# Intelligent Control of Hydroponic Farm

E6765_2018Spring_*5555*_report
Yuval Schaal ys3055, Ido Michael im2482, Siao-Ting Wang sw3092
*Columbia University*

## Abstract

*Hydroponic is emerging as a new agricultural system to control the absolute terms of the plants environment. An automated system like that have all of the positive effects with no attention at all. There are many technical challenges in developing such a system both software and hardware wise. While working with many sensors and measurement units the emphasis should be on the conversion and normalizing the values to the same scale. In our case the Raspberry Pi board needed to maintain a continuous connection to both the server and DB which is restricting the availability of running over batteries power. The results has overcome our original goals of a remotely controlled hydroponic system with a convenient graphical user interface, notifications, history collection and at a low cost.*

## 1. Overview

### 1.1 Problem in a Nutshell

There's a global crisis of food supply and it is only getting worse. By the year of 2040 all of our food will run out and the food that will remain will be extremely expensive which will lead to starvation. This global problem also affects our food supply already. In order to get the food into the grocery stores and eventually to our plates farmers pick the fruits when they are still unripe which is bad for our health. Another critical issue is pesticides and the avoidance with chemicals.

In the recent years there is high awareness to healthy organic food without any chemicals. Many companies and farms supply groceries directly to the client removing the need to pick up premature vegetables and fruit and improving the produce quality. People has an environmental, health and public interest to grow their own food with home farms to save money and be sure they get a high quality organic vegetables. Another aspect is the hobby of gardening as a therapy. Gardening usually requires prior professional knowledge and can be complicated to learn especially with delicate vegetable families. By automating farming and hydroponics those domains can be more accessible and intuitive creating more hobbyists.

### 1.2 Prior Work

Throughout the research done prior to beginning this project, we saw that home automatic hydroponic systems are available for purchase. However, they can be pretty pricey, ranging from $250 to $4,000 dollars for a reasonable home system. Our goal was to lower this cost to provide this item to the average household owner, so they can be able to grow their own produce easily from their home.

Diego Domingue published an article relating to autonomous hydroponic systems and the PH levels needed to elevate lettuce production [7]. This article discusses the hydroponic farm advantages and how these types of farms have a huge role in the future. Domingue talks about the importance of gathering data from these farms to allow for the best environment for the lettuce plants. He demonstrates the fluctuation of the PH levels over time and how the plants' growth reacted to that. For our project, we used this knowledge gained by this work to build our system to measure the ph levels and be able to control the environment.

We came across Austin simonson, who created an instructables page for a small hydroponic farm with an arduino [8]. In this instructables he talks about the benefits of hydroponic farms and how the produce from them does not only taste better, but is also more nutritional. Other benefits mentioned in this article is that more produce can be grown in less space. This allows even people in big cities with limited space, such as New York City, to make use of a hydroponic farm to grow their plants and produce. Our motivation only spiked more after reading this article. Not only did he mention it was possible to build this system, but he mentioned how rewarding it was. We knew that we wanted to have more features than this type of system to complete our hydroponic system with all the information we gathered in our research.

## 2. System Overview

This section describes the system design of our intelligent control of the hydroponic farm. Specifically, we discuss our goals (§2.1), challenges (§2.2), hardware design (§2.3) and software design (§2.4).

## 2.1. Objectives

Our project aims to create an intelligent control of the hydroponic farm. Therefore, we need to meet three requirements. First, we need to have the ability to remotely monitor our plants' conditions. Second, we can remotely change the plants' conditions to optimal conditions since we want to build a farm that is hydroponic and assure complete control over growing conditions. Third, we need to make sure the farm operate correctly and automatically. If there is something happen, there should be some alerts to notify users to check the status of the growing farm.

## 2.2. Technical Challenges

To have better control over the environment of our farm, we need to put enough sensors. The more sensors we put, the more information we have. However, different sensors have different measurement unit and scaling. Besides, they may only support analog input or output which means we need to use another ADC. Therefore, how to handle these sensors well and collect accurate data will be one challenge.

Another challenge is at the backend side. To save and analyze all the data we gathered from the sensors, we need to set up a powerful backend. The backend should support REST API to communicate with our IOT devices. In the meantime, it can also process the sensor data to generate some useful observations for us to use. Therefore, we decide to use Django to build our backend though we are not familiar with it.

