College of Engineering

COMP 491 – Computer Engineering Design Project Final Report

Zektor:

AI in Vertical Farming

Batuhan Arat, Volkan Işık

Project Advisor: Sajit Rao

Spring 2024

Table of Contents

1. Abstract

Vertical farming presents a sustainable solution for urban agriculture by growing plants indoors in vertically stacked layers, independent of weather conditions and soil quality. Despite its advantages, the high costs of implementation and operation remain significant challenges. This project, Zektor, aims to enhance vertical farming through automation of monitoring, focusing on environmental control and plant monitoring. We developed an automated monitoring system using IoT and image acquisition technology. Environmental data is collected via sensors and images of plants are captured using a robotic system with a Raspberry Pi. These images are analyzed by AI models to assess plant growth and health. The results are displayed on a user-friendly dashboard. Through stress testing, we demonstrated the system's reliability and efficiency. Future work will focus on improving image acquisition, expanding AI models, and enhancing the user interface to support scalable and comprehensive plant monitoring. This project lays the groundwork for fully automated, efficient, and sustainable vertical farming systems.

2. Introduction

Vertical farming is a promising method that allows plants to be grown indoors by stacking them vertically. Unlike conventional farming, vertical farming is not affected by weather conditions and does not require rich soil, making it suitable for urban environments [1]. Most vertical farming methods do not require any soil, enabling plant cultivation in cities and reducing transportation losses. Vertical farming also allows for higher plant density per unit area, which prevents deforestation for farmland creation. Additionally, many vertical farms use air filtering to prevent insect contamination, eliminating the need for pesticides.

Despite these advantages, vertical farming faces significant challenges, particularly high entry and running costs. Building the necessary structures, pumping mechanisms, and air filtration systems requires substantial initial investment. Operating costs are also high due to the need for LED lighting, water, and fertilizers. Furthermore, current systems are designed for human operation, necessitating clean room protocols and space for stairs, which reduces the efficiency of space utilization.

We believe that automation is key to realizing the full potential of vertical farming. By eliminating human factors, we can reduce costs and increase productivity. Automated systems, designed for robotic operation, will only require water, electricity, and fertilizers, without the need for human operators. This approach will enable efficient food production in diverse environments, such as urban rooftops, deserts, and shipping containers. Mass production of these systems could address food sustainability challenges, even in regions unsuitable for traditional farming or lacking qualified labor.

To achieve our goals, we have developed an automated monitoring system for vertical farming. Effective system control requires monitoring environmental factors and assessing each plant's condition. Our system's hardware consists of two components: IoT and image acquisition. The IoT component collects environmental data via sensors and sends it to a database using microcontrollers. The image acquisition component involves a robotic system with a Raspberry Pi and camera that captures plant images and transmits them to the database.

We have trained 2 AI models using machine learning algorithms and a lettuce image database to detect plant growth and health status. As discussed by Zhang et al. [8], we have used data augmentation techniques to enhance our image dataset and train our algorithm. The model results, along with sensor data, displayed on a dashboard that we have developed.

This project aims to address the initial step of vertical farming automation, focusing on system control and plant monitoring to pave the way for fully automated, efficient, and sustainable vertical farming systems.

Alternative designs for addressing the monitoring problem heavily depend on IoT technologies [2]. These designs use sensors to monitor Co2 levels, light intensity, electrical conductivity (EC), pH and more to assess plant status. However, relying solely on sensor data does not provide the full picture. While both approaches can work together, sensor data alone cannot capture the detailed visual cues that indicate a plant's condition. Additionally, seeing real images of the plants enhances the user’s sense of control and provides a more comprehensive understanding of the entire system.

Another design approach involves use of drones. Drones circulate around the facility and capture plant images, which are then analyzed using machine learning algorithms to extract information about plant status. Our main difference comes from the image acquisition method. Using drones is harder to control, harder to do location tracking and more prone to errors due to stability issues. In contrast, our solution is easier to control and more stable because our hardware operates on pre-built structures in vertical farming facilities. Additionally, our technology stack makes our approach more cost-effective.

3. System Design

Our system consists of an image gatherer system, IoT, two machine learning models, databases, front-end and required backend software. The image gatherer system collects images of each plant and sends them to the database. Images are forwarded to machine learning models. Sensor data is gathered through IoT. The results of models, images and sensor data are shown in the UI with sensor values collected from IoT.

