The American University in Cairo
School of Sciences and Engineering

plantiful
AgriTech Solutions

CSCE 4981: SENIOR PROJECT II

An Intelligent Monitoring System for Precision Agriculture

Project Report

Written By

Farah Seifeldin - 900160195
Lobna ElHawary - 900160270
Maram Abbas - 900153570
Mariam ElZiady - 900161869
Samer Basta - 900150910

Supervised By

Dr. Mohamed Shalan
Dr. Tamer ElBatt

May 22, 2021

Table of Contents

Introduction **3**

Project Objectives **3**

Project Importance **3**

Project Implementation **4**

- Remote-Sensing 5
  - Implementation 5
  - Testing 9
    - DHT11 Sensor 9
    - pH Sensor 11
    - Soil Moisture Sensor 11
    - Server Testing 12
- Remote-Capturing 13
  - Implementation 13
  - Testing 14
- Plant Analyzer 14
  - Dataset Collection 14
  - Health Model 15
    - Implementation and Testing 15
      - Segmenter 15
      - Classifier 17
        - Trainer/Tester 17
        - Datasets 18
        - Dataset Processing 18
        - Models 19
        - Saving 21
      - Predictor 21
  - Growth Stage Model 21
    - Implementation and Testing 21
    - Step 1: Classifying Fruiting vs. Non-Fruiting 21
    - Step 2: Classifying Initial vs. Flowering 23
    - Testing of Two-Step Algorithm 24
    - Results 25
  - Deployment 25

Web Application 26
Implementation 26
Backend 26
Frontend 28
Testing 28
Results **28**
Deployment Plan **30**
Conclusion **31**
Appendix: Glossary **32**

Introduction

Throughout our senior project, we have implemented a system that monitors greenhouses in terms of sensor levels and intelligent diagnostics. In this report, we will be outlining the objectives and importance of our project. We will also explain technical details of our implementation and testing so that future engineers who will access our project can deploy, maintain, or optimize it. Finally, we will go over our results and deployment plan followed by concluding remarks. All of these main sections are separated by lines for ease of reading, and all terms are defined in the glossary appendix at the end of the report.

Project Objectives

Our project aims to ensure the health of agricultural crops whether in greenhouses or farms. To do so, we planned to report the most crucial plant parameters to crop owners. These parameters were obtained through our discussions with CARES, our collaborators for the project, and include the following: pH, soil moisture, temperature, humidity, leaf health, and growth stage. Our objective was to collect the first four parameters by IoT then determine leaf health and growth stage by AI.

Project Importance

The importance of the project lies in its contribution to the field of precision agriculture which replaces traditional agriculture. The reason why traditional agriculture has become inefficient is that it is remarkably human-dependent and thus error-prone and costly. If they seek a fully healthy crop, farmers and researchers have to examine one plant at a time almost every day, not just by sight but by hand-held sensors. Also, the hassle of storing and organizing all the sensor

readings and seen diagnostics (health and growth stage) is significant for any large crop owner.

Also, the project encourages more precision agriculture research and implementation, specifically in local communities. Many of the organizations in the country have requirements that are not fulfilled by products in the market, and published research about those requirements can help manufacturers provide products that can satisfy the organizations.

Finally, the project contributes to the technological industry by providing models, codes, datasets, and ideas that could be inspiration in the fields of IoT, AI, and web development.

Project Implementation

Our system is divided into three main microservices: the remote-sensing service which includes remote-capturing, the plant analyzer service, and the web application service.

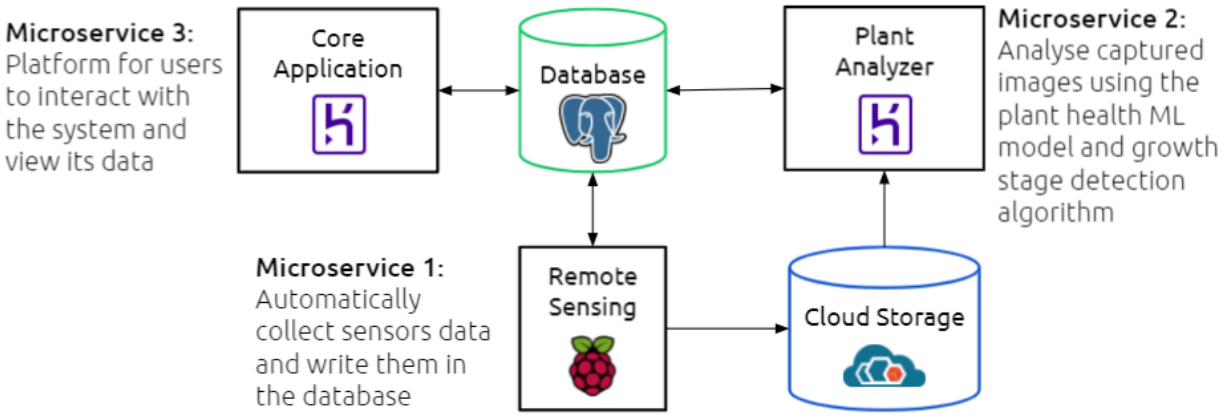

Fig. 1: System Overview

The remote-sensing system is the body that reads and stores sensor readings. The remote-capturing system is responsible for automatically operating a streaming

camera across the plants then storing the snapshots to be used by the plant analyzer system. The plant analyzer uses the stored snapshots to report the plants' health status and growth stage. Finally, the web application provides users with user-friendly data visualization and alerts as well as a way to interface with the hardware system.

