Graduating Gardeners

Author: Hiroko Abe, Sarah Jang, Kanon Kihara:
Electrical and Computer Engineering, Carnegie Mellon University

***Abstract* — An automatic greenhouse system capable of maintaining specific temperature, lighting, and watering conditions, as well as detecting plant growth status and defects with an accuracy of over 90%. The greenhouse is connected to an interactive web application where users can receive alerts if any changes occur to the plants, monitor their plants live, and manually control environmental variables in a timely fashion, where the greenhouse environment will be adjusted to the new settings within an hour. Compared to complex and expensive industrial greenhouse systems, we aim to provide an intuitive, compact, low-cost, and effective greenhouse system for new gardeners**.

***Index Terms* — Automation, Computer vision (CV), Edge Detection, Greenhouse, HSV Color Detection, Microcontroller, NumPy, OpenCV, Web application**

I. Introduction

The demand for gardening supplies has increased significantly and caused frequent supply shortages throughout 2020 as more and more people started gardening for the first time during the Covid-19 pandemic. For many who are new to gardening, creating a gardening environment at home could be an expensive and complicated task. Growing plants that have specific environmental needs is also challenging for beginners. Greenhouse technology has been around for a very long time, and there are many greenhouses on the market that have advanced features to increase environmental sustainability, plant growth, and productivity. However, these systems are intended for large-scale industrial usage, which require a lot of investment in infrastructure and specialized knowledge to operate. We are planning to implement some of these features that help optimize plant growth on a small-scale greenhouse system with a user-friendly website in order to cater towards normal people who want to enjoy gardening at home.

Our greenhouse system automatically maintains specific temperature, lighting, and watering conditions, and alerts users of plant growth status and defects. The greenhouse is connected to an interactive web application where users can receive the alerts, monitor their plants live, and manually control environmental variables. With the assumption that this greenhouse system will be used to grow common household plants, our goal was to build a greenhouse that is able to maintain a target soil moisture percentage with a 5% tolerance, a target temperature with a 5°F tolerance, and provide the plants with a target of 4 hours of light everyday. The system should accurately detect plant growth stages and defects with an accuracy greater than 90%. When a user changes the target temperature, soil moisture, or light duration on the web application, the new target should be met within an hour of the command. The web application UI should be intuitive and user friendly to provide the user with an effortless, yet successful gardening experience.

II. Design Requirements

In order to set specific, quantitative requirements for our project, we decided to focus the scope of our greenhouse to automatically grow pea shoots. They are relatively easy to grow and only take about 2 to 3 weeks until harvest, which gives us enough time to test the greenhouse with a full growth cycle of the pea shoots. Based on specific growing conditions of pea shoots, we are requiring our greenhouse to be able to maintain a target soil moisture percentage (volumetric water content) with a 5% tolerance, in order to ensure that the soil moisture percentage will stay well in between the field capacity and permanent wilting point. We are also requiring our greenhouse to be able to maintain the target temperature with a 5°F tolerance, since pea shoots should ideally be grown in temperatures between 55°F and 65°F. By setting the minimum target temperature to 60°F, we are ensuring that the ideal temperature range is maintained within the greenhouse even during the winter. The greenhouse should be able to provide a target amount of light to the plants (in hours/day), which is determined by the lighting schedule set by the user on the web application. The greenhouse needs to adjust quickly to manual changes of environmental variables too; when a user changes the target temperature, soil moisture, or light duration on the web application, the new target should be met within an hour of the command.

On the software side, we require that our greenhouse is able to classify certain growth stages like germination, young plant, flowering/ fruiting, and harvest with an accuracy of greater than 90%, and detect common defects like diseases and withering with a false positive rate of less than 10% and a false negative rate of less than 5%. We also expect the stem bending detection to categorize stems as bending greater or less than 45° with an error rate less than 10%. Different plants have different growth stages, but in the case of pea shoots, growth stages like sprouting and harvesting can be classified based on simple metrics such as height and the number of leaves, so we are requiring our CV algorithm to accurately classify the plants' growth stages. Detecting plant defects is more challenging, but since we do not want to risk ignoring plant defects that lead to serious complications, we are tolerating a higher false positive rate than the false negative rate. Security is also an important aspect, so we will map a single account registered on the web application to a single greenhouse by asking users to create an account with a password and authenticating requests to control the greenhouse. The user's commands to change temperature, soil moisture, or light duration conditions should be sent and received by the greenhouse hardware within a minute to help ensure that the previous requirement of adjusting conditions within an hour of the command is met. We plan to live stream the plant on the web application, which will require a camera with day and night vision, a wide field of view, and a reliable connection to keep the video streaming available 24/7.

Through the same wifi, we hope that the live stream latency is less than 10 seconds. Finally, the UI of the web application should be intuitive and easy to navigate to provide the user with a pleasant experience.