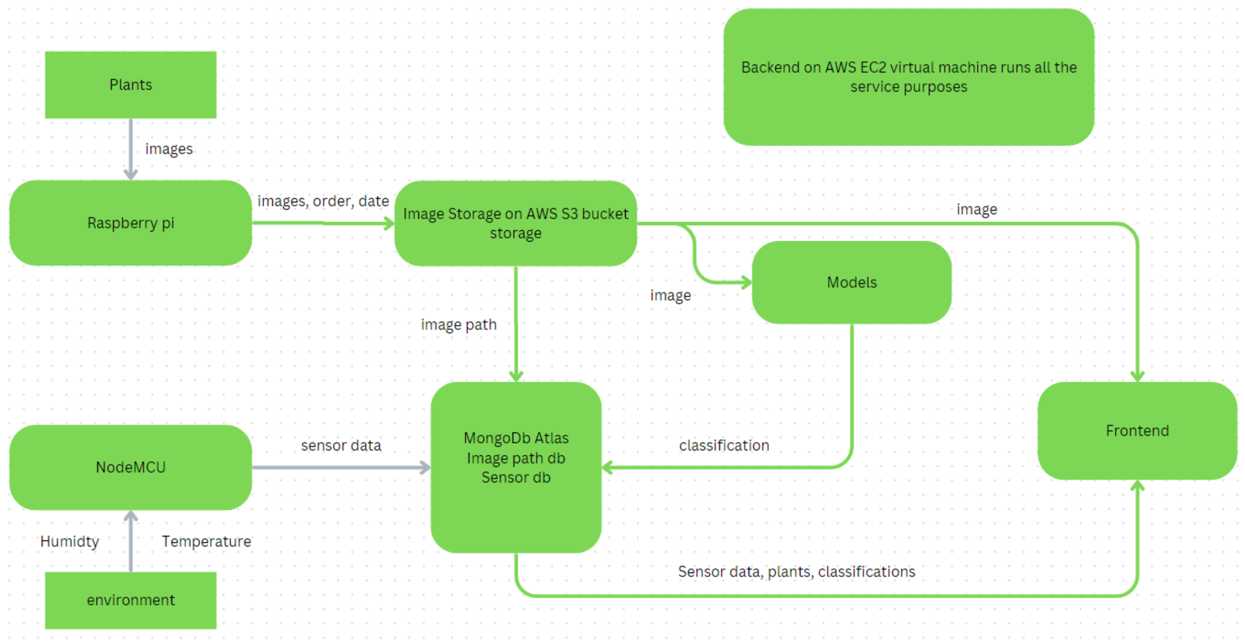

Backend on AWS EC2 virtual machine runs all the service purposes
Plants
images
Raspberry pi
images, order, date
Image Storage on AWS S3 bucket storage
image
Models
image
image path
MongoDb Atlas Image path db Sensor db
classification
Frontend
NodeMCU
sensor data
Humidty
Temperature
environment
Sensor data, plants, classifications

Figure1: System Diagram

a. Image Gatherer System Design

The image gatherer system aims to collect images of each plant. The system is controlled by a Raspberry Pi 4. Raspberry Pi is mounted on a rail with belts controlled with a motor unit. Raspberry Pi is responsible for collecting images, sending those images to the server, and giving required order to the motor unit. We are running a python code on Raspberry Pi that is responsible for the tasks.

The motor unit consists of a nema 17 step motor and a NodeMCU microcontroller. NodeMCU is connected to the internet and listens for orders from Raspberry Pi. When an order is received it controls the step motor. Step motor provides the required movements by turning the belts connected to the Raspberry Pi.

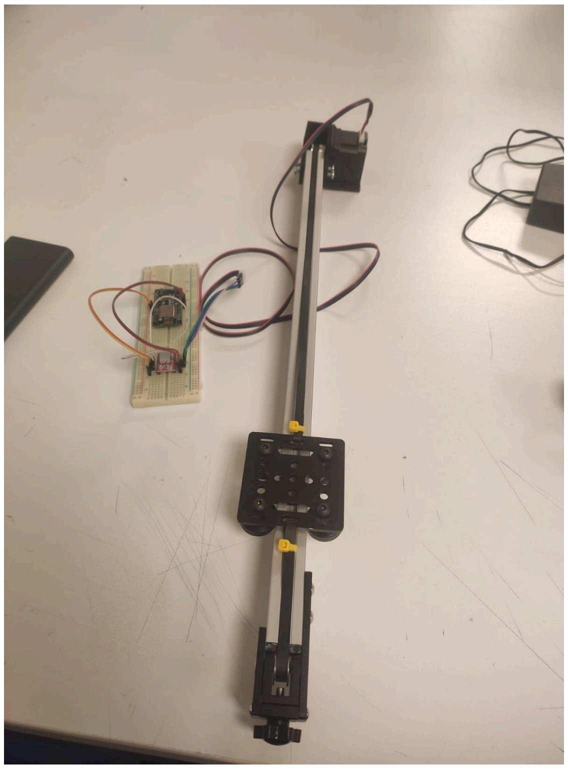


Figure 2: Controller and Step Motor

To assemble the system, we needed to design some parts with Fusion 360 and 3D printed them. We designed a part to assemble the step motor into the sigma profile we use as the rail. We designed a case to mount the Raspberry Pi and power bank into the car used on the rail and mount the camera on Raspberry Pi.