Remote-Sensing

Implementation

The remote-sensing system is responsible for getting sensor data from the sensor nodes and adding them to the database. A sensor node contains an ESP32, a temperature sensor, a humidity sensor, a soil moisture sensor, and a pH sensor. After the sensor node collects the readings, it sends it to the Raspberry Pi, which has a server running on it in order to add the data to the database. This happens by the following procedure:

1. Once the sensor node is first connected to power, it turns itself to an access point that you can connect to from any WiFi device. In the examples below we used an iPhone. The ESP32 is the one responsible for creating the access point. The access point name is divided into two parts. The first part is SBAP which stands for *S*ensor *B*lock *A*ccess *P*oint. The second part is the MAC address of the ESP32 included in that sensor node. Fig. 2 shows the access point being recommended in the networks list that the device can connect to.

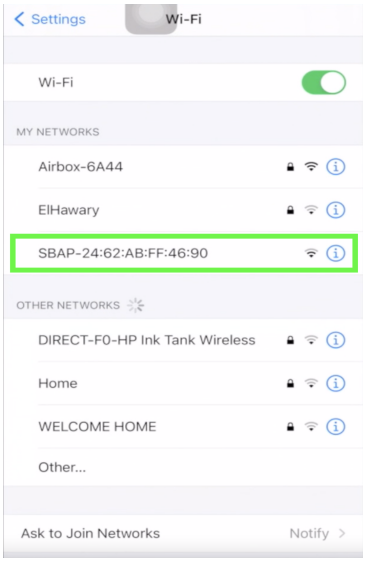

Fig. 2: Access Point Recommendation

2. The sensor node has a web server running on it; hence, once the connection to the access point is successful, access to its web server will be granted as shown in Fig. 2. The web server helps in controlling some configurations of that sensor node. These configurations include choosing an existing Wi-Fi network to connect to, choosing to let the sensor node to send data or not, and changing the interval time of which the sensors collect their readings and send them to the Raspberry Pi.

   The web server is hosted on the ESP32 on port 80. When the “OK” button is pressed in the configuration page, a get request is sent. Parameters are then collected and reflected on the variables so that the configurations take place.

Fig. 3: Initial Sensor Block Configuration

3. After adding all the configurations and connecting to the WiFi network chosen, the access point will turn off automatically. Revisiting the web server can be done by accessing the IP address of the sensor node in the new network as shown in Fig. 3.

Fig. 4: Recurring Sensor Block Configuration

4. Once all configurations are set, the sensor node will start to send data to the Raspberry Pi in the specified interval (if the user chose to send data in the configuration page).

   The sensor node connections are shown in Fig. 5. Sensors are connected to the ESP32 through its GPIO pins. All components are powered through rechargeable batteries. These batteries can be recharged/changed whenever needed. It is advised to use batteries with a high capacity. All sensors along with the ESP32 except for the pH sensor needs 3.3V. The pH sensor needs 5V. ESP32 is usually put in low power mode when waiting for the interval time to finish in order to save power as much as possible.

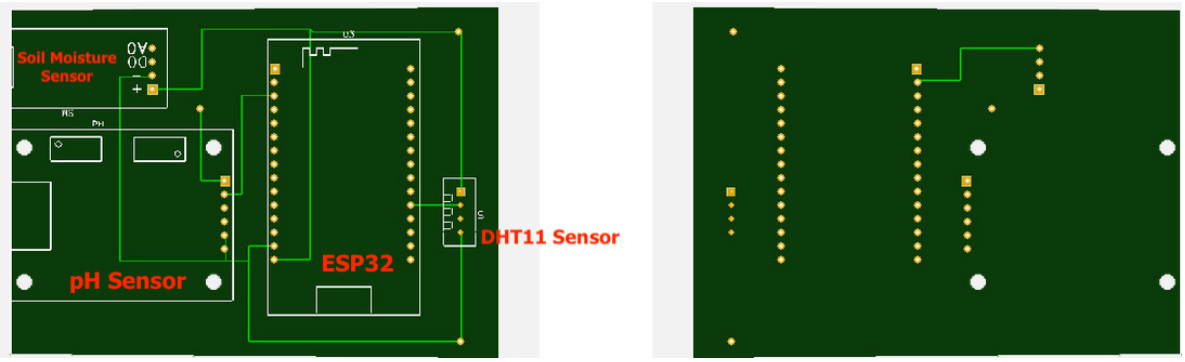

Fig. 5: Sensor Node PCB Layout

5. In order for the Raspberry Pi to differentiate between the data coming from different sensor nodes, i.e. knowing the data received is from which sensor node, the sensor node does not only send the data, it also sends the MAC address of the ESP32 included in it.

6. The Raspberry Pi receives the sensor data and checks whether the data is in the expected range or not. If not, a notification is sent to all users that have access to the project that contains that specific sensor node.