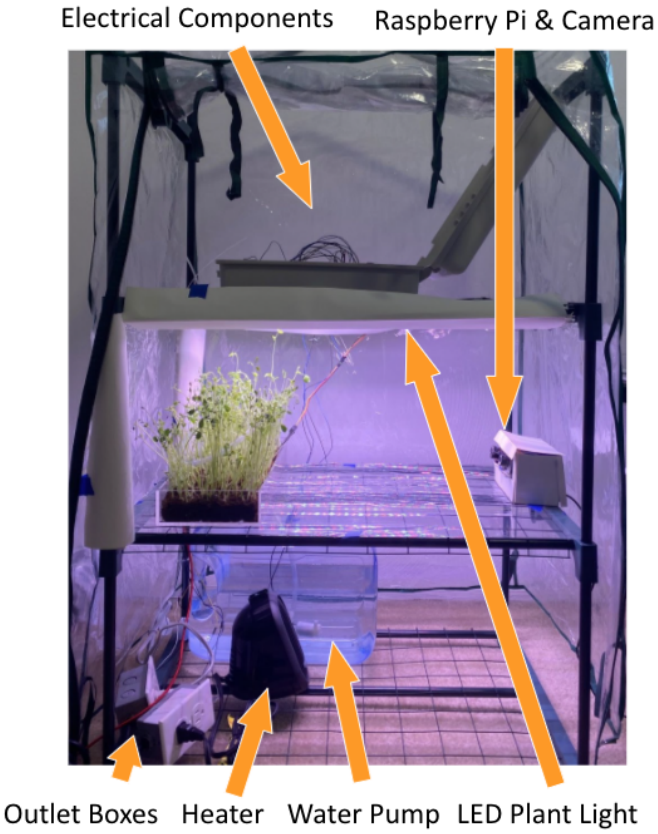


Fig. 1. Labeled picture of the greenhouse in operation

III. Architecture and/or Principle of Operation

In our project, we have 3 main crucial components which are

- Hardware system
- CV analysis
- Web application

The key to the success of this project will be the hardware system setup. In Fig. 1, we have overall system design of the greenhouse, indicating the location of each of the components. Because the greenhouse has shelves, we can place a water tank and plant grow light on the top shelf. On the middle shelf, we will have two plant trays with moisture sensors, Raspberry Pi that holds our IR-cut camera, and ESP32. We will also have two water pumps for each of the plant trays. Lastly, on the bottom shelf, we have a mini heater, 5V relay, and extension cord. Unlike what we indicated in our initial design report, we decided to get rid of the fan or cooling system because we were not able to find any compact AC within our budget. Therefore, we are only focusing on the heating system.

A soil moisture sensor, light intensity sensor, and temperature sensor are connected to the ESP32 board. The data collected from these sensors are processed within ESP32 and then sent to our web application backend by using AWS IoT and AWS DynamoDB. ESP32 also sends signals to the 5V relay so that it can turn on/off the mini heater, water pumps, and LED plant grow light.

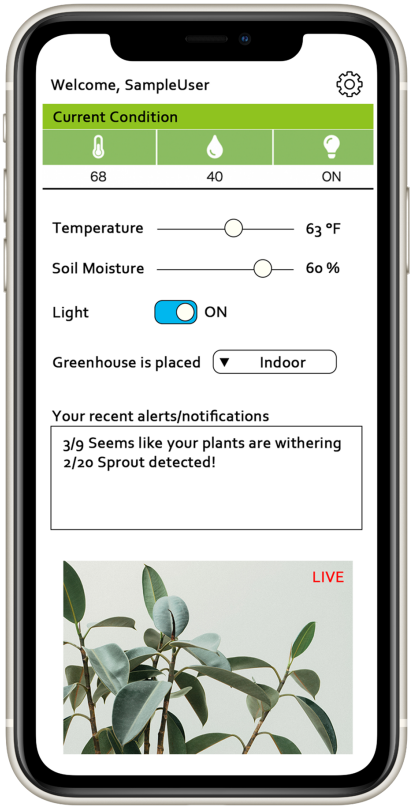


Fig. 2. Initial web application UI design

The plants are always monitored by an IR-cut camera that has day and night vision. The captured video is sent to a web application via WI-FI and with a live streaming script. This video or image is also sent to our CV analysis system which is operated by OpenCV. We have 3 detections within our CV algorithm: leaf shape detection, HSV color detection, and stem detection. OpenCV is able to achieve these detections swiftly as it already has HSV color detection, edge detection, size/shape detection functions within its library. Our 3 main detections are then used for growth stage classifier, defect detector, and stem/vine bending measurer. CV analysis information is sent to the web application backend for further use.