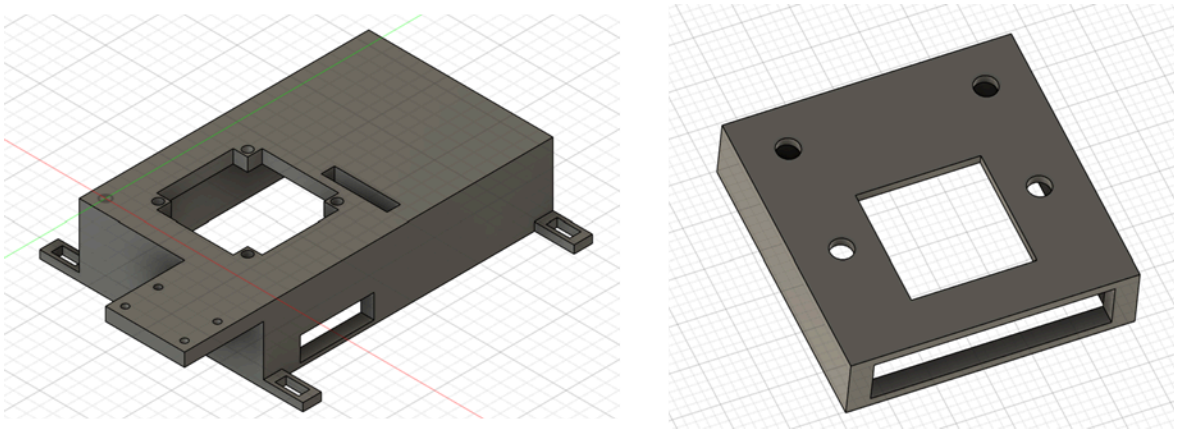


Figure 3: Fusion 360 Design of Hardware Case

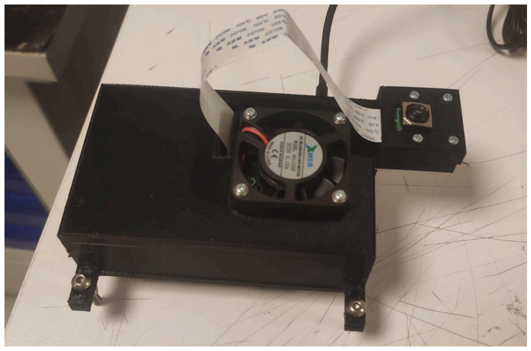


Figure 4: Raspberry Pi Mounted on Hardware Case

b. Machine Learning System Design

In our proposal, we decided to develop only a model to guess the growth phases of lettuce plants. With feedback from our advisor Sajit Rao, we decided to also add another model for lettuce health detection.

The first model is responsible for guessing the growth phases of lettuce plants. For that purpose, we found a dataset [3] that contains images of lettuces in different development phases. We manually categorized those images into 4 groups. After dividing the dataset for validation and test, we augmented those images to ~10500 images. We trained a classification model based on those data. After our research, we decided to try two different models of architecture. We trained a U-Net model [4], which was not very successful. We believe that this is due to the size of our dataset. Then, we trained a ConvNet [5], which was more appropriate for our dataset.

The second model is responsible for guessing the health status of lettuce plants. For that purpose, we searched online for a public dataset, but we could not find a useful one. We decided to gather our own database for that. For that purpose, we rotted lettuces and collected a dataset with different sized lettuce in different stages of decay. We collected 350 images. After dividing the dataset for validation and test, we augmented those images to ~6000 images. We trained a ConvNet that clusters the images into two classes, health and unhealthy.

c. Demo Structure System Design

Initially, we did not plan to develop a hydroponics system since we wanted to focus on the image gatherer technology. With feedback from Sajit Rao, we decided to assemble a demo system to work on. We built a small hydroponics system with metal profiles as frames. We installed a tray with a water circulation mechanism. We also attached the image gatherer system and sensors on this platform.