7. After checking on the readings, the readings are then added to the database in order for it to be read later on from the web application.

Testing

DHT11 Sensor

The DHT11 sensor did not need any prior calibrations; it was powered and connected to the ESP32 directly. The tables below show some tests done in order to make sure that the readings that came out of the DHT11 sensor were correct. When testing the temperature, the tests were done in a closed controlled room by an air conditioner. Testing the humidity was done inside CARES' greenhouse since the humidity was already controlled there.

Test Number	Air Conditioner Reading (℃)	DHT11 Reading (℃)
Test 1	18	17.95
Test 2	19	19.30
Test 3	19	19.26
Test 4	20	19.88
Test 5	20	20.16
Test 6	20	19.92
Test 7	21	21.10
Test 8	21	21.22
Test 9	22	21.94
Test 10	23	22.98

Table 1: Temperature Tests

Test Number	Greenhouse Reading (%)	DHT11 Reading (%)
Test 1	22	22.29
Test 2	23	22.86
Test 3	24	23.83
Test 4	24	24.22
Test 5	25	25.17
Test 6	25	25.11
Test 7	26	25.91
Test 8	27	27.06
Test 9	28	27.80
Test 10	29	29.18

Table 2: Humidity Tests

pH Sensor

To calibrate the pH sensor, we had to use a voltmeter. Voltage has to be 2.5V when pin p0 and the BNC connector are in short circuit for it to give correct readings. To test if the readings were correct, we used different solutions that have different pH levels. The table below shows the tests' results.

Test Number	Solution pH	pH Sensor Reading
Test 1	3	3.29
Test 2	3	3.11
Test 3	3	2.89
Test 4	4	4.20
Test 5	4	4.17
Test 6	7	6.88
Test 7	7	6.92
Test 8	9	9.14
Test 9	9	9.21
Test 10	11	11.04

Table 3: pH Tests

Soil Moisture Sensor

Calibrating the soil moisture sensor was necessary since we had to define being put in a super dry soil as 0% moisture while being put in water as 100% moisture. The tests in the table below were done by putting the soil moisture sensor in different soils that have different moisture levels. We divided the levels between 1 - 10 in

which 1 means that the soil was VERY dry and 10 means that the soil was VERY moist.

Test Number	Soil Moisture Level	Soil Moisture Sensor Reading (%)
Test 1	1	1.29%
Test 2	2	13.93%
Test 3	3	19.77%
Test 4	4	39.59%
Test 5	5	48.02%
Test 6	6	55.62%
Test 7	7	68.23%
Test 8	8	77.91%
Test 9	9	89.49%
Test 10	10	96.10%

Table 4: Soil Moisture Tests

Server Testing

To test the server running on the RPi, we made sure that the readings that were read from the sensors were actually the same ones that reached the server without any corruption. We also made sure that the data was being uploaded correctly to the database without any failures. In conclusion, the sensor readings that were sent to the ESP32 were the same ones found in the database after this whole process is done. At least 50 tests were done and the expected results were found.

Remote-Capturing

Implementation

For the remote-capturing system, the EZVIZ C6CN IP camera is used. The camera is mounted on top of the DAGU that was pre-programmed by us using an STM Nucleo K303F8 microcontroller to move the length of a row in the CARES greenhouse then turn 180° and repeat the same motion. It is programmed to pause after every second of motion for 30 seconds, so the RPi-hosted script can safely capture a snapshot of the plants.

The EZVIZ mobile application is used to configure the IP camera and connect it to the designated WiFi network. The camera is powered by a micro USB 2m cable that is plugged into an outlet. The STM microcontroller on the DAGU is similarly powered but using a 1m cable. Ideally, longer cables will be used to facilitate the complete motion.

The script on the RPi accesses the camera's network stream using the RTSP protocol through a function provided by the openCV library. The IP address of the camera in the network is necessary to do that. Then, two things happen. First, the captured image is automatically uploaded to Heroku's Cloud Cube storage which gives us access to 5GB of storage on an AWS S3 bucket. Second, after the image is uploaded successfully, an entry composed of the group_id associated with this camera_id, the image path in the S3 bucket, a flag to indicate if it has been processed by the plant analyzer, and finally a timestamp of when it was created, is added to the image table in the database. The plant analyzer, explained next, will use the image table to retrieve unprocessed images and process them.

Testing

To test the remote-capturing system, we first tested the camera with the image capturing and uploading script and the DAGU separately. For the DAGU, we tested its motion multiple times by altering code parameters until we achieved our desired target. Then, the camper-DAGU setup was tested in the field three times to ensure that our mechanical setup was durable and that the camera stayed still on top of the DAGU. In these trials, we tested the synchronization of the DAGU and the RPi capturing script and modified some parameters accordingly. Finally, we tested our setup in the greenhouse a final time before recording the demo.

Plant Analyzer

Dataset Collection

To supplement the data for our health model and to provide the entire data for our growth stage model, we carried out manual data collection at the CARES greenhouse. We used a Sony Exmor HD camera which we mounted on a tripod. As an initial fault, we took the images while hanging white background behind the plants. This was to remove noise from the plants behind and the greenhouse walls. However, this only made the training and testing images far from the actual real-time captured images. Nonetheless, we recognized our erroneous procedure later and took no-background images, which still made their way into the two models and helped save them.