Our users will interact most with our web application. The user will first register or login to our web application by creating a new account or logging in with Google OAuth2. Then the user will be led to the main user page that shows the information about their greenhouse, shown in Fig. 2. While the user can monitor the live video of the greenhouse, the user page also indicates the current temperature and soil moisture, and whether the light is turned on/off. The user can change the value of these parameters manually within the web application. The user can also indicate if the greenhouse is placed indoors or outdoors. This information is necessary since regular light intensity sensors cannot differentiate between indoor lighting and sunlight, but indoor lighting does

not contain the necessary wavelengths for optimum plant growth. Therefore, we assume that all light measured by the light intensity sensor is indoor lighting if the user indicates that the greenhouse is located indoors, while we assume that all light measured by the light intensity sensor is sunlight if the user indicates that the greenhouse is located outdoors. By using data from our hardware and CV analysis, we send out notifications to users through our web application. For example, when the temperature gets too high or if some of the plants start to wither, the user receives a SMS notification on their mobile phone. The user can also specify what kind of notifications or how often they want to receive them.

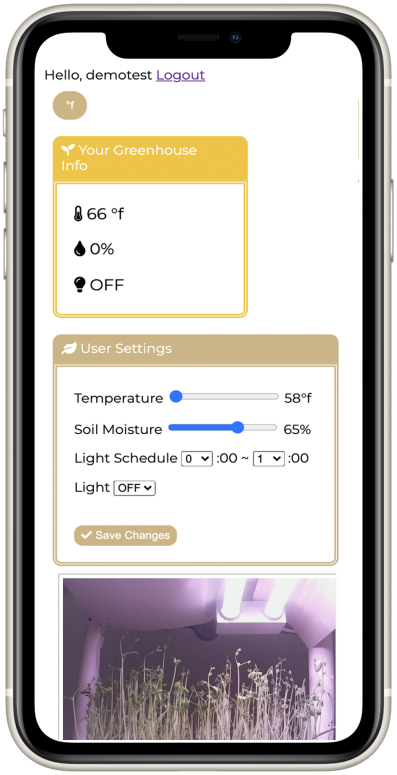


Fig. 3. Final web application UI

There are several changes that we made to the design as we progressed. Within the greenhouse, we were initially planning to place two plant trays on the middle shelf. However, because we also had to place the RPi camera on the middle shelf, and it requires a certain distance to perform CV on our plants, we decided to set up only one plant tray. As a result of this, we are no longer using our second soil moisture sensor. Another change we made was with the light. Our initial plan was to automate turning on/off the LED grow light depending on how much sunlight the plant has already received. However, it was difficult for us to move our entire greenhouse outside and conduct multiple tests to calibrate the light intensity for the sunlight. Therefore, we got rid of this function, and instead, the users can now schedule the light on their own (Fig. 3). Our goal of web UI design was to fit every component within a single page without scrolling because that is how most of the mobile app works. Because our new UI design took up a bit more space than we expected, we decided to remove a notification section to achieve our goal. For the CV application, we added a background removal and pot removal step, because there was a lot of noise in our video that kept making our analysis jump from one measurement to another. Instead of distinguishing leaves and stems by their shapes, we decided to extract the leaves with HSV Color Detection. Therefore, we measure the angle of a stem between the vertical axis and the line that goes from the plant center to the flower centers collected from the flower detector, as this approach was producing much more accurate stem bending measurements and bending seems to happen mostly when fruits or flowers are weighing the plant down. Finally, our website will no longer be deployed on EC2. While one greenhouse should be linked to one user, our currency setup provides a system where every user is linked to one greenhouse. This is because there was no time for us to build another greenhouse and test out this user linking process. Therefore, there was a concurrency issue when multiple users try to change parameters on a deployed website. Moreover, our live video only works when the website is under the same wifi as the RPi camera. Hence, we realized that deployment on EC2 is not what we want to focus within this project.

IV. Design Trade Studies

V. ESP32 vs Arduino vs. Raspberry Pi

TABLE I. Microcontroller Comparison

Board	Comparison Aspect		
	Required Parts	***Price (Sparkfun)***	***Max Temp***
ESP32S	Micro USB Cable	$19.95	125°C
Arduino Uno	USB-B Cable	$22.95	85°C
Raspberry Pi 3 B+	Micro USB Cable, SD Card, SD Card Reader	$35.00	85°C

We are using an ESP32 board to gather sensor data, send sensor data to AWS, and control the 5V relays. There are other microcontrollers like the Raspberry Pi and Arduino that could also perform the same functions. All 3 of these boards support WiFi and Bluetooth connectivity, and can be connected to temperature, soil moisture, and light intensity sensors. However, there are some key differences between them that helped us decide to use the ESP32 board. Table 1 above shows that the ESP32 board is the cheapest option, and can operate at a higher temperature than the Raspberry Pi or the Arduino Uno. While it is true that the Raspberry Pi is more expensive because it has more memory and processing power than the other 2 boards, our greenhouse does not require those extra capabilities since most of the data will be stored and processed using AWS IoT. It is also worth noting that some household plants that originate from tropical regions thrive in higher temperatures, so it is safer to use a board with a higher maximum operating temperature just in case the internal temperature of the greenhouse is set very high.

