError
46         self.interval = interval
47
48         print(f"{datetime.datetime.utcnow()} - job set interval {self.interval}")
49
50     def setCrontab(self, crontab):
51         self.crontab = crontab
52
53     def runPinInterval(self, pin, interval):
54         if interval < 0.001:
55             return
56
57         pin.on()
58         print(f"{datetime.datetime.utcnow()} - pin on")
59
60         time.sleep(interval)
61
62         pin.off()
63         print(f"{datetime.datetime.utcnow()} - pin off")
64
```

```
65      def addJob(self):
66          try:
67              self.scheduler.remove_job(str(self.jobId))
68              print(f"{datetime.datetime.utcnow()} - job {self.jobId} stopped")
69          except JobLookupError:
70              pass
71
72          self.jobId += 1
73
74          self.scheduler.add_job(
75              func = self.runPinInterval,
76              args = [self.pin, self.interval],
77              trigger = CronTrigger.from_crontab(self.crontab),
78              id = str(self.jobId)
79          )
80          print(f"{datetime.datetime.utcnow()} - job {self.jobId} started")
81
82  pumpScheduler = Scheduler(26)
83  lightScheduler = Scheduler(19)
84
85  class ReadSerialThread(threading.Thread):
86      def __init__(self, path):
87          threading.Thread.__init__(self)
88
89          self.path = path
90          self.ser = None
91
92          self.loadSerial()
93
94          self.header = ['temp', 'humid', 'light']
95          self.id = 1
96
97      def loadSerial(self):
98          try:
99              self.ser = serial.Serial(self.path, 9600, timeout = 1)
100             self.ser.flush()
101         except serial.serialutil.SerialException:
102             pass
103
104     def innerLoop(self):
105         if self.ser is None:
106             self.loadSerial()
107             return
108
109         if not self.ser.is_open:
110             try:
111                 self.ser.open()
112             except serial.serialutil.SerialException:
113                 pass
114
115         try:
116             if (self.ser.is_open) and (self.ser.in_waiting > 0):
```

```
117                 item = {}
118                 row = self.ser.readline().decode('utf-8').rstrip().split(',')
119 
120                 item['id'] = self.id
121                 self.id += 1
122 
123                 item['timestamp'] = datetime.datetime.utcnow().isoformat()
124 
125                 for i in range(len(self.header)):
126                     item[self.header[i]] = row[i]
127 
128                 obs.append(item)
129         except OSError:
130             self.ser.close()
131 
132     def run(self):
133         while True:
134             self.innerLoop()
135             time.sleep(0.1)
136 
137 ReadSerialThread('/dev/ttyACM0').start()
138 
139 @app.route('/api/obs', methods=['GET'])
140 def getData():
141     return json.dumps(obs, indent = 2, default = str)
142 
143 @app.route('/api/obs/<int:itemId>', methods=['GET'])
144 def getDataId(itemId):
145     result = None
146 
147     for item in obs:
148         if item['id'] == itemId:
149             result = item
150 
151     return json.dumps(result, indent = 2, default = str)
152 
153 @app.route('/api/obs/<int:idFrom>/<int:idTo>', methods=['GET'])
154 def getDataFromTo(idFrom, idTo):
155     result = []
156 
157     for item in obs:
158         if (item['id'] >= idFrom) and (item['id'] <= idTo):
159             result.append(item)
160 
161     return json.dumps(result, indent = 2, default = str)
162 
163 @app.route('/api/obs/lastDay')
164 def getObsLastDay():
165     result = []
166 
167     lastDay = datetime.datetime.utcnow() - datetime.timedelta(hours=24)
168 
```