We had to ensure that all images from the same growth stage had the same scale so that the model would not, for example, predict a fruiting plant as fruiting because its cucumber fruits look too small in the image when much larger and

fruiting-qualifying in reality. The exact photo-taking parameters for each stage are listed in Table 5.

Stage Name	Tripod Position	Tripod Height	Camera Angle*
Initial	45 cm away from plants	60 cm	10°
Flowering	45 cm away from plants	60 cm	10°
Fruiting	90 cm away from plants	60 cm	10°

Table 5: Dataset Collection Parameters

* The camera angle is measured counterclockwise from the positive x-axis to the line perpendicular to the camera's lens when placed at the origin.

Health Model

Implementation and Testing

The health model is deployed as part of the plant analyzer and consists of two actors: the segmenter and the classifier.

Segmenter

Because the classifier described below depends on leaf-focused images, segmentation[1] was required for all images the camera takes, which span whole

[1] Although segmentation is correctly defined as the extraction of the exact outlines of objects, not just their bounding boxes like detection, we are using the term segmentation since detection is popularly known (and misinterpreted) as the reporting of presence/absence of an object.

plants. The segmenter aims to find most leaves in a camera-captured image. The classifier will then run on the saved (leaf-focused) images, not the original plant-spanning image. The segmenter first finds all valid contours in the image. Valid contours are, in our case, medium-sized dark green contours. The contours were found through HSV contouring in the OpenCV framework and were optimized for each single image. The optimization was by trying many hue sensitivities to find the one yielding the most valid contours. After all the valid contours are found, rectangles are made from their leftmost top corner to their rightmost bottom corner. Finally, the segmenter then crops those rectangles out of the image like shown in Fig. 6.

Regarding the rationale for the valid contours, we specified dark green contours because those are the ones away from the light source and thus the most bottom ones. Bottom leaves are most of interest because they are the oldest in plants and accordingly the most likely to develop diseases. As for the medium-sized specification, it is because green contours that were too small did not have any semantic significance because they were not indicative of any leaves, or at least halves of leaves, and green contours that were too large spanned multiple leaves, violating the objective of the segmenter.

Fig. 6: Segmentation Result. Each colored rectangle in the original image (left) maps to an output image (right) with the same color border.

Classifier

The classifier is the body that classifies plant leaves into healthy or unhealthy. The classifier does not report the cause of disease, as the experts from CARES, representing the needs of researchers, were only interested in the binary classification of healthy vs. unhealthy. Also, we removed the disease distinction because our only cucumber (our testing plant at CARES) dataset was binary-labeled (healthy vs. unhealthy). Another reasoning was to avoid the logistical complication of having the CARES experts label each image in our manually collected dataset.The classifier is divided into two consequent actors: the trainer/tester and the predictor.

Trainer/Tester

The trainer/tester part of the classifier is what creates a model instance and uses large data to actually train and test that model instance. Finally, it saves the model

instance after training and testing to be used by the predictor. We will outline the trainer/tester's used datasets, processing, models, and saving method.

Datasets

Name	Ours	Kafrelsheikh University (KU)	PlantVillage Kaggle (PV-K)	PlantVillage TensorFlow (PV-TF) *Augmented*
Labeling	Binary	Binary	Disease	Disease
Label source	CARES	Uploader	Uploader	Uploader
Image location	CARES greenhouse #2	Unknown	Unknown	Unknown
Image form	Leaf-focused	Leaf-focused	Leaf-focused	Leaf-focused
Size	173 images	691 images	20,639 images	60,343 images
Plants	Cucumber	Cucumber	Bell pepper Tomato Potato	Apple Bell pepper Blueberry Cherry Corn Grape Orange Peach Potato Soybean Strawberry Squash Raspberry Tomato

Table 6: Health Dataset

Dataset Processing

PV-K and PV-TF were reorganized to match our binary description by placing all healthy classes (labelled by plant_healthy) together and all unhealthy classes (labelled by plant_disease) together. Fig. 7 shows the process. All images in all

sub-datasets were normalized[2] to make sure all features (image pixels) have a fair chance of determining the classification output.