VI. Irrigation Systems

While designing the automatic watering system, we compared different types of irrigation systems to choose the most effective one for our purpose. The 3 common types of irrigation that could be used within a greenhouse are drip irrigation, sprinkler irrigation, and sub-irrigation. Drip irrigation delivers water at or near the root of plants using a drip, spray or stream. Sprinkler irrigation utilizes overhead high-pressure sprinklers or guns to distribute water. Sub-irrigation distributes water across land by raising the water table through a system of pumping stations, canals, gates, or ditches. A simple drip irrigation system only requires a water pump and a soaker hose to build, while evaporation and runoff are minimized. Sprinkler irrigation is also simple to implement, since it only requires a hose connected to a sprinkler. However, there would be a need to carefully contain the water within the greenhouse since the greenhouse is mainly tested for use indoors. Sub-irrigation can control water flow more efficiently and precisely than a sprinkler, but would be the most complicated to implement, since it requires a system of pumping stations, canals, gates, or ditches to control the water table. Thus, we came to the conclusion that the drip irrigation system would be the most simple and effective way to water the plants within the greenhouse.

A. AWS vs. Other Cloud Solutions

As mentioned before, we are using AWS IoT in order to send/receive commands to/from our ESP32. Even though there are many other cloud solutions such as Microsoft Azure, we wanted to achieve a quicker communication between hardware and software by completing all systems within AWS.

Similarly, while we have options such as MongoDB that can hold more complex data, we decided to use AWS DynamoDB as we will be able transfer our data quickly from/to AWS IoT when we try to send commands to the hardware. Moreover, all the data we need to store are simple values, therefore DynamoDB is sufficient for our purpose.

B. Django vs. Other backend web frames

We chose Django over other web frameworks because it is a high-end Python web framework. Our CV analysis will also be written in Python. Once our CV analysis detects problems on plants, we need to organize this information and send them to the users as a notification. Therefore, it is the best practice for us to use Django and keep our overall programming language the same by using Python for faster communication.

C. Twilio vs Nexmo SMS Messaging

Our SMS API, Twilio, can be also customized with Python. We also looked into other SMS APIs but most of them have limited or superabundant services. For example, Nexmo SMS Messaging, the one we were also looking at, has many features including calling and also global texting; however, these features are unnecessary for the project. The cost performance of Twilio also seemed reasonable. The overall cost depends on how many messages we receive/send, so it would stay within our budget since the greenhouse system sends only a couple messages per day.

D. Computer Vision vs. Machine Learning Model

Initially, we were planning on implementing a Machine Learning model to categorize growth stages and to compare healthy or diseased plant images to the greenhouse plants, but through further research we found that this process holds high risks and complicates visual plant growth analysis. Because every plant grows so differently even within a species of plants, there is much room for error and misleading classification with image-fed learning models. We would also have had to be reliant on common images from online for our training data, but with most images online taken with specific lighting and grown in commercial amounts, such data would not be appropriate for the small scale system we've designed.

Alternatively, computer vision applications are much more common and foreseeable in smart planting systems. With the CV approach, not only is a growth stage classifier and defect detection possible, but also vine bending is much more viable. We could much more efficiently customize the analysis to work with the lighting system that the plants are under as well, as opposed to the online images we may have had to rely on for the learning model. The growth stage classifier could also be much easily implemented with a pixel per metric method, where the height of the plant in the image can be converted to real-life size by a conversion constant, rather than being trained with images. Because of the simplicity and safety of the CV approach compared to the learning model, we decided to take the former path.

E. Raspberry Pi IR-Cut Camera vs. Logitech C920

TABLE I. Camera Comparison

Camera	Price	Resolution	Strengths
RPi IR-Cut	$24.99	1080p/30fps	Day & Night Vision
Logitech C920	$69.99	1080p/30fps	RightLight Tech.

Initially, we had looked into the Logitech C920 camera, as it is a popular webcam for CV projects and it had RightLight technology, which adjusts the video quality to the lighting conditions. This would reduce the complications of trying to adjust the LED lights of the greenhouse for the pea shoots to be visible. Further, with constant image quality, we would not need to change the specific HSV color values for edge detection of the video. However, when we pivoted to adding a live stream monitoring system, we realized that we would need both day and night vision. The RPi IR-Cut camera was the only camera we could find which integrated both vision types in one camera, and we would've needed to add a filter or work with two cameras if we chose the Logitech C290. Because of the lower price, equal resolution, and the day and night camera, we continued with the RPi IR-Cut camera. As for the lighting conditions for high quality imaging, we adjust the LED plant lights to accommodate for the RPi camera.