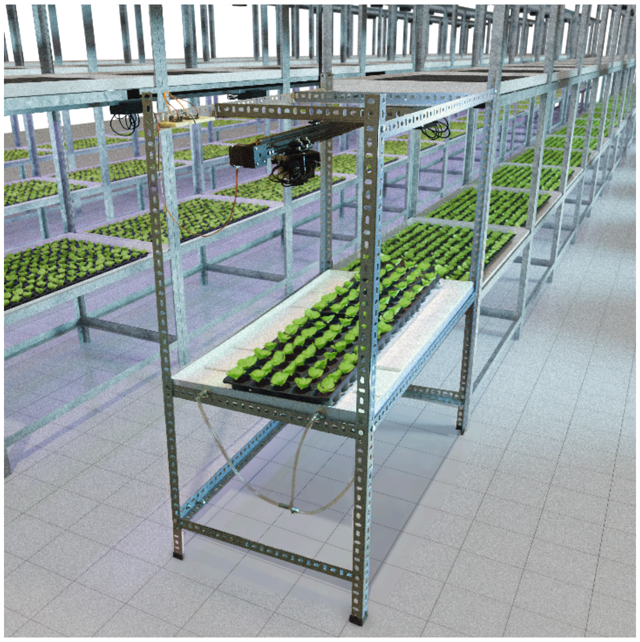


Figure 5: Photoshopped Image of Our Structure in Vertical Farming Facility

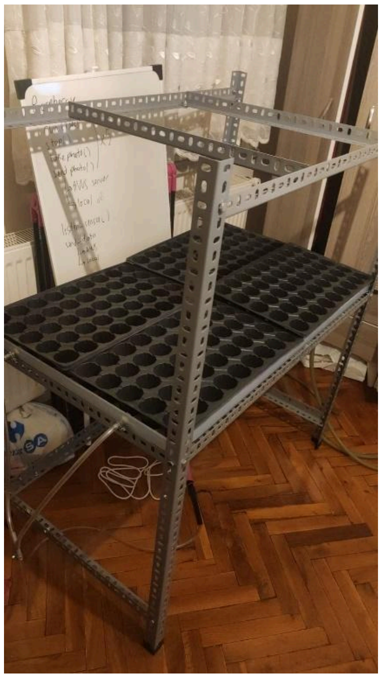


Figure 6: Demo System Without Plants

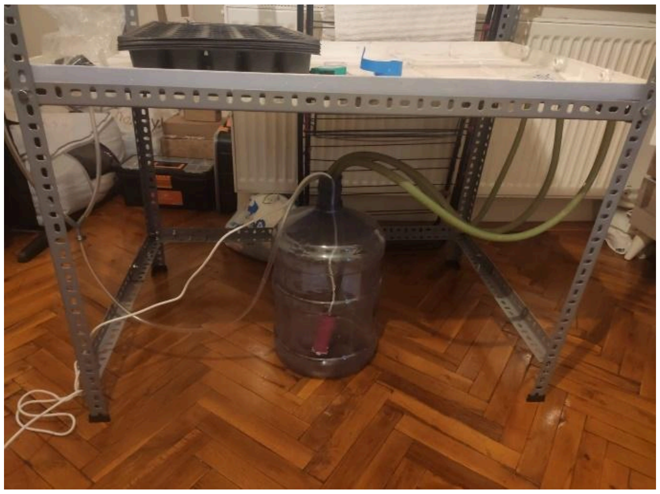


Figure 7: Demo System With Water Pump

d. IoT System Design

We developed an IoT setup to gather sensor data. Initially we planned to include humidity, temperature, light intensity and CO2 levels. After discussing with our advisor and with each other, we decided that the real value of the project comes from the AI and image gatherer part. So, we decided to gather only humidity and temperature data. We shifted our focus from here to other parts.

e. Frontend System Design

Frontend system is written in React.js and styling is done with CSS. User can log in with “userId” and create a “userId” for the demonstration purposes. User interface composed of 3 main sections. Current humidity and temperature indicator section, plants section and information table about development phases with harvest estimation. In the plants section, we show each plant with the last image, health information, order information and development phase information. We color coded the health status as red for unhealthy, green for healthy plants. When you click on a specific plant, you can see all of the images of that plant with phase and health information in chronological order. We provided a user friendly dashboard that user can see the whole story of each plant. Frontend design is inspired from [6].

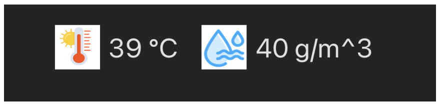
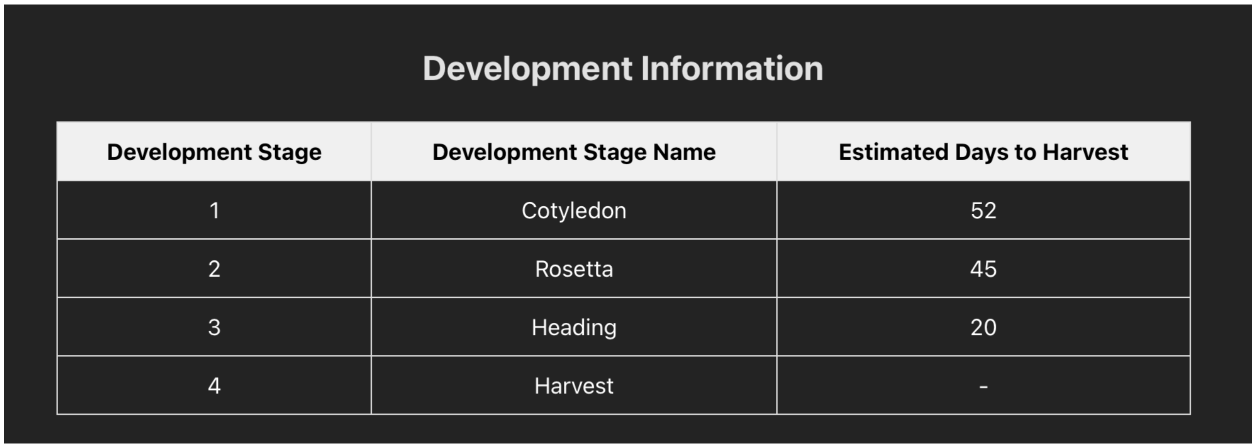