## 2.3 Hardware Design

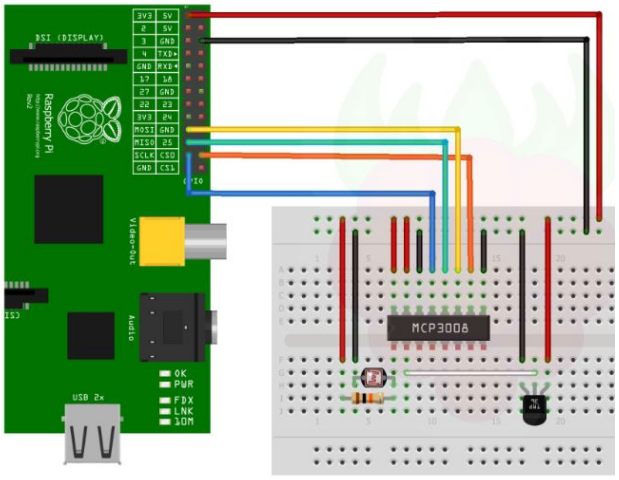


Figure 1: ADC connection to Raspberry Pi

### 2.3.1 Hardware Design

Hardware Parts:

- Analog Light Sensor - Adafruit GA1A12S202 Log-scale Analog Light Sensor
- Servo Motors x3 - FEETECH FS90R
- Temperature and Humidity Sensor - DHT22 temperature-humidity sensor
- PH Sensor - Gravity Analog pH Meter Kit
- ADC - Adafruit MCP3008 8-Channel 10-Bit ADC With SPI Interface for Raspberry Pi
- Webcam - Fosmon USB 6 LED 12.0 Megapixel USB PC Webcam Web Camera
- Raspberry Pi 3 Model B

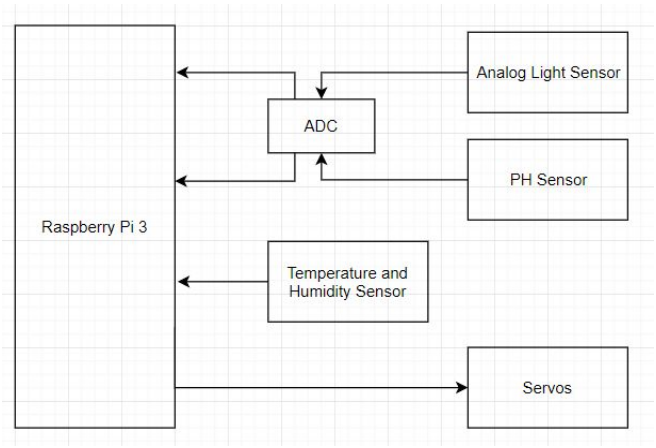


Figure 2: Hardware Information Flow

In Figure 2 the hardware information flow can be seen. Two of the sensor we used were analog. The Raspberry Pi 3 does not have any analog input pins. To solve this issue, we used an analog to digital converter (ADC), specifically the MCP3008. We connected the ADC to the Pi using the MOSI, SCO, MISO, and SCLK as shown in Figure 1. The digital outputs of the humidity and temperature sensor could be read straight by the Pi. The Pi was also connection by GPIOs to the servo motors (3), so that the Pi can control their movements.

### 2.3.2 Implementation

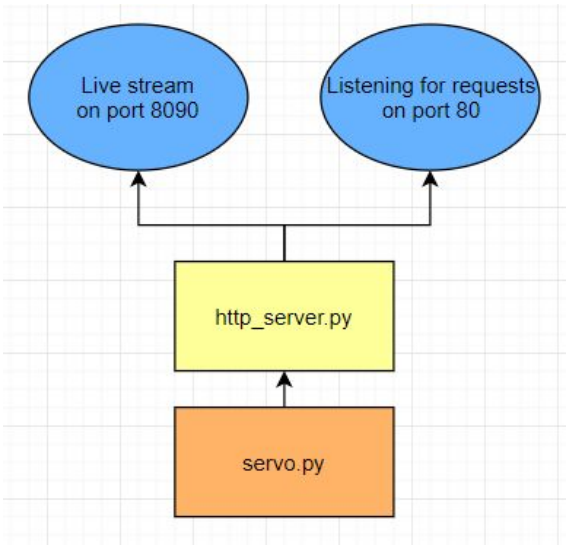


Figure 3: http_server flow

The http_server flow diagram (Figure 3) displays how the http_server works. http server sets up a web server on port 80 listening to requests. This script also uses localtunnel [9] to forward its' private ip address to a public address that is static. The way localtunnel works is that it will assign the Pi a unique publicly accessible url that will proxy all requests the Pi's locally running web server. The http server imports servo.py script to control the servo motors once a request is received (either waterup, phup, or phdown, discussed in the next section of this paper )