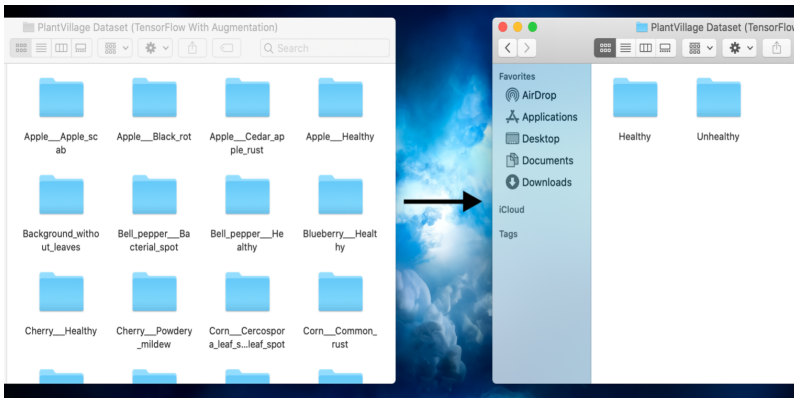

Fig. 7: Health Dataset Reorganization

Models

We implemented various trials for the classifier to reach our top accuracy. The trials included various machine learning models used with various data. The data used for each trial was a result of forming a dataset from at least two of our sub-datasets. The dataset formation process goes as follows: 80% of each involved sub-dataset are randomly selected to form the sub-dataset's training section while the remaining 20% form its testing section. After that is done for all of the involved sub-datasets, all training sections are appended together to form the model's training dataset, and all testing sections are appended together to form the model's testing section. We call this type of data formation "representative" because all sub-datasets are sufficiently represented in both the training and testing datasets. Fig. 8 shows the testing accuracies for all trials, and table 7 shows the full description of each trial.

[2] Data normalization: the process of dividing by the effective range of inputs to equalize the effect of differently ranged inputs on the output.

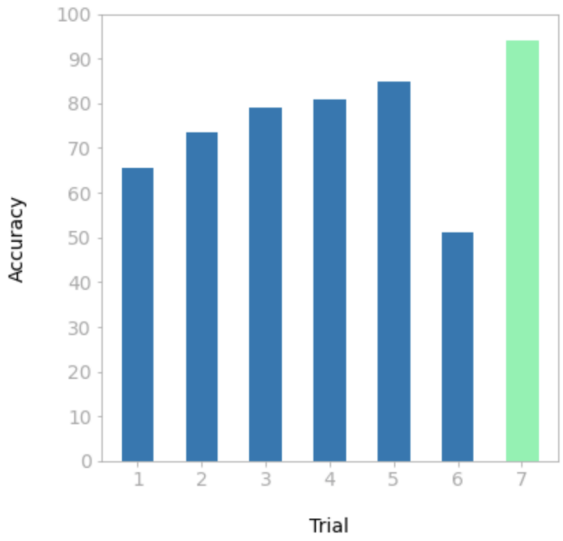

Fig. 8: Health Model - Testing Accuracy Vs. Trial Graph

Trial number	1	2	3	4	5	6	7
Datasets Used	Ours KU PV-K	Ours KU PV-K PV-TF	Ours KU	Ours KU	Ours KU	Ours KU	Ours KU PV-K
Total Size	21,503	81,846	864	864	864	864	21,503
Dataset Formation	Represent-ative	Represent-ative	Represent-ative	Represent-ative	Represent-ative	Represent-ative	Represent-ative
Training Percentage	80%	80%	80%	80%	80%	80%	80%
Input Size	150x150x3	150x150x3	150x150x3	150x150x3	150x150x3	150x150x3	150x150x3
Model Name	VGG16	VGG16	SVM	SVM	SVM	SVM	SVM
Model Parameters	Adam optimizer Softmax loss 10 epochs Batch of 32	Adam optimizer Softmax loss 10 epochs Batch of 32	Linear kernel	Poly-nomial kernel	RBF kernel	Sigmoid kernel	RBF kernel

Table 7: Health Trials Descriptions

The model we have selected for deployment belongs to trial #7 since it achieved the highest **testing accuracy (94%)**. It is worthy to note that, for 15 plants, trial #2 fared relatively well, giving an accuracy of 73.4%.

Saving

After it was trained and tested, the selected model was saved in a .sav file.

Predictor

The predictor is the body that loads the saved model (result of the trainer/tester) and reports the decision for a single image prediction.

Growth Stage Model

Implementation and Testing

The algorithm explained below is used to predict the current growth stage of cucumber plants from images. The growth stages it predicts are initial, flowering, and fruiting. The initial stage is characterized by small leaves and short plants while the flowering has larger leaves and 1-3 yellow flowers per plant. Finally, the fruiting stage is characterized by the presence of cucumber fruit and many large leaves. To make the prediction problem simpler, we decided to break down the 3-class classification problem into two 2-class classifications.

Step 1: Classifying Fruiting vs. Non-Fruiting

The first step of the algorithm predicts Fruitings vs. Non-fruiting images. To do that, we rely on the presence of fruit i.e. cucumbers in the image. To detect these cucumbers, a cascade classifier was trained on negative samples (not containing the fruit) and 1000 samples of cucumber fruit features (shown in fig. 9) extracted manually from the growth stages dataset we collected. After experimenting with

different parameters for the cascade classifier, we decided on using 13 stages with a 20 x 20 pixels window of comparison and a minimum hit rate of 0.99.

Fig. 9: Samples of Fruit Features

Fig. 10: Detected Fruit Features in Images

After running the classifier on multiple fruiting and non-fruiting images, it was clear that some false positives were detected, but nevertheless, the number of detected fruit features in fruiting images far outweighed that in non-fruiting images. This helped us come up with a threshold of 8 detected fruit features in an image such that an image containing features less than that threshold is classified as non-fruiting. If it has more features than the threshold value, then it is classified as fruiting. This is illustrated in figure 10 above. The bottom two images belong to non-fruiting stages and few false positives were detected in them. On the other

hand, the top two images are in the fruiting stage and evidently, all cucumber fruits were detected in them. The proposed threshold-based algorithm was tested on 1266 images divided into 33.8% fruiting stage images and 66.2% non-fruiting stage images (initial and flowering). The results are highlighted in the table below.