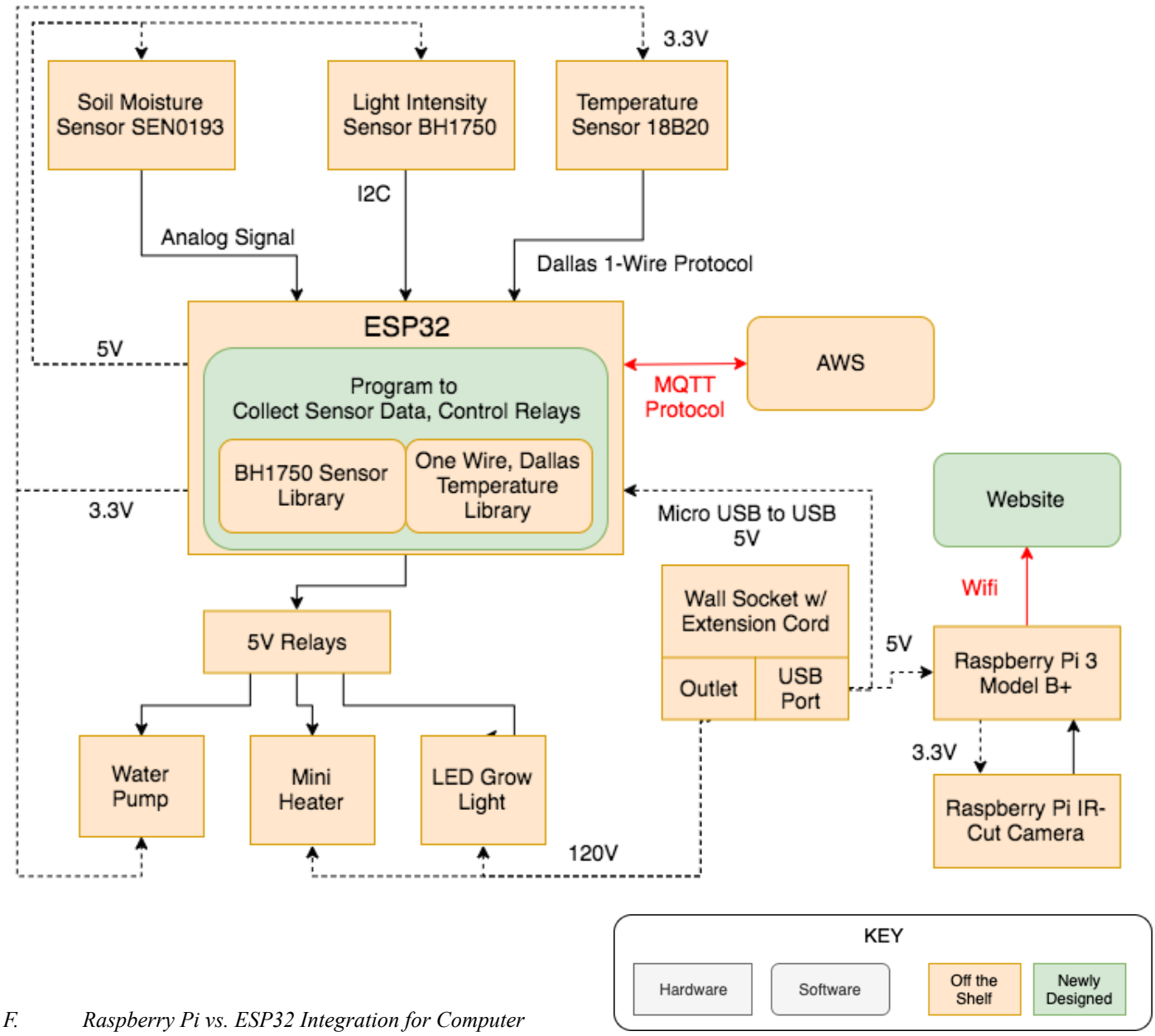


Fig. 4. Hardware Block Diagram

F. Raspberry Pi vs. ESP32 Integration for Computer Vision

Initially, we were planning on connecting the Raspberry Pi Camera to the ESP32 and uploading the CV application to the ESP32 since the ESP32 is cheaper and all the hardware would be connected to one board. However, the pins used to connect a RPi camera module to the ESP32 are not compatible. Further, there are no ESP32 camera modules that have both day and night vision, which is integral for the live stream. Because Sarah and Hiroko are working separately, we thought it would be best to work with hardware that suits their respective components that they were working on. Further, the RPi was available for rental from the ECE department. Hence, we decided to implement the CV application on the RPi, since there is a larger community of CV projects made with Raspberry Pi Camera Modules, the Raspberry Pi ensures the processing power for constant live streaming, the RPi could be borrowed, and its compatible camera module comes with both day and night vision.

VII. System Description

A. Hardware Subsystem

The hardware subsystem of the greenhouse revolves around the ESP32 board that gathers sensor data, sends sensor data to AWS, and controls the watering, heating, and lighting systems. Temperature is measured by the waterproof 18B20 temperature sensor, which uses the Dallas 1-Wire Protocol to send the data over to the ESP32 using 1 GPIO port. Light intensity is measured by the BH1750 photodetector, which communicates with the ESP32 via the I2C bus. In order to extract the sensor data from the input signals, the ESP32 has the BH1750 sensor library, OneWire library, and the DallasTemperature library installed. As shown in Fig. 1, our greenhouse has space for 1 tray of pea shoot plants, so there is

a capacitive soil moisture sensor SEN0193 that sends analog signals to the ESP32, which has built-in ADCs to convert them into digital signals. All of the sensors except the temperature sensor are powered by the 5V output from the ESP32, while the temperature sensor is powered through the data line. Once the ESP32 has all of the input signals converted into temperature, soil moisture percentage, and light intensity values, the information is sent to AWS using the MQTT protocol, so that the software subsystem can analyze the data.