We use localtunnel to proxy two different ports of the Pi. As can be seen in figure 4, we are using the urls yuvaldoingiot.localtunnel.com and idodoingiot.localtunnel.com. These are using port 80 and 8089 respectively and forwarding their information to the correct address. This allows the Amazon Elastic Compute (EC2) instance to be able to communicate with our Raspberry Pi. By running this http server on the Pi, and proxying to a public address, we are able to listen and receive requests from sources outside the network trying to send messages to the Pi. This enables the system to communicate as a whole.

```
subprocess.Popen("lt --port 80 --subdomain
yuvaldoingiot", shell=True)

subprocess.Popen("lt --port 8090 --subdomain
idodoingiot", shell=True)
```

Figure 4: Overall software design

The script that gathers the sensor information and sends it to our EC2 instance to put in the database is the mainserver.py script. This piece of code imports multiple files, as seen in Figure 5, such as readPHandLight.py and AdafruitDHT.py (temperature and humidity) to gather all the different sensor information. After all the information from the four different sensors have been gathered, a post request is sent to the EC2 instance with a body of temperature, humidity, ph, and light. This information then gets a timestamp and is put into the SQLLite database running on the EC2 instance.

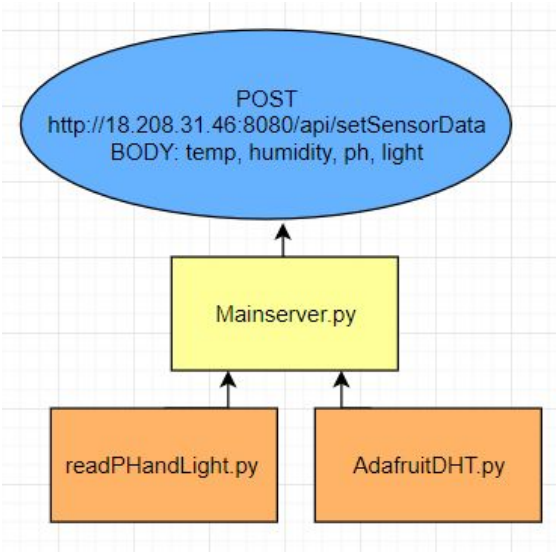


Figure 5: Mainserver flow

### 2.4.1 Software Design

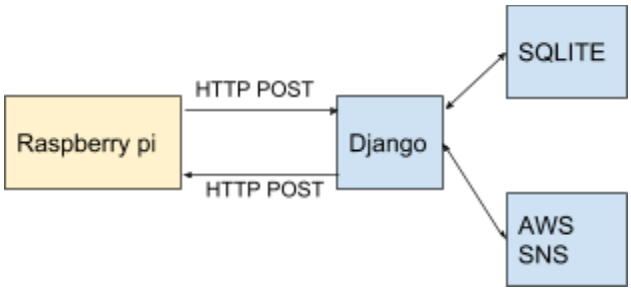


Figure 6: Overall software design

Figure 6 shows an overall software design. In our software system, we used Django to build our frontend and backend. SQLITE is the database that we saved anything we collected from the Raspberry Pi. Finally, we placed one HTTP server on the Raspberry Pi to receive any control messages that we wanted to send to the sensors.

After collecting data from the sensors, Raspberry Pi sent them to our Django backend by REST API. Our backend is provided by Django REST framework. The

framework provided some ways to build the HTTP GET and HTTP POST. After receiving the sensor data from Raspberry Pi, Django saved all of them into the SQLITE database. The accumulated sensor data that saved in the database provided us some useful information. First, if users want to know the trend of those sensor data or want to learn at what situation, the plants grow better, they can query the database. Second, Raspberry Pi sent sensor data to our database every minute. Therefore, if we can keep receiving sensor data, we can guarantee that the sensors worked very well. However, if we can not receive new sensor data in more than 10 minutes, we will assume that there is something wrong with the sensors. Then we will send a short message by AWS SNS to the user to ask he or she to check the sensors or all the environment. We also provided one service to let users control sensors remotely. After pressing any control button, Django will send an HTTP POST request to our Raspberry Pi. Raspberry Pi will forward the command to the sensors.

Figure 7 show the detailed control flows of the software design. Flow 1, 2 and 3 are provided by Django backend while flow 4, 5 and 6 are provided by the Raspberry Pi server.