```
169     for item in obs:
170         if datetime.datetime.fromisoformat(item['timestamp']) > lastDay:
171             result.append(item)
172 
173     return json.dumps(result, indent = 2, default = str)
174 
175 @app.route('/api/obs/newest')
176 def getObsNewest():
177     result = None
178 
179     try:
180         result = obs[-1]
181     except IndexError:
182         pass
183 
184     return json.dumps(result, indent = 2, default = str)
185 
186 @app.route('/api/pump/schedule', methods=['POST'])
187 def addPumpSchedule():
188     statusCode = 0
189 
190     try:
191         data = str(flask.request.get_json(force=True)['data'])
192 
193         pumpScheduler.setCrontab(data)
194         pumpScheduler.addJob()
195     except ValueError:
196         statusCode = -1
197     except:
198         statusCode = -2
199 
200     return json.dumps({'status': statusCode}, indent = 2, default = str)
201 
202 @app.route('/api/pump/interval', methods=['POST'])
203 def setPumpInterval():
204     statusCode = 0
205 
206     try:
207         data = float(flask.request.get_json(force=True)['data'])
208 
209         pumpScheduler.setInterval(data)
210         pumpScheduler.addJob()
211     except ValueError:
212         statusCode = -1
213     except:
214         statusCode = -2
215 
216     return json.dumps({'status': statusCode}, indent = 2, default = str)
217 
218 @app.route('/api/light/schedule', methods=['POST'])
219 def addLightSchedule():
220     statusCode = 0
```

```
221
222     try:
223         data = str(flask.request.get_json(force=True)['data'])
224
225         lightScheduler.setCrontab(data)
226         lightScheduler.addJob()
227     except ValueError:
228         statusCode = -1
229     except:
230         statusCode = -2
231
232     return json.dumps({'status': statusCode}, indent = 2, default = str)
233
234 @app.route('/api/light/interval', methods=['POST'])
235 def setLightInterval():
236     statusCode = 0
237
238     try:
239         data = float(flask.request.get_json(force=True)['data'])
240
241         lightScheduler.setInterval(data)
242         lightScheduler.addJob()
243     except ValueError:
244         statusCode = -1
245     except:
246         statusCode = -2
247
248     return json.dumps({'status': statusCode}, indent = 2, default = str)
249
250 if __name__ == '__main__':
251     app.run()
```

A.2 Web User Interface

raspberrypi/ui/package.json

```
1  {
2    "name": "ui",
3    "version": "0.1.0",
4    "private": true,
5    "scripts": {
6      "dev": "next dev",
7      "build": "next build",
8      "start": "next start"
9    },
10   "dependencies": {
11     "chart.js": "^2.9.4",
12     "next": "10.0.4",
13     "react": "17.0.1",
14     "react-chartjs-2": "^2.11.1",
15     "react-dom": "17.0.1"
```

```
16   }
17 }
```

raspberrypi/ui/pages/_app.js

```
1 import '../styles/globals.css'
2
3
4 function MyApp({ Component, pageProps }) {
5   return <Component {...pageProps} />
6 }
7
8 export default MyApp
```

raspberrypi/ui/pages/index.js

```
1 import React from 'react';
2 import styles from '../styles/Home.module.css';
3
4 import Head from 'next/head';
5 import LineGraph from '../components/LineGraph';
6 import TextField from '../components/TextField';
7
8
9 export async function getServerSideProps()
10 {
11   const res = await fetch('http://localhost:3000/api/obs/lastDay');
12   const data = await res.json();
13
14   return { props: { data } };
15 }
16
17 export default class Home extends React.Component
18 {
19   constructor(props)
20   {
21     super(props);
22
23     this.state = {
24       'data': this.props.data
25     };
26   }
27
28   getStartId()
29   {
30     if (this.state.data.length > 0)
31       return this.state.data[this.state.data.length - 1].id + 1;
32     return 1;
33   }
34
35   componentDidMount()
```

```
36  {
37    let newest = () => {
38      fetch(`http://localhost:3000/api/obs/newest`)
39        .then(j => j.json())
40        .then(newest => {
41          fetch(
42            'http://localhost:3000/api/obs/getRange',
43            {
44              'method': 'POST',
45              'body': JSON.stringify({'start': this.getStartId(), 'end': newest.id})
46            }
47          )
48            .then(j => j.json())
49            .then(missing => {
50              let data = this.state.data;
51
52              for (let item of missing)
53              {
54                let match = false;
55
56                for (let innerItem of data)
57                {
58                  if (item.id == innerItem.id)
59                  {
60                    match = true;
61                  }
62                }
63
64                if (!match)
65                {
66                  data.push(item);
67                }
68              }
69
70              this.setState({ data });
71            });
72        })
73        .catch(() => { });
74    }
75
76    setInterval(newest, 10000);
77  }
78
79  render()
80  {
81    // https://stackoverflow.com/a/7036361
82    let rePositiveInteger = /^[1-9]\d*$/;
83
84    // https://stackoverflow.com/a/17858524
85    let reCrontab = /^(\*|([0-9]|1[0-9]|2[0-9]|3[0-9]|4[0-9]|5[0-9])
    |\*\/([0-9]|1[0-9]|2[0-9]|3[0-9]|4[0-9]|5[0-9])) (\*|([0-9]|1[0-9]|2[0-3])
    |\*\/([0-9]|1[0-9]|2[0-3])) (\*|([1-9]|1[0-9]|2[0-9]|3[0-1])
```