39 °C 40 g/m^3

Figure 8: Sensor Section


Development Information

Development Stage	Development Stage Name	Estimated Days to Harvest
1	Cotyledon	52
2	Rosetta	45
3	Heading	20
4	Harvest	-

Figure 9: Development Stage Information Section

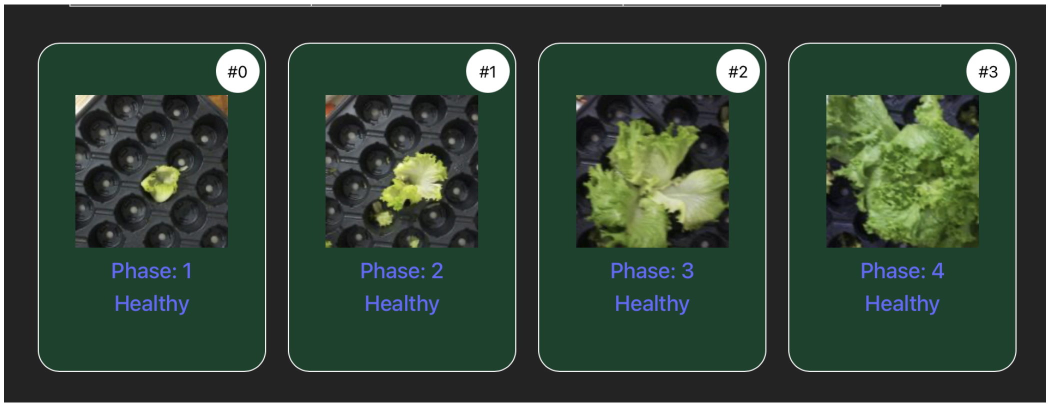


Figure 10: Plant Section

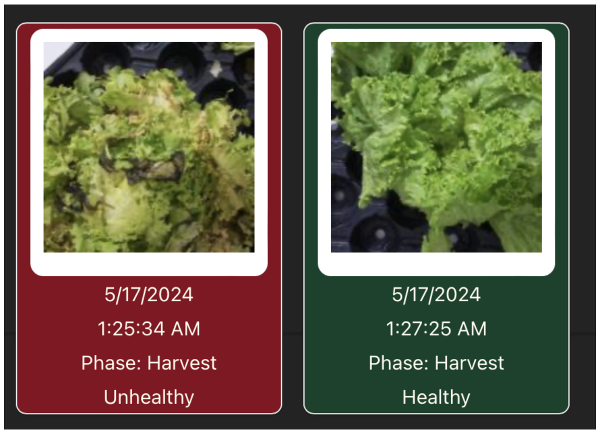
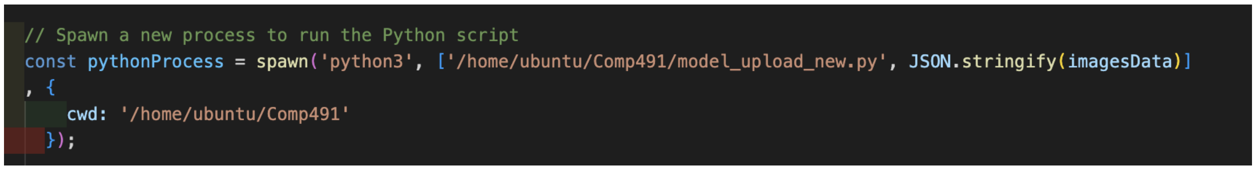