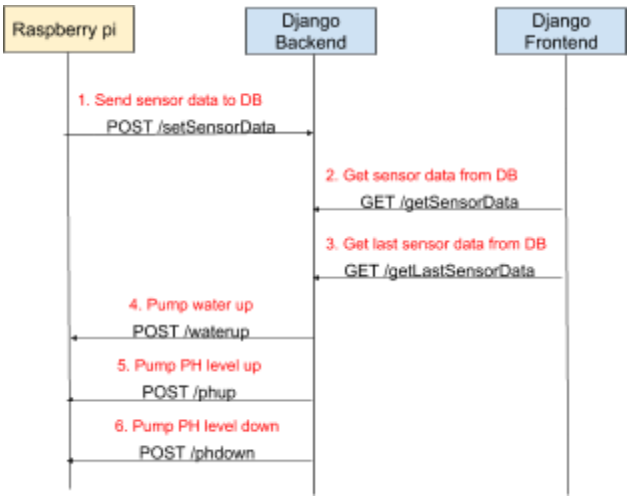


Figure 7: Control flow

### 2.4.4 Video Stream

The Streaming process included 2 components, collecting video feed from the webcam connected to the Raspberry Pi and a local server to present this stream. We also tried using a youtube live option as a direct stream but both collecting the video feed, encoding it to the correct format and uploading it to youtube on the same time was too heavy computationally for the Pi. In order to collect the feed we there was a couple of libraries we needed to install, Vlc which includes codecs within and previewing the collected video on the Pi. The second software was ffmpeg which includes ffserver inside (the server component) of the package. The server was configured on a local file called ffserver.conf which can be found in our main repository, [10]. We launched both the server and ffmpeg to collect the video feed using a shell script called webcam.sh, the server was routed to port 8090 and included a python mapping to a static IP to avoid changing the URL each time we launch the server.

## 3. Results

In this section, we demo an automated farm which has the following functions:

- Provide 4 kinds of sensor data:
  - Humidity
  - Temperature
  - PH Level
  - Lighting
- Sensor data can be uploaded to the backend.
- Users can watch the live streaming of the farm.
- Users can get the history of all the sensor data.
- Users can get current sensor status.
- Users can control PH and water level remotely.