```
    |\*\/([1-9]|1[0-9]|2[0-9]|3[0-1])) (\*|([1-9]|1[0-2])|\*\/([1-9]|1[0-2]))
    (\*|([0-6])|\*\/([0-6]))$/;
86
87      return (
88        <div style={{background: '#181818', minHeight: '100vh'}}>
89          <Head>
90            <title>Create Next App</title>
91            <link rel="icon" href="/favicon.ico" />
92          </Head>
93
94          <div className={styles.grid}>
95            <LineGraph
96              data={this.state.data}
97              dataKey={'temp'}
98              dataMin={16}
99              dataMax={30}
100             text={'Temperature  C '}
101             color={'rgba(75,192,192,0.65)'}
102             textColor={'#FFF'}
103           />
104           <LineGraph
105             data={this.state.data}
106             dataKey={'humid'}
107             dataMin={20}
108             dataMax={80}
109             text={'Humidity %'}
110             color={'rgba(192,192,75,0.65)'}
111             textColor={'#FFF'}
112           />
113         </div>
114         <div className={styles.scheduleControls}>
115           <div className={styles.pumpScheduleControls}>
116             <TextField
117               text={'Pump Schedule (cron)'}
118               dataKey={'pumpSchedule'}
119               placeHolder={'* * * * *'}
120               url={'http://localhost:3000/api/pump/schedule'}
121               re={reCrontab}
122             />
123
124             <TextField
125               text={'Pump Interval (s)'}
126               dataKey={'pumpInterval'}
127               placeHolder={'2'}
128               url={'http://localhost:3000/api/pump/interval'}
129               re={rePositiveInteger}
130             />
131           </div>
132           <div className={styles.lightScheduleControls}>
133             <TextField
134               text={'Light Schedule (cron)'}
135               dataKey={'lightSchedule'}
```

```
136                 placeHolder={'* * * * *'}
137                 url={'http://localhost:3000/api/light/schedule'}
138                 re={reCrontab}
139               />
140
141               <TextField
142                 text={'Light Interval (s)'}
143                 dataKey={'lightInterval'}
144                 placeHolder={'21600'}
145                 url={'http://localhost:3000/api/light/interval'}
146                 re={rePositiveInteger}
147               />
148             </div>
149           </div>
150         </div>
151       );
152     }
153 }
```

raspberrypi/ui/pages/api/light/interval.js

```
1 export { default } from '../../../helpers/APIForward';
```

raspberrypi/ui/pages/api/light/schedule.js

```
1 export { default } from '../../../helpers/APIForward';
```

/raspberrypi/ui/pages/api/obs/getRange.js

```
1  export default async function APIForward(req, res)
2  {
3    if (req.method === 'POST')
4    {
5      let body = JSON.parse(req.body);
6
7      let data = await fetch(`http://localhost:5000/api/obs/${body.start}/${body.end}`);
8      data = await data.json();
9
10     res.status(200)
11       .json(data);
12   }
13
14   res.status(405)
15     .end();
16 }
```

/raspberrypi/ui/pages/api/obs/lastDay.js

```
1 export { default } from '../../../helpers/APIForward';
```

/raspberrypi/ui/pages/api/obs/newest.js

```
export { default } from '../../../helpers/APIForward';
```

raspberrypi/ui/pages/api/pump/interval.js

```
export { default } from '../../../helpers/APIForward';
```

raspberrypi/ui/pages/api/pump/schedule.js

```
export { default } from '../../../helpers/APIForward';
```

raspberrypi/ui/helpers/APIForward.js