Figure 11: Small screenshot from plant's detail section

f. Backend System Design

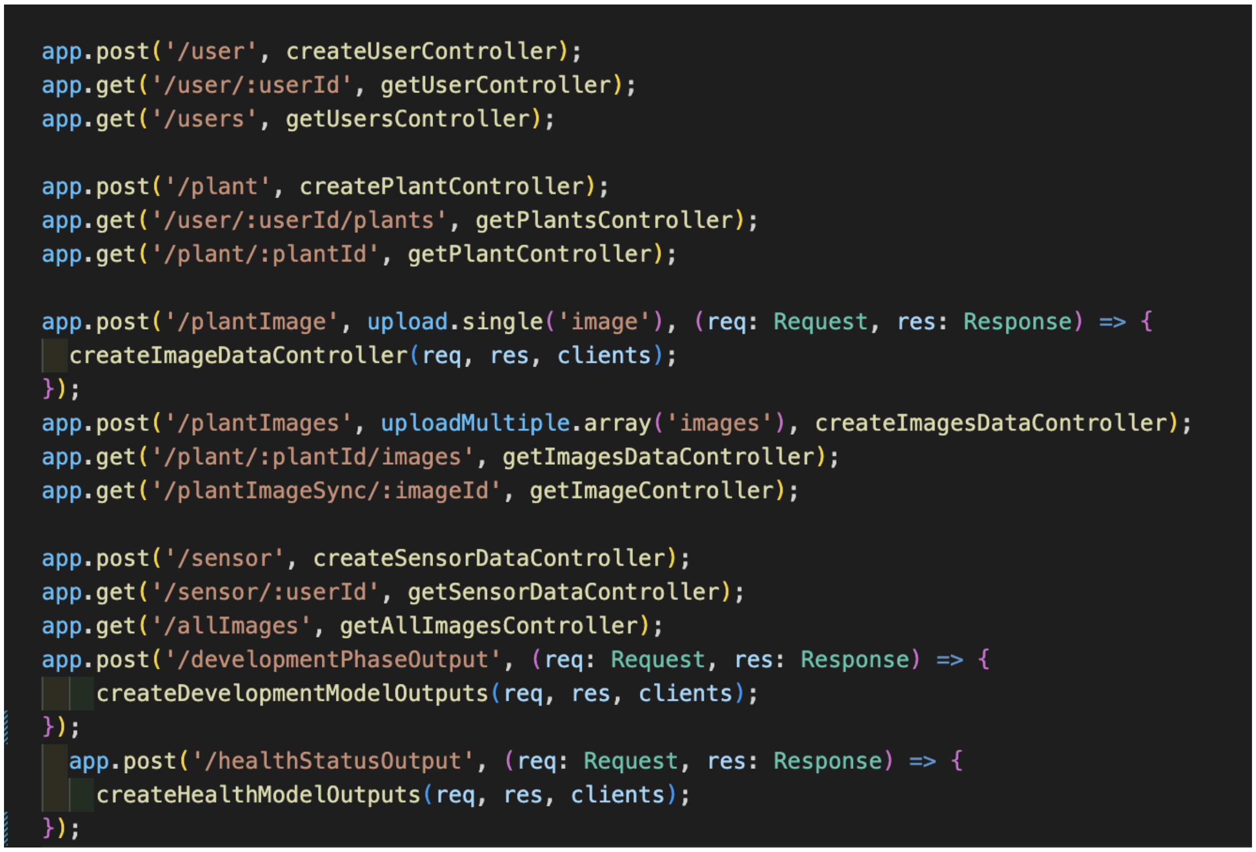
Our backend system is written in Express.js. Api endpoints are divided into two parts such as communication between hardware and machine learning models and interaction between user interface and database. Model and backend is deployed on the Amazon EC2 server. In order to demonstrate real time data flow, we used web sockets to show images at the user interface while the camera is moving. When our hardware completes its tour, it sends all of the images at once to the backend. Backend processes those images and forwards them to the model by running a python script. Since models and backend are deployed on the same virtual machine, we think that this is the easiest solution to do. When model decides on the labels, it uses sockets to update the user interface.



```
// Spawn a new process to run the Python script
const pythonProcess = spawn('python3', ['/home/ubuntu/Comp491/model_upload_new.py', JSON.stringify(imagesData)]
, {
    cwd: '/home/ubuntu/Comp491'
  });
```

Figure 12: Backend code that forward images and runs the model

Endpoints related to the communication between user interface and database are designed with Restful API structure. All of the endpoints are given below.



```
app.post('/user', createUserController);
app.get('/user/:userId', getUserController);
app.get('/users', getUsersController);

app.post('/plant', createPlantController);
app.get('/user/:userId/plants', getPlantsController);
app.get('/plant/:plantId', getPlantController);

app.post('/plantImage', upload.single('image'), (req: Request, res: Response) => {
  createImageDataController(req, res, clients);
});
app.post('/plantImages', uploadMultiple.array('images'), createImagesDataController);
app.get('/plant/:plantId/images', getImagesDataController);
app.get('/plantImageSync/:imageId', getImageController);

app.post('/sensor', createSensorDataController);
app.get('/sensor/:userId', getSensorDataController);
app.get('/allImages', getAllImagesController);
app.post('/developmentPhaseOutput', (req: Request, res: Response) => {
    createDevelopmentModelOutputs(req, res, clients);
});
  app.post('/healthStatusOutput', (req: Request, res: Response) => {
    createHealthModelOutputs(req, res, clients);
});
```

Figure 13: Endpoints

g. Database System Design

We implement a database on MongoDB Atlas that successfully stores the data of the users. In the users table we only hold userID and id of the plants that user has. We did not implement authentication methods such as email and password registration because userId's are specific to the user itself. It is provided by us manually. UserId is embedded in the hardware.