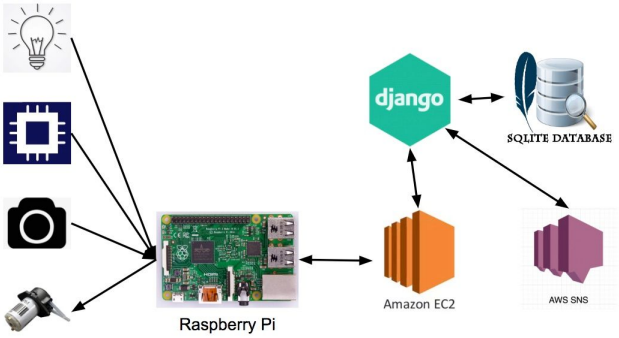


Figure 8: Overall system setup

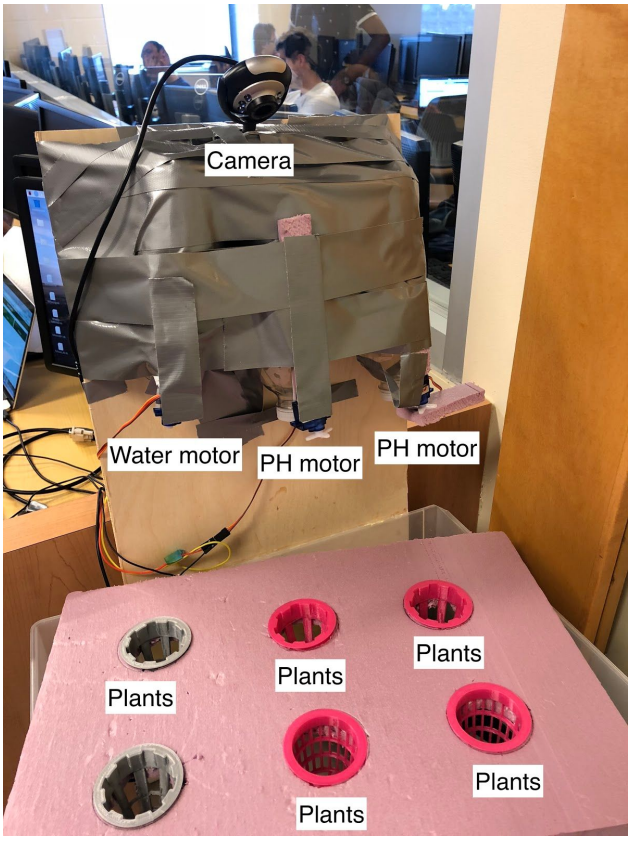


Figure 9: System setup

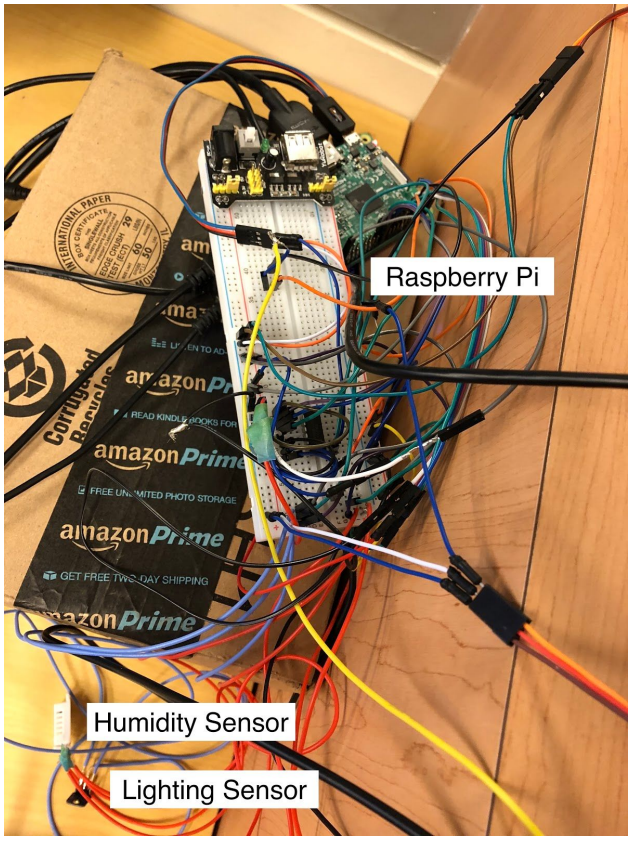


Figure 10: Raspberry Pi and sensors

### 3.1 Environment Setup

In our system. we use Raspberry Pi to interface with our 5 sensors, camera sensor, humidity sensor, temperature sensor, PH sensor and lighting sensor. We also set up an HTTP server on Raspberry Pi to control those sensors remotely. Our backend and frontend are located on AWS EC2. We use Ubuntu as our operating system. Figure 8 shows the block diagram of our system.

Figure 9 is our automated farm. There is a camera on the top and it can provide the live streaming video for users. In the middle are the motors which are used to pump water or PH to those plants.

Figure 10 shows our Raspberry Pi and related sensor controllers all connected using a breadboard and some jumper wires. Since the lighting sensor we used is analog, we needed to connect that sensor the a converter to retrieve the output of it on the Pi.

### 3.1 Frontend Results

Figure 11 is our home page. In the middle of home page presents a live streaming video, which can let users watch their plants anytime. In the bottom of home page shows the current environment status of our plants. Users can monitor their plants' environment by checking those status.

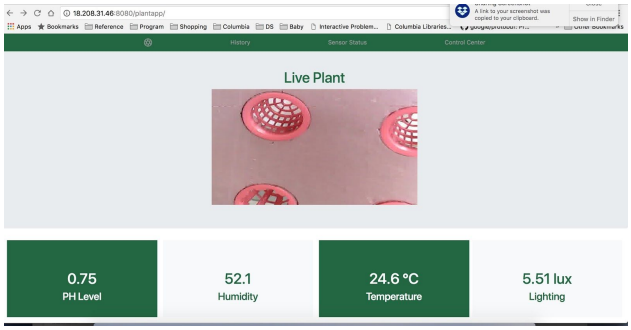


Figure 11: Home page

In figure 12 our history page of the monitoring system is displayed. Here we show the latest 50 sensor data. Users can have some idea of what the past one hour looks like.

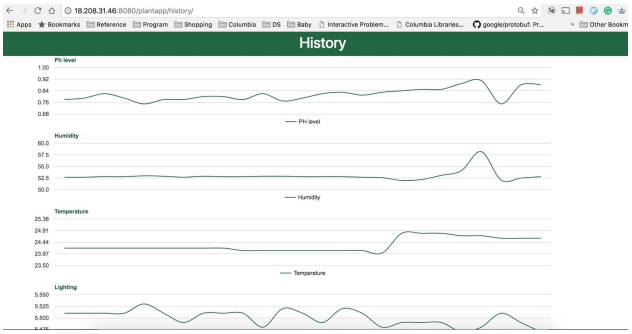


Figure 12: History page

Figure 13 is our status page. Here we show the current sensor status. Users can get the notification while the sensors are going wrong.