```
export default async function APIForward(req, res)
{
  let url = `http://localhost:5000${req.url}`;

  if (req.method === 'GET')
  {
    let data = await fetch(url);
    data = await data.json();

    res.status(200)
      .json(data);
  }

  if (req.method === 'POST')
  {
    let data = await fetch(
      url,
      {
        'method': 'POST',
        'body': req.body
      }
    );
    data = await data.json();

    res.status(200)
      .json(data);
  }

  res.status(405)
    .end();
}
```

raspberrypi/ui/components/LineGraph.js

```
import React from 'react';
import styles from '../styles/LineGraph.module.css';
```

```
3
4  import { Line } from 'react-chartjs-2';
5
6
7  export default class Graph extends React.Component
8  {
9    render()
10   {
11     let label = [];
12     let data = [];
13
14     const now = new Date();
15
16     let minuteData = [];
17
18     for (let item of this.props.data)
19     {
20       if ((now - new Date(item['timestamp'])) <= 5*60*1010)
21       {
22         minuteData.push(item);
23       }
24     }
25
26     for (let i = 0; i <= 5*6; i++)
27     {
28       let sum = 0;
29       let count = 0;
30
31       for (let item of minuteData)
32       {
33         if (
34           ((now - new Date(item['timestamp'])) >= (i)*10000) &&
35           ((now - new Date(item['timestamp'])) <= (i+1)*10000)
36         )
37         {
38           let dataKey = parseFloat(item[this.props.dataKey]);
39
40           if (dataKey > -999.00)
41           {
42             sum += dataKey;
43             count += 1;
44           }
45         }
46       }
47
48       data.push((sum / count).toFixed(2));
49       label.push((i)*10000);
50     }
51
52     let lineData = {
53       labels: label,
54       datasets: [
```

```
55          {
56            label: this.props.text,
57            fill: false,
58            pointRadius: 0,
59            pointHoverRadius: 0,
60            pointHitRadius: 7,
61            borderColor: this.props.color,
62            backgroundColor: this.props.color,
63            data: data,
64          }
65        ]
66      };
67
68      let lineOptions = {
69        responsive: true,
70        maintainAspectRatio: false,
71        legend: {
72          position: 'top',
73          labels: {
74            fontColor: this.props.textColor
75          },
76        },
77        scales: {
78          xAxes: [
79            {
80              type: 'time',
81              time: {
82                unit: 'minute',
83                displayFormats: {
84                  minute: 'm:ss'
85                },
86                tooltipFormat:'m:ss',
87              },
88              ticks: {
89                fontColor: 'inherit',
90                max: 5*60*1001,
91                min: 0,
92                stepSize: 0,
93                maxTicksLimit: 1,
94                maxRotation: 0,
95                minRotation: 0,
96                reverse: true,
97              },
98              gridLines: {
99                color: 'rgba(255,255,255,0.12)',
100               zeroLineColor: 'rgba(255,255,255,0.12)',
101             },
102             scaleLabel: {
103               fontColor: 'inherit',
104               display: true,
105               labelString: 'Last 5 minutes'
106             },
```

```
            }
          ],
          yAxes: [
            {
              ticks: {
                fontColor: 'inherit',
                suggestedMax: this.props.dataMax,
                suggestedMin: this.props.dataMin,
                stepSize: 0.5,
                maxTicksLimit: 6,
                maxRotation: 0,
                minRotation: 0,
              },
              gridLines: {
                color: 'rgba(255,255,255,0.12)',
                zeroLineColor: 'rgba(255,255,255,0.12)',
              },
            }
          ],
        }
      }

      return (
        <div className={styles.container}>
          <Line
            data={lineData}
            options={lineOptions}
          />
        </div>
      );
    }
}
```

raspberrypi/ui/components/TextField.js

```
import React from 'react';
import styles from '../styles/TextField.module.css';


export default class TextField extends React.Component
{
  constructor(props)
  {
    super(props);

    this.event = {};
    this.element = React.createRef();
  }

  changeInputColor(color)
  {
    this.event.target.elements[0].style = `border-color: ${color}`;
```