Image itself is not stored on the MongoDB Atlas, they are stored in the Amazon S3 Bucket. Storing images on MongoDB Atlas requires base64 decoding of the image data which can result in possible loss in original image base to the conversions. More importantly, the base64 image holds a substantial amount of memory in the image database. Each image data also needs to hold the development phase, health status, date and id of the plant to which the image belongs. If the image itself takes large memory, scalability of the imageData table is affected. Queries not related to the image itself (such as developmentPhase information of the imageData) slows at speed due to the bad design. Therefore, instead of storing the image itself as base 64, we store just the location of the image. Image itself is stored in the Amazon S3 bucket, the url of the S3 object is stored in the MongoDB.

Plants table hold userId, type of the plant, order which corresponds to the location of the plant in the system, development phase, health status and array of the images' id. As we discussed,
imageData also holds development phase and health status information, plant itself holds the development phase and health status information of the last image it is taken.

Since we are downgrading the importance of the sensor values, we delete the Co2 level and light intensity fields from the sensor table. We are just measuring and storing the temperature in celsius, and humidity in g/m^3 which represents the data from the whole environment, not specific to the plant.

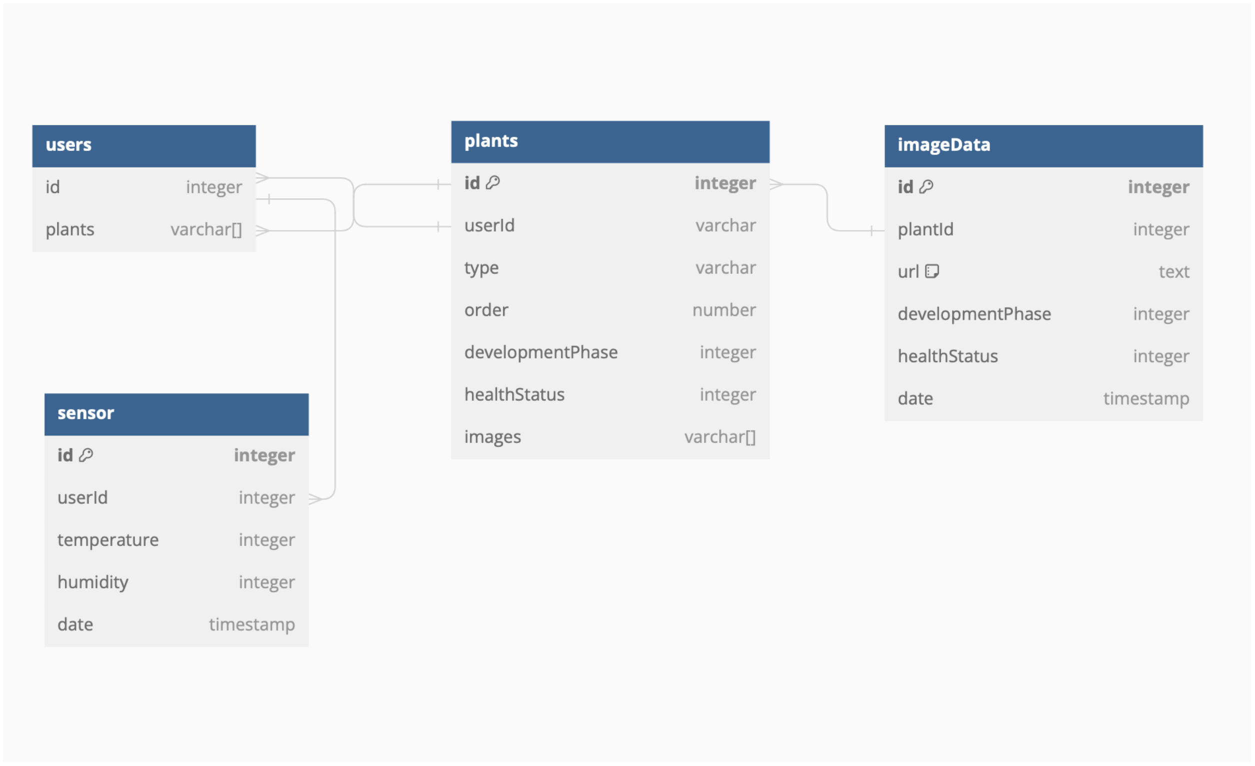


Figure 14: Database Design[8]

4. Analysis and Results

a. Stress Testing

We were able to successfully assemble all systems discussed above. To ensure that our system works well at each level, we have conducted ~25 hours of stress testing on the system. Except for some minor issues, which we solved, our system worked well on stress testing. We did not face any problems on mechanical parts, on image gathering system, on server or on UI.

b. Results of the Model

In the growth phase guessing model, we achieved a 0.85 test accuracy. One of the most important metrics for the success of the model was the confusion matrix. We see that almost all errors were on the neighboring classes, which also shows the success of the model. There would be a problem if it was guessing class 1 (cotty) lettuce as class 4 (ready to harvest).