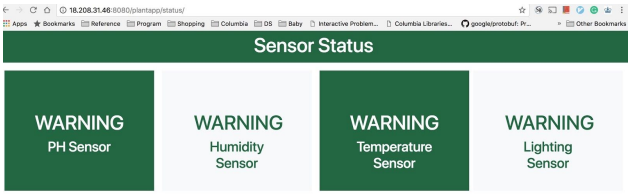


Figure 13: Sensor status page

Figure 14 is our remote control page. We provide three kinds of buttons, which are adding water, adding PH value and reducing PH level. By pressing the button, it can trigger the remote motors to add something to plants.

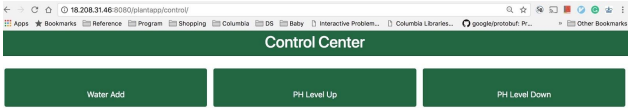


Figure 14: Controller page

Figure 15 demonstrates our REST API. We used JSON format as the data structure to communicate between the backend and users.

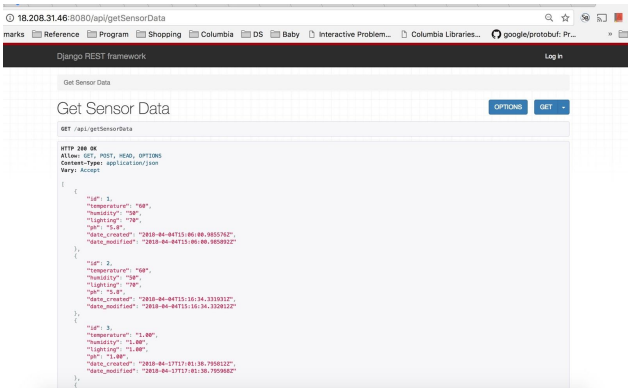


Figure 15: API page

## 4. Demonstration

This is a video that we demonstrate our overall system: https://youtu.be/75QBGPH4HLU

## 5. Discussion and Further Work

From the results we gathered, we saw that the system as a whole worked pretty well with minimum latency. Many improvements could be made to make the system more reliable as well as more fully automated to fit users' needs. We think that a system that automatically manages your plants for you based on the optimal environment preset or pre-learned from the plant is ideal. This way the user can have minimal effort in taking care of the plant and the system can do so on it's own, sending alerts to the user when any action needs to be taken.

This setup could also have been done differently using an arduino or another microcontroller. The advantages of using an arduino is to be able to read analog inputs without a converter. The Pi, as we used in this project, is very heavy and all of its abilities are not being used in the system. We think the cost of the system as a whole could be lowered with a lighter microcontroller that only provides the necessary functions outlined in this document.

An addition that could be made is to create the system as fully automatic with learned optimized environments, a further data collection has to be made with many plants growing in a hydroponic environment such as the system. After data has been gathered from those environments for multiple weeks, a machine learning (ML) algorithm can be applied. The ML algorithm will then train on the data given to it and will know the optimized environment for the plants in the system. With our automation set in place, the system can keep the plants in the hydroponic setup at exactly the correct environment to optimize the growth and quality of the plants.

## 6. Conclusion

Our intelligent control of hydroponic farm demonstrates an agricultural system that can control the absolute terms of the plants' environment. We monitor our plants' conditions by deploying the sensors and Raspberry pi around the plants. We set up one HTTP server and one backend. The HTTP server controls the plants' conditions and it can let us remotely change the plants' conditions to optimal conditions. The backend saves the sensor data to the database and analyzes it to generate some useful information for users to check. On one hand, it alarms users to check their farm when it notices something abnormal in the sensor data. On the other hand, it provides the logging history for users to do further analysis. In conclusion, we successfully design and implement a practical agricultural system at a low cost that can meet all the requirements we mentioned in this paper.