```
18      new Promise(res => setTimeout(res, 2000))
19        .then(() => { this.event.target.elements[0].style = 'border-color: transperent';
      });
20    }
21
22    componentDidMount()
23    {
24      this.element.current.onsubmit = (event) => {
25        this.event = event;
26        this.event.preventDefault();
27
28        let input = this.event.target.elements[0].value;
29
30        // Test if crontab is valid
31        if ((this.props.re) && (!this.props.re.test(input)))
32        {
33          this.changeInputColor('red');
34          return;
35        }
36
37        fetch(
38          this.props.url,
39          {
40            'method': 'POST',
41            'body': JSON.stringify({'data': input})
42          }
43        )
44          .then(body => body.json())
45          .then(j => console.log(j));
46
47        this.changeInputColor('green');
48      }
49    }
50
51    render()
52    {
53      return (
54        <form ref={this.element} className={styles.form}>
55          <label className={styles.label} htmlFor={this.props.dataKey}>{this.props.text
      }</label>
56            <input className={styles.input} type="text" id={this.props.dataKey} name={this.
      props.dataKey} placeholder={this.props.placeHolder}/>
57        </form>
58      );
59    }
60  }
```

raspberrypi/ui/styles/globals.css

```
1  html, body
2  {
3    padding: 0;
```

```
  margin: 0;
  font-family: -apple-system, BlinkMacSystemFont, Segoe UI, Roboto, Oxygen,
    Ubuntu, Cantarell, Fira Sans, Droid Sans, Helvetica Neue, sans-serif;
  color: #FFF;
  background: '#181818';
}

a
{
  color: inherit;
  text-decoration: none;
}

*
{
  box-sizing: border-box;
}
```

raspberrypi/ui/styles/Home.module.css

```
.container {
  min-height: 100vh;
  padding: 0 0.5rem;
  display: flex;
  flex-direction: column;
  justify-content: center;
  align-items: center;
}

.main {
  padding: 5rem 0;
  flex: 1;
  display: flex;
  flex-direction: column;
  justify-content: center;
  align-items: center;
}

.footer {
  width: 100%;
  height: 100px;
  border-top: 1px solid #eaeaea;
  display: flex;
  justify-content: center;
  align-items: center;
}

.footer img {
  margin-left: 0.5rem;
}

.footer a {
```

```
33    display: flex;
34    justify-content: center;
35    align-items: center;
36  }
37
38  .title a {
39    color: #0070f3;
40    text-decoration: none;
41  }
42
43  .title a:hover,
44  .title a:focus,
45  .title a:active {
46    text-decoration: underline;
47  }
48
49  .title {
50    margin: 0;
51    line-height: 1.15;
52    font-size: 4rem;
53  }
54
55  .title,
56  .description {
57    text-align: center;
58  }
59
60  .description {
61    line-height: 1.5;
62    font-size: 1.5rem;
63  }
64
65  .code {
66    background: #fafafa;
67    border-radius: 5px;
68    padding: 0.75rem;
69    font-size: 1.1rem;
70    font-family: Menlo, Monaco, Lucida Console, Liberation Mono, DejaVu Sans Mono,
71      Bitstream Vera Sans Mono, Courier New, monospace;
72  }
73
74  .grid
75  {
76    display: flex;
77    align-items: center;
78    justify-content: center;
79    flex-wrap: wrap;
80  }
81
82  .card {
83    margin: 1rem;
84    flex-basis: 45%;
```

```
85    padding: 1.5rem;
86    text-align: left;
87    color: inherit;
88    text-decoration: none;
89    border: 1px solid #eaeaea;
90    border-radius: 10px;
91    transition: color 0.15s ease, border-color 0.15s ease;
92  }
93
94  .card:hover,
95  .card:focus,
96  .card:active {
97    color: #0070f3;
98    border-color: #0070f3;
99  }
100
101 .card h3 {
102   margin: 0 0 1rem 0;
103   font-size: 1.5rem;
104 }
105
106 .card p {
107   margin: 0;
108   font-size: 1.25rem;
109   line-height: 1.5;
110 }
111
112 .logo {
113   height: 1em;
114 }
115
116 .scheduleControls
117 {
118
119 }
120
121 @media (max-width: 799px)
122 {
123   .grid
124   {
125     width: 100%;
126     flex-direction: column;
127   }
128 }
```

raspberrypi/ui/styles/LineGraph.module.css