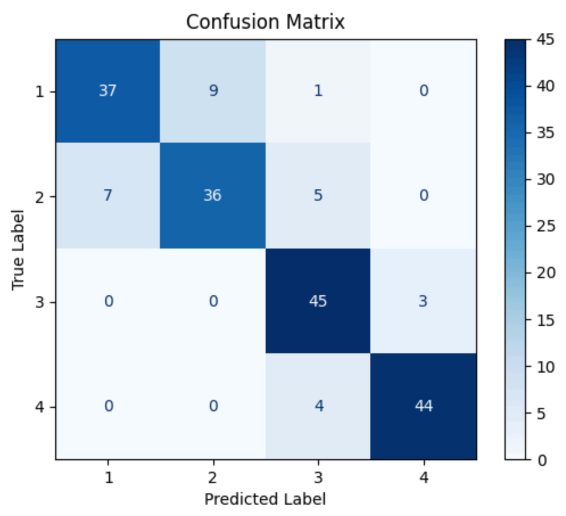


Figure 15: Confusion Matrix of Development Phase Model

In the health detection model, we achieved a 0.86 test accuracy.

5. Conclusion

We have successfully implemented hardware that can gather sensor data from the environment and image gatherer system that captures images of each plant. We have also developed an infrastructure that forwards this data to our machine learning models and frontend application, which evaluates the data to determine the plant's health status, estimated harvest time, and development phase information. Results are displayed on the user-friendly interface.

a. Future Works

i. Image Gatherer System

Our system collects images of plants on a straight line, on only one dimension. The current system is not able to collect images of lettuces, spread on two dimensions. We are planning to develop image processing techniques to divide each taken image into different frames, each containing a plant on the tray. So, we will be able to collect images of lettuce spread on two dimensions with the same movement of the camera.

This system is developed to be used in vertical farming facilities. It will increase the installation costs if we use a robot for each vertically stacked layer. We need to design the mechanics so that our image gatherer system is able to move through each vertically stacked layer.

ii. Developing the Model

We need to collect more data and develop our model. We need to gather data with different environmental conditions like light intensity, different backgrounds to increase accuracy on any installed system. Our system only works for lettuce plants. If we have data for other plants, we can expand our system to those plants, too.

Our health detection algorithm gives healthy or unhealthy as output. It would be better if our system gives a numeric value on the health of that plant because there are degrees of problems on plants. Also, we do not have any idea about the reason for the problem itself in this algorithm. It would be better if our algorithm was also able to tell the problem in the plant. For instance, it would be better if it gives an output as “bacterial infection, 60% health”.

iii. Developing the User Interface

We are planning to enhance the user interface design on our frontend. Currently, the main plant section is designed for a comfortable interface displaying 5-10 plant data points for demo purposes. However this can be scaled to accommodate thousands of plants with an adjustable main plant section. Users will be able to customize the sections according to the architecture of their facilities. Each section could display 50-100 plants and be organized by plant type or other criteria, allowing for a more scalable user interface.

Additionally, when the user clicks a specific plant, the plant page is displayed with all images of that plant captured over the entire timeline. This can be enhanced with filtering options, enabling users to perform more detailed analyses. For example, users will be able to filter images by specific dates, such as the last week or the last month.

Moreover, we aim to add a comprehensive plant list section where users can filter plants based on type, health status, and development phase. This will provide a more organized and efficient way to manage and monitor large numbers of plants, ensuring that the user interface remains intuitive and scalable as the system grows.

6. References

1. https://doi.org/10.15623/ijret.2013.0203013
2. https://doi.org/10.1109/KST.2018.8426141
3. https://doi.org/10.5281/zenodo.7433286
4. https://doi.org/10.3389/fpls.2022.980581
5. https://doi.org/10.3390/horticulturae8121124
6. https://github.com/codyseibert/flashcardsage
7. https://dbdiagram.io/
8. https://doi.org/10.1038/s41438-020-00345-6

7. Appendix

I. https://github.com/volkanisk/raspberry.git
    A. The github link for the codes run in raspberry pi.
II. https://github.com/batuhanarat/zektor.co
    A. Client folder is for frontend code.
    B. Server folder is for backend code.
III. https://github.com/batuhanarat/Comp491
    A. The github link for model related code.