On the other hand, the ESP32 receives commands from AWS to turn the water pumps, heater, and LED plant light on/off. The ESP32 is connected to 5V relays that are wired to an electrical outlet where the various appliances are plugged into, which gives the board the power to turn the appliances on and off. As shown in Fig. 1, the small space heater is placed at the bottom of the greenhouse shelf since the warm air will rise and create a convection current to maintain a constant temperature throughout the greenhouse. The special LED plant light is placed above the plants to provide light with PAR, the range of light that can be used by plants to photosynthesise. A water pump is placed inside a water tank to pump water through a tube to water the plant tray.

In order to provide the computer vision subsystem with live images of the plant, a Raspberry Pi IR-cut camera connected to a Raspberry Pi is placed inside of the greenhouse shelf. The details of these components will be explained later in the computer vision section, since the camera has been separated from the other sensors and appliances in order for us to parallelize the workflow. The camera and Raspberry Pi were brought to Pittsburgh and integrated into the physical greenhouse system once the computer vision algorithm was completed towards the end of the semester.

After the electric components were wired and placed within the greenhouse, we used a waterproof electrical junction box to contain the sensors and microcontrollers to protect the parts from water.

B. Web Application Subsystem

Web application backend is developed with Django while the frontend is developed with Javascript and HTML. With Twilio API, we send out SMS alerts and notifications to users. The data used within our web application is stored in AWS DynamoDB.

From Fig. 5, the ESP32 sends data to AWS DynamoDB using AWS IoT with MQTT protocol and AWS Lambda. From here, our backend retrieves this data from the database by using boto3, which is an AWS software development kit for python, and renders these values on the website. Whenever the user modifies the parameters of the greenhouse, this data is sent to AWS DynamoDB and then to AWS IoT by using python requests library, so that our web application and hardware can communicate with each other. We are sending a 3-bit signal back to ESP32 via AWS IoT. Each bit corresponds to turning on/off the heater, water pump, and LED grow light.

The sensor data from ESP32 is further analyzed within the backend. Especially for the soil moisture sensor, because it is sending analog signals, we calibrate these values into soil moisture percentage that is easier for the users to understand. We are also using these data to send users notifications. For example, if the temperature target is met, we will send users a notification and turn off the heater automatically. If it goes below the target temperature, we will automatically turn the heater on while also sending alerts. We are periodically checking whether these target values are met by having a python script run every 10 minutes to analyze the data. Moreover, even when the user is not interacting with the website, AWS IoT is constantly sharing information from the hardware to the web application every 15 seconds. Hence, within the database, the current information of the greenhouse is always updated.

From Fig. 6, the data from CV analysis and live streaming system is handled in a similar manner. By using our live stream script, the unprocessed video data is directly rendered onto the website using the IP address of the IR-Cut Camera. The data from the camera is also analyzed within OpenCV. This analyzed data holds information about growth stage and defects. The backend receives this information and sends alerts or notifications accordingly.

The website frontend structure is pretty simple because all the important information is handled within the backend. It is a single page application that the user can access from a PC and also from their mobile phones.

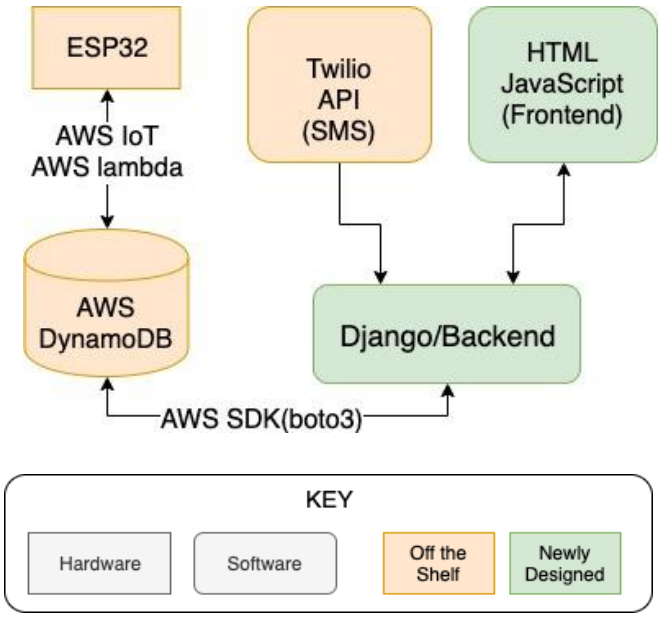


Fig. 5. Web Application Block diagram

C. *Computer Vision Subsystem*

The computer vision application is reliant on the OpenCV and NumPy libraries, specifically in the HSV Color Detection and Edge Detection functionalities. We first extract the contours of the plants using a more rigorous edge detection algorithm and white out the pixels outside of the contour which is the background so that the colors of the background do not interfere with the later analysis. Then, the background-removed image goes through a pot removal

algorithm which looks for an object that is of typical pot HSV values and size. During this process, we collect the midpoint of the pot which is the plant center and whites out the pot for cleaner analysis later on.