Accuracy	Precision	Recall	F1 Score
93.5%	88.8%	92.5%	90.6%

Table 8: Step 1 Growth Stage Classification Results

Step 2: Classifying Initial vs. Flowering

To differentiate between the initial and flowering stages, multiple features were used:

1. Total green area for the 3 largest contours
2. Total yellow area for the 3 largest contours
3. Number of green contours
4. Number of yellow contours
5. Number of flowers using the number of yellow contours with area between 500 and 3000.

These features were decided to reflect the differences between initial and flowering stages, which are leaf size and number and number of flowers. A feature vector was extracted from 50 initial stage images and 50 flowering stage images. Logistic regression was then applied to these feature vectors to predict the probability of an input image falling between these two classes.

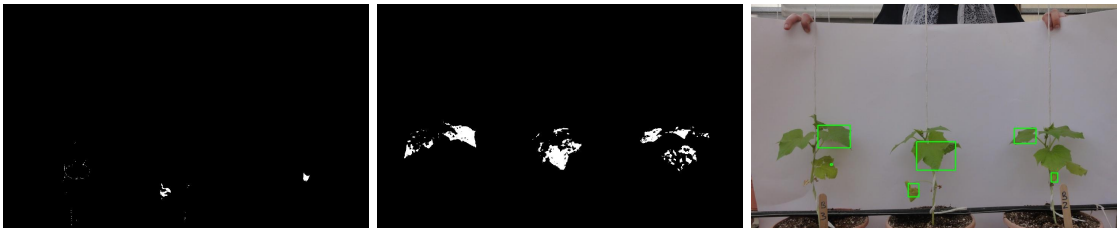

Fig. 11: Initial stage masks of yellow and green contours respectively

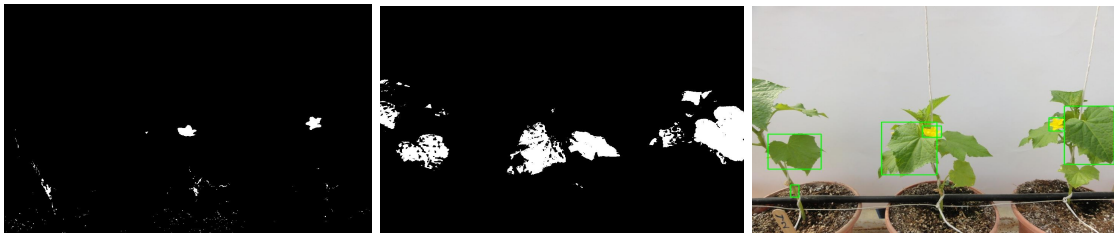

Fig. 12: Flowering stage masks of yellow and green contours respectively

The resulting logistic regression model was tested on 738 images composed of 57.7% initial stage images and 42.3% flowering stage images, and the results are summarized in the table below.

Accuracy	Precision	Recall	F1 Score
94.9%	99.6%	88.1%	93.5%

Table 9: Step 2 Growth Stage Classification Results

Testing of Two-Step Algorithm

Now that both steps of the algorithm have achieved satisfactory results, we tested the full algorithm on our dataset which was made of 1166 images divided according to the pie chart below. To test the images, a text file was created with the path of the images and their labels. The images were fed into the algorithm and the predicted result was compared with the label.

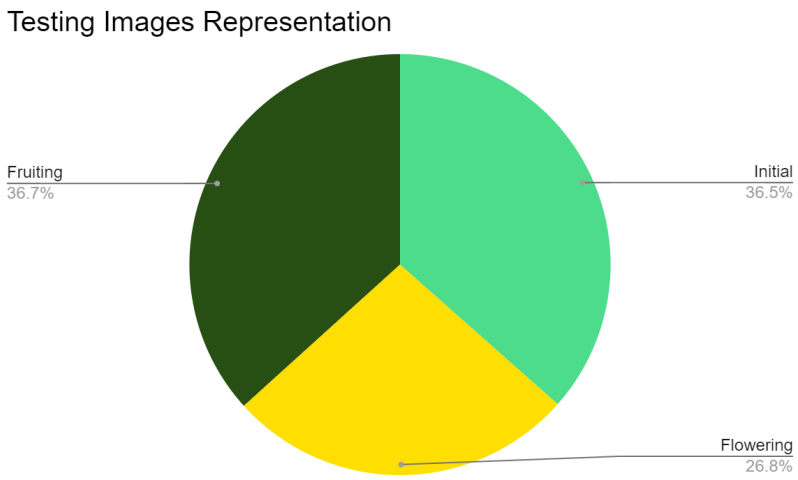

Fig. 13: Testing dataset representation

Results

The accuracy of our algorithm was around 91% which is very satisfactory. We expect this accuracy to slightly decrease in the field since some of the images used for training contained a white background that manually removed background noise.