## 7. References

[1]https://www.sciencedirect.com/science/article/pii/S0168169912000361
[2]https://www.sciencedirect.com/science/article/pii/S0168169913000264
[3] https://cloudponics.com/
[4]http://www.instructables.com/id/Hyduino-Automated-Hydroponics-with-an-Arduino/
[5]https://youtu.be/75QBGPH4HLU
[6] Diego S.Domingues, and Hideaki W.Takahashi. "Automated System Developed to Control PH and Concentration of Nutrient Solution Evaluated in Hydroponic Lettuce Production." Computers and Electronics in Agriculture, Elsevier, 28 Mar. 2012, www.sciencedirect.com/science/article/pii/S0168169912000361
[7] austinsimonson. "Hyduino - Automated Hydroponics With an Arduino." Instructables.com, Instructables, 12 Oct. 2017, www.instructables.com/id/Hyduino-Automated-Hydroponics-with-an-Arduino/.
[8] https://localtunnel.github.io/www/

## A. Tools and Resources

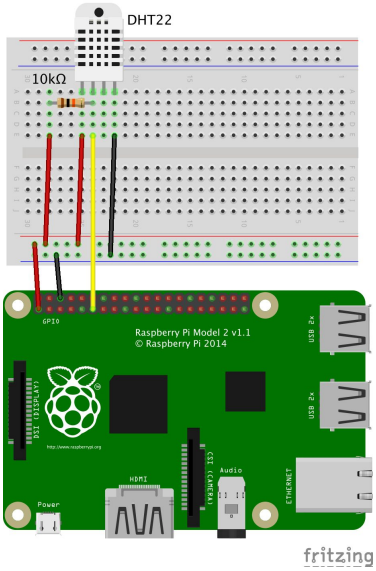


Figure 16:Temperature and humidity sensor diagram

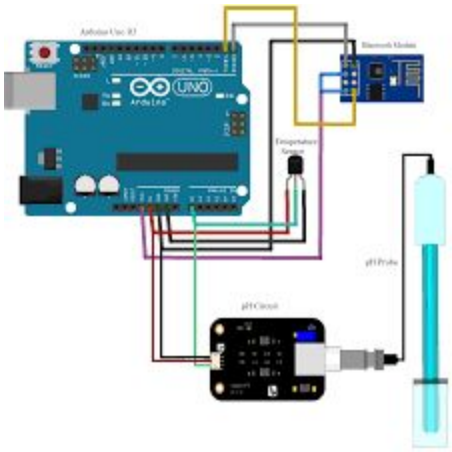

Figure 17: Ph Sensor diagram

## B. Github Repository

### B.1 IOT Device Codebase

This is the github for the source code running on the Pi: https://github.com/idomic/IoTFinalProject

### B.2 Frontend and Backend Codebase

We use Django to build our frontend and backend. We also use the Django REST framework to build the REST API. The detailed codebase is put in https://github.com/siaoting/django/tree/master/django-plant

The following codes are the structure of our database. Here we saved all kinds of data we collected from sensors.

```
In models.py
class SensorData(models.Model):
    temperature = models.CharField()
    humidity = models.CharField()
    lighting = models.CharField()
    ph = models.CharField()
    date_created = models.DateTimeField()
    date_modified = models.DateTimeField()
```

Figure 18: codes for controlling database

The following codes how we control the REST API to communicate with clients.

```
In views.py
//This is for POST /setSensorData
class SetSensorData(generics.ListCreateAPIView):
    serializer_class = SensorDataSerializer
    def get_queryset(self):
        return None

    def perform_create(self, serializer):
```

```
        serializer.save()

//This is for GET /getLastSensorData
class GetLastSensorData(generics.RetrieveAPIView):
    serializer_class = SensorDataSerializer
    def get_object(self):
        try:
            query = SensorData.objects.latest('id')
        except SensorData.DoesNotExist:
            query = ""
        return query

//This is for GET /getSensorData
class GetSensorData(generics.ListCreateAPIView):
    serializer_class = SensorDataSerializer
    def get_queryset(self):
        if 'size' not in self.request.query_params:
            queryset = SensorData.objects.all()
        else:
            size = int(self.request.query_params['size'])
            queryset =
SensorData.objects.all().order_by('-id')[:size]
        return queryset

In urls.py
urlpatterns = [
    path('getLastSensorData',
GetLastSensorData.as_view(), name =
'getLastSensorData'),
    path('getSensorData', GetSensorData.as_view(),
name = 'getSensorData'),
    path('setSensorData', SetSensorData.as_view(),
name = 'setSensorData'),
]
```