There are three main components in the CV application: the classification of four growth stages (germination, young plant, flowering/ fruiting, and harvest), the detection of common diseases or withering, and the measurer for extreme vine or stem bend. The application is integrated onto a Raspberry Pi 3 Model B+ and a real-time video is captured through the Raspberry Pi IR-Cut Camera Module which includes both day and night vision for 24/ 7 monitoring. The camera must be placed sideways to capture a side view of the pea shoots in order for the detectors to work properly.

The first parameter that determines which growth stage the pea shoots are at is the height of the plant, which is measured through the pixel per metric method. To initialize the method, it takes in the distance from the camera to an object along with the height of an object, and divides the pixels by the real height of the object. With this pixel per metric conversion scale, we multiply it by the height of the pixels captured in the camera to determine the real life plant height. The second parameter that most likely distinguishes flowering from harvest is the detection of flower and fruits, which is implemented by passing another layer of the HSV Color Detection onto the non-stem parts, since the leaves and flower shapes of pea shoots are similar. We also collect the centers of the clusters of flowers or fruits for later analysis.

The defect detector analyzes the leaves distinguished by the leaf shape detector and searches for any spotting, a common disease and insect bite pattern. For withering, we look for yellowish or brownish colors.

For detecting extreme bending of stems or vines, we find the line between each flower or fruit center and the plant center collected earlier. Then, we find the angle between each of these lines and the positive y-axis that goes through the plant center. Any angle greater than 45 degrees is considered extreme and is advised to be staked.

The first time a growth stage is reached, the user will be alerted. Users are also notified of every new defect or withering leaf that is detected. Notifications are sent as a SMS text message through the Twilio API.

Further, there is a live stream monitoring system that is captured and ran through the same Raspberry Pi and camera used for CV analysis. The live stream is available through referencing its IP address and is embedded into the website for the user to interact with.

For further information and details, refer to the block diagram, Fig. 6.

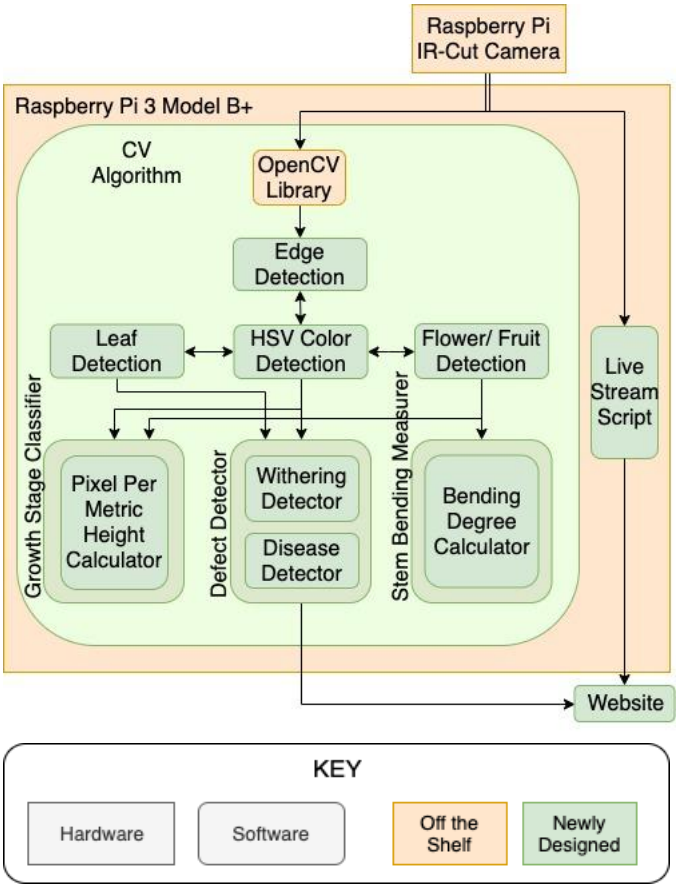


Fig. 6. Computer Vision Application Block Diagram

VIII. Test and Validation

A. *Results for Computer Vision Application*

For the CV application, there were 5 separate tests for the growth stage classifier, disease detection, withering detection, stem bending, and latency.

For the growth stage classifier, 15 images of a pea shoot plant were taken over 12 days, and 5 images of flowering plants with 4 of them being different snapdragons each in different growth stages were tested. The tallest pea shoot's height and the flower plants' heights were measured with a ruler and predicted through CV, and we saved the difference between these heights as the height difference. The tallest pea shoots' height differences averaged about .47 cm. The flower and fruit detection was also tested, and with no errors from the flower/ fruit detection, we achieved an error rate of 10% which was our initial ideal threshold. Most of the error rate seemed to come from the transition period of when a plant goes from one stage to another. Pea shoots' only indication of maturity are its height, so without the flower/ fruit detection applied to reaffirm a growth stage, there was more error in classifying the pea shoot growth stage.