Accuracy	Precision	Recall	F1 Score
90.7%	90.3%	94.7%	92.5%

Table 10: Growth Stage Classification Results

Deployment

The entire health model and growth stage algorithm are hosted on Heroku and a cron job is deployed on Heroku as well. It runs every 5 minutes (for testing purposes) and once per day (in a normal scenario). The cron job gets the path of unprocessed images from the database and retrieves them from the AWS S3 bucket. Then, the image is passed first through the growth stage classifier and the

current growth stage is predicted. Secondly, it passes through the segmenter which saves the leaf images in the virtual environment. The leaf images are then processed one by one and fed into the health model. If a leaf is found to be unhealthy, then the whole image is deemed to be unhealthy and this will be the prediction. Otherwise, it is considered healthy. The predictions are then written into the prediction table in the database with the associated group_id of the camera.

Web Application

Implementation

The web application was implemented using an MVC architecture, a structure where models, views, and controllers are used. Python's django web development framework was used to build the application along with a PostgreSQL database.

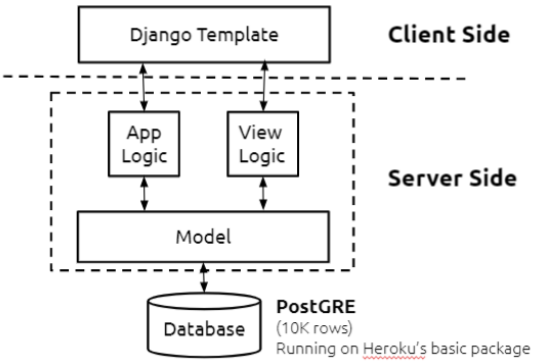

Fig. 14: Web application architecture

Backend

The database we used is Heroku's add on PostgreSQL database. The database is linked to our Django app using the models classes which represent every entity in the database.

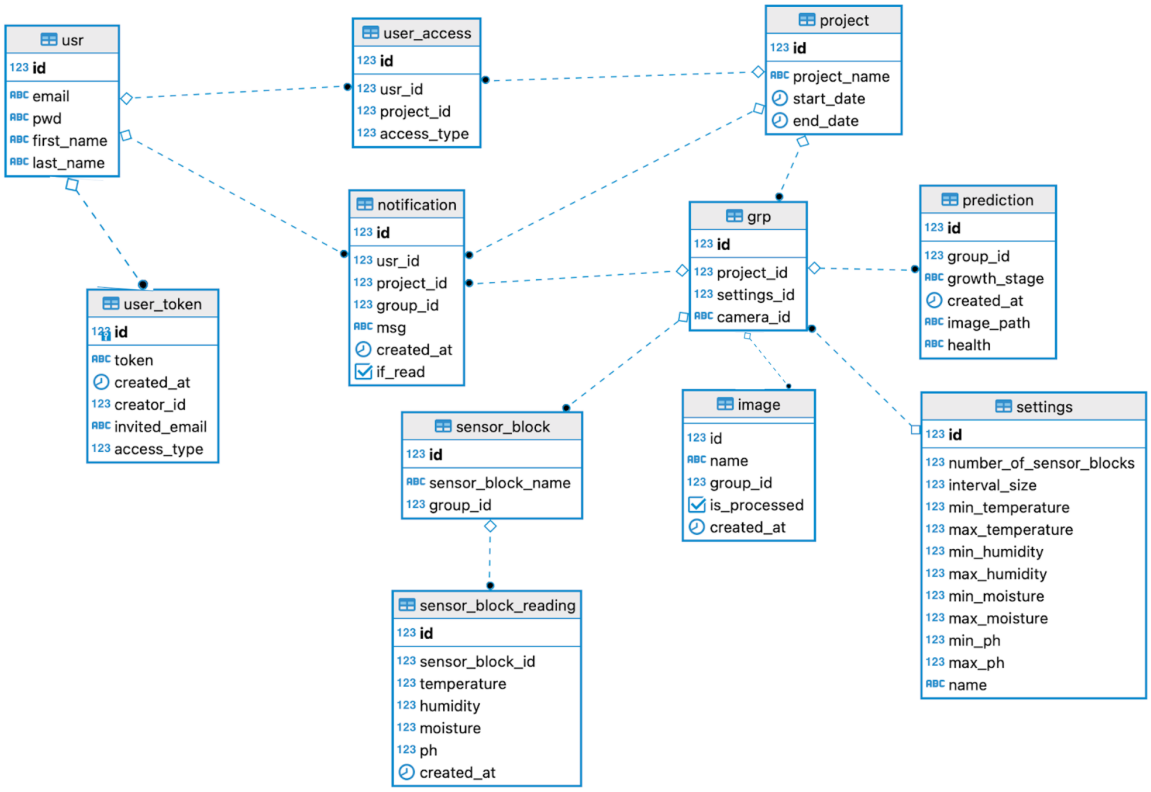

Fig. 14: Database ERD

The controller file in our application contains all the functions that retrieve data from the database such as sensor readings, user projects, and groups associated with each project. Django views are used to control the interaction between the frontend and the database through POST requests. Every page has its own view that is responsible for performing functionalities like reading and writing from the database, storing user sessions, and rendering the frontend. The urls.py file is where every view function is linked to its page's URL. The URL pattern is also defined there. Finally, a settings file is necessary to configure the connection to the database and any extra libraries/settings that we use.