```
1  .container
2  {
3    height: 300px;
4    margin-top: 60px;
5  }
```

```
6
7  @media (min-width: 800px)
8  {
9    .container
10   {
11     height: 400px;
12     margin-left: 30px;
13     margin-right: 30px;
14   }
15 }
```

raspberrypi/ui/styles/TextField.module.css

```
1  .form
2  {
3    margin-top: 60px;
4    text-align: center;
5  }
6
7  .label
8  {
9    display: block;
10   margin-bottom: 10px;
11 }
12
13 .input
14 {
15   padding: 5px;
16   border-radius: 5px;
17   border-width: 3px;
18   border-color: transparent;
19   border-style: solid;
20   outline: none;
21   width: 120px;
22   text-align: center;
23 }
24
25 .input:focus
26 {
27   border-color: #29b6f6;
28 }
```

Bibliography

[1] Svalbard daylight hours. https://www.timeanddate.com/sun/@7521757. Accessed: 2020-12-30.

[2] K.D. Abhyankar. A survey of the solar atmospheric models. *Bulletin of the Astronomical Society of India*, 5:40 – 44, 1977.

[3] Led lights used in plant growth experiments for deep space missions. https://www.nasa.gov/centers/kennedy/home/plant_growth.html. Accessed: 2020-01-03.

[4] Reinhold Ziegler Synergy International Inc. The vertical aeroponic growing system, 2005.

[5] Paul Demo Margaret Chiipanthenga1, Moses Maliro1 and Joyce Njoloma. Potential of aeroponics system in the production ofquality potato (solanum tuberosum l.) seed indeveloping countries. *African Journal of Biotechnology*, 11:3993 – 3999, 2012.

[6] Matthew B. Rogers Peter Q. Storm Bryant Z. Bennett, Jonathan C. Overton. Greenhouse systems for deployment on mars.

[7] General purpose input output (gpio) pins. https://www.tomshardware.com/reviews/raspberry-pi-gpio-pinout,6122.html. Accessed: 2020-01-02.

[8] 2018 brand new micro submersible and amphibious dc motor pump water pump 3/4.5v 80-100l/h europe drop shipping. https://www.aliexpress.com/item/32894401053.html?spm=a2g0o.productlist.0.0.17461159Czz9KO&algo_pvid=7608455c-fdcb-46c8-9674-f432c6f7213e&algo_expid=7608455c-fdcb-46c8-9674-f432c6f7213e-0&btsid=2100bdde16096258833253652e2890&ws_ab_test=searchweb0_0,searchweb201602_,searchweb201603_. Accessed: 2020-01-02.

[9] How to power a project. https://learn.sparkfun.com/tutorials/how-to-power-a-project/remotemobile-power. Accessed: 2020-01-03.

[10] Programming, scripting, and markup languages. https://insights.stackoverflow.com/survey/2020#technology-programming-scripting-and-markup-languages-all-respondents. Accessed: 2020-01-03.

[11] Short introduction. https://pythonhosted.org/pyserial/shortintro.html. Accessed: 2020-01-02.

[12] gpiozero. https://gpiozero.readthedocs.io/en/stable/. Accessed: 2020-01-02.

[13] About node.js, and why you should add node.js to your skill set? http://blog.training.com/2016/09/about-nodejs-and-why-you-should-add.html. Accessed: 2020-01-03.

[14] Programming language popularity. https://spring.io/blog/2009/02/23/programming-language-popularity. Accessed: 2020-01-03.

[15] Facebook's react javascript user interfaces library receives mixed reviews. https://www.infoq.com/news/2013/06/facebook-react/. Accessed: 2020-01-03.

[16] 1.0.0. https://github.com/vercel/next.js/releases/tag/1.0.0. Accessed: 2020-01-03.

[17] How to set up the dht11 humidity sensor on an arduino. https://www.circuitbasics.com/how-to-set-up-the-dht11-humidity-sensor-on-an-arduino/. Accessed: 2020-01-02.

[18] Luminosity measurement with a photoresistor. https://www.aranacorp.com/en/luminosity-measurement-with-a-photoresistor/. Accessed: 2020-01-03.

[19] Quickstart. https://flask.palletsprojects.com/en/1.1.x/quickstart/. Accessed: 2020-01-02.

[20] User guide. https://apscheduler.readthedocs.io/en/stable/userguide.html. Accessed: 2020-01-02.