Figure 19: codes for REST API

The following codes are how we query the database and show the sensor data on our frontend.

```
In views.py
//This is for home page
def index(request):
    template =
loader.get_template('plantapp/index.html')
        data = SensorData.objects.latest('id')
        temp = data.temperature
        humidity = data.humidity
        lighting = data.lighting
    return HttpResponse(xxxx)

//This is for history page
def history(request):
    template =
loader.get_template('plantapp/history.html')
    last =
SensorData.objects.all().order_by('-date_created')[:50]
    phs = []
    phs.append(["Date", "PH level"])
    humids = []
    humids.append(["Date", "Humidity"])
    temperatures = []
    temperatures.append(["Date", "Temperature"])
    lights = []
    lights.append(["Date", "Lighting"])
    for v in last:
        date = v.date_created.strftime("%H:%M")
        phs.append([date, float(v.ph)])
        humids.append([date, float(v.humidity)])
        temperatures.append([date, float(v.temperature)])
        lights.append([date, float(v.lighting)])
    return HttpResponse(xxxx)

//This is for status page
def status(request):
    template =
loader.get_template('plantapp/status.html')
    status = "OFF"
    try:
        data = SensorData.objects.latest('id')
        print(data.date_created, datetime.now())
        last = data.date_created.timestamp()
        now = datetime.now().timestamp()
        diff = now - last
        if diff < (5 * 60):
            status = "ON"
        elif diff < (10 * 60):
            status = "WARNING"
    except SensorData.DoesNotExist:
        status = "OFF"
    if status != 'ON':
        plantapp.sns.send_message()
    return HttpResponse(xxxx)

//This is for control page
def control(request):
    template =
loader.get_template('plantapp/control.html')
    try:
        data = SensorData.objects.latest('id')
        temp = data.temperature #Todo: set to water
pressure
        humidity = data.humidity
        lighting = data.lighting
        ph = data.ph
    except SensorData.DoesNotExist:
        temp, humidity, lighting, ph = 0, 0, 0, 0
```

```
    return HttpResponse(template.render(context,
request))

In urls.py
urlpatterns = [
    path('', views.index, name='index'),
    path('history/', views.history, name='history'),
    path('status/', views.status, name='status'),
    path('control/', views.control, name='control'),
]
```

Figure 20: codes for frontend

## C. Detailed Organization of Codebase

### C.1 Frontend and Backend Organization

In our Django codebase, there are three sub-folders. One is plant folder which is used to maintain overall server environment settings. One is plantapp folder which is used to build our frontend. The last one is the api folder which is used to build the REST API and SQLITE database.

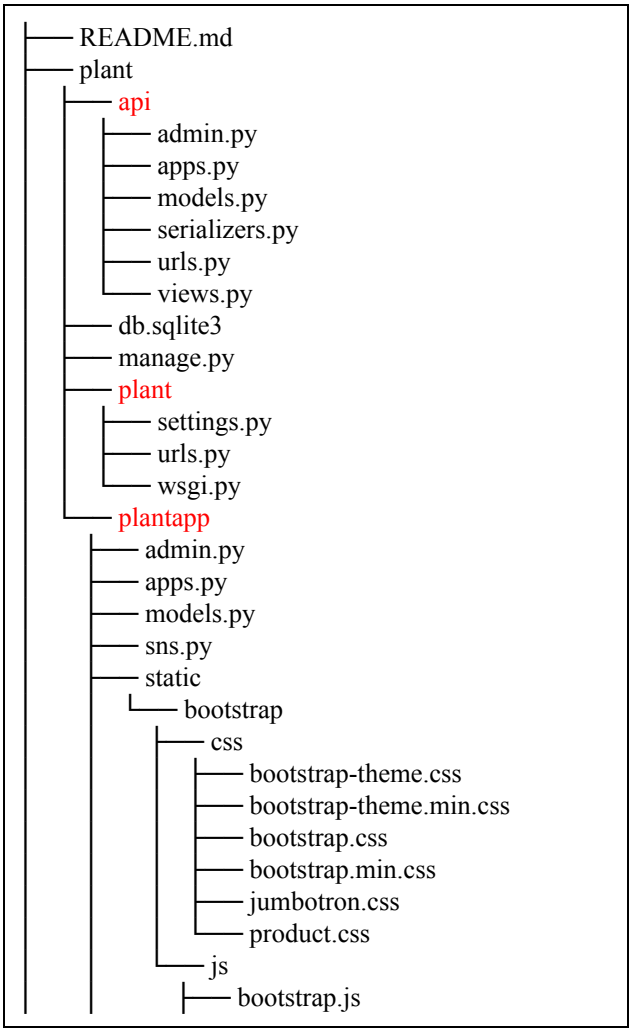


Figure 21: Frontend and backend organization