Frontend

The frontend follows a template with a set Cascading Style Sheets (CSS) with our preconfigured colours, mostly following a colour scheme of green and blue, very similar to this report. It is written in HTML and uses Django's templates to interact with the backend through POST requests.

Testing

We followed an agile process while working on the web application by dividing it into smaller features that each of us worked on and tested rigorously on a local environment. When testing was deemed complete and the feature functioned perfectly, we would merge it with the code on the production environment and redeploy it on Heroku. Then, regression testing would take place to make sure that the added functionality did not affect existing functionalities. For more information you can refer to the traceability matrix in the SDD document.

Results

The results of our implementation and testing are as follows (in order of appearance in user scenario):

1. Effective registration and login into the web application*
2. Effective creation of projects*
3. Effective entry and storage of project settings**
4. Effective entry or loading then storage of group settings***
5. Effective entry and storage of sensor node names***
6. Effective recommendation of sensor nodes as access points on nearby devices****
7. Effective changing of interval settings****
8. Effective motion of the camera***
9. Effective timed snapshot-taking of camera stream on Raspberry Pi***

10. Effective uploading of snapshots to Cloud Cube***
11. Effective taking and uploading of all sensor readings****
12. Effective displaying of the latest group-specific or group-averaged sensor readings***
13. Effective displaying of weekly readings graph for each sensor***
14. Effective prediction of growth stage **(91% accuracy)** ***
15. Effective displaying of daily image captioned with growth stage prediction until fruiting is detected***
16. Effective prediction of health status **(94% accuracy)** ***
17. Effective displaying of daily image captioned with health status only if it has unhealthy leaves***
18. Effective generation of variable timespan report for all sensor readings***
19. Effective notification of safe range-overstepping sensor readings***
20. Effective email notification of safe range-overstepping sensor readings***
21. Effective changing of project and group settings** and ***
22. Effective sharing with other people as editors or viewers**
23. Effective sending of invitation with security token and registration redirection for sharing recipients**

* For each user

** For each project

*** For each group

**** For each sensor node

Deployment Plan

Regarding the deployment of the system, the following has been done. All of our code is published on GitHub with sufficient code description. The code will also be accompanied with both this report and the user guide. Both our manually collected growth stage and health datasets are published on Kaggle. SRS and SDD documents are also ready for those that request them.

As for our deployment at CARES, we will redirect them to the resources mentioned above. Considering that we were met with funding constraints, we could not deploy enough sensor nodes and cameras to be supplied to CARES as we had hoped. However, a major part of our deployment at CARES and globally is the web application as it is still up and running on Heroku and free for anyone to use. Also, the senior agricultural researcher at CARES, who was our supervisor, has noted that CARES is only interested in the publishing of our research for future large scale deployment by tech enterprises.

Furthermore, our health model could be deployed on any bell pepper, cucumber, tomato, and potato crops since it was trained and tested on plants of those kinds.

The links to our GitHub repository, Kaggle dataset, and web application are in the readme file.

Conclusion

In addressing the project's shortcomings, it is worthy of noting that our remote-capturing microservice is not fit for imperfectly paved agricultural areas due to the use of a vehicle with wheels. More research is needed on an automatic motion mechanism, to be effective for all agricultural areas without violating any (Egyptian) security laws, such as drones.

Moreover, the growth stage model should be modified to classify stages of more plants than cucumber since, as it stands, it is not a plant-generic model. Another shortcoming of the growth stage model is its training on white-background images which do not match the appearance of real-time captured images.

As for the health model, testing needs to include data not associated with the training data (ours, KU, and PV-K) in order to determine if the model has suffered from training-testing over-similarity. A way to fix this is to test the model on many images from the CARES greenhouse (not too similar to most of the training data), not just the 20% of our 173 images. This solution was unfortunately made infeasible by the early termination of CARES' experiment; however, we urge scientists to attempt the more vigorous testing.

The health model could also be revised to work on the 15 plants from trial #2. If scientists could raise the 73% to perhaps 90%, the model would be excellent for most greenhouses and fields worldwide.

However, we can confidently say that our system is a complete monitoring system that is reliable, flexible, and efficient. The system uses up-to-date technologies from all computer disciplines and integrates them into one cohesive entity.

Appendix: Glossary

AI
Artificial intelligence. The study of equipping machines with intelligent thinking.

CARES
Center for Applied Research on the Environment and Sustainability. An organization in AUC.

IoT
Internet of Things.

CSS
Cascading Style Sheets.

CV
Computer Vision. The field of computing that deals with allowing computers to gain understanding from digital images.

ML
Machine Learning. The field of computing that deals with automated data analysis.

IoT
Internet of Things. The field of computer engineering that deals with the embedding of real-life objects with sensors

Sensor Node / Sensor Blocks
Block of the sensors of our project: DHT11 (temp and humidity), moisture, and pH.

RPi
Raspberry Pi 4.0. A microcontroller with an operating system and Wi-Fi module.

ESP32
A system on a chip microcontroller with integrated Wi-Fi and dual-mode Bluetooth.