TABLE II. GROWTH STAGE CLASSIFIER

Growth Day	Flowering Error (0->No Error, 1->Error)	Real Stage vs Classified Stage (0->Same, 1->Different)	Height Difference
Day 1	0	0	0
Day 2	0	0	0.008
Day 3	0	1	0.569
Day 4	0	0	0.218
Day 5	0	0	0.006
Day 6	0	0	0.193
Day 7	0	0	0.279
Day 8	0	0	0.898
Day 9	0	0	1.012
Day 9.5	0	0	0.587
Day 10	0	1	0.273
Day 10.5	0	0	0.601
Day 11	0	0	1.318
Day 11.5	0	0	0.596
Day 12	0	0	0.5
Bought	0	0	0.301
Bought	0	0	0.474
Bought	0	0	0.303
Bought	0	0	0.944
Bought	0	0	0.257
		Error Rate (%): 10	Avg: 0.46685

Pea Shoot
Snap Dragon Sprouting
Snap Dragon Young No Flowering
Purple Pansy Flowering
Blue Pansy Flowering
Snap Dragon Flowering
Seed Geranium Flowering

Since it is difficult to assert diseases onto a plant and due to safety concerns, we simulated white and dark spotting diseases. For the white spotting, we used flour and some cotton to simulate moldy diseases. For the dark spotting, we drew some dots and patches of disease with black and red ink pen. Simulating disease on some leaves while keeping other leaves clean, we took 5 different side profiles for each 4 flower plants which in total are 20 images: the blue pansy, the purple pansy, the pink snapdragon, and the red seed geranium. A patch is considered the area of a bounding box that the CV analysis draws that contains a cluster of spotting or a large rectangular area outside the bounding box that doesn't contain spotting. For each image, we count the total number of patches, the number of patches that contain spotting which are the true positives and the number of patches that contain healthy green leaves which are the true negatives. We then run the analysis, and count the number of bounding boxes in the disease detection image that don't contain any spotting which is the false positives. After, we look for clusters of spotting that are not in bounding boxes and count them as the false negatives. With this data, we calculate the false positive and negative rates and find that for white spotting, the false positive rate is around 2% which is less than the 10% we had in mind and the false negative is around 3% which is less than the 5% goal. For the dark spotting, the false positive rate was higher than our goal with 15% and the false negative rate was higher as well with 9%. This could be because the shadow and dents of the leaves are being confused for disease patterns, while white stands out much more in the HSV scale.

TABLE III. DISEASE DETECTION

Image	# White Spotting False Positives Patches	# White Spotting False Negatives Patches	# Dark Spotting False Positive Patches	# Dark Spotting False Negatives Patches
Blue Pansy View 1	0	0	1	0
Blue Pansy View 2	1	0	1	0
Blue Pansy View 3	0	1	0	1
Blue Pansy View 4	0	0	1	0
Blue Pansy View 5	0	0	0	1
Purple Pansy View 1	1	0	2	0
Purple Pansy View 2	0	0	2	0
Purple Pansy View 3	0	1	1	1
Purple Pansy View 4	0	0	1	0
Purple Pansy View 5	0	0	3	1
Snap Dragon Flowering 1	0	0	0	0
Snap Dragon Flowering 2	0	0	1	1
Snap Dragon Flowering 3	0	0	0	0
Snap Dragon Flowering 4	0	0	2	0
Snap Dragon Flowering 5	0	0	1	0
Seed Geranium Flowering	0	0	0	1
Seed Geranium Flowering	0	0	1	0
Seed Geranium Flowering	0	0	1	0
Seed Geranium Flowering	0	0	1	0
Seed Geranium Flowering	0	0	0	0
	True Positive White Spotting: 61	True Negative White Spotting: 97	True Positive Dark Spotting: 58	True Negative Dark Spotting: 105
	False Positive Rate (%): 2.02	False Negative Rate (%): 3.17	False Positive Rate (%): 15.32	False Negative Rate (%): 9.38

The withering was also tested with 20 images, but we tested with just pea shoots, blue pansy, and purple pansy plants. Once the first pea shoots matured, sections of the pea shoots and soil were cut out and dried while the remaining soil and pea shoots in the planting tray were watered as usual. Then, the next days after that the dried pea shoots and soil patches were placed in different spots of the planting tray, and we ran the CV analysis which grouped withered pea shoots in bounding boxes. Blue and purple pansy flowers with some withered leaves mixed in were bought so that we could test whether the CV could distinguish healthy leaves from withered leaves. The true positive is the number of healthy leaves in the image, while the true negative is the number of withered leaves in the image. Then, the false positive is the number of healthy leaves that were within the bounding boxes that were meant for withered leaves. The false negative is the number of withered leaves that were undetected. With these measurements, the false positive rate is around 8% which is less than the 10% goal metric, and the false negative rate is 5.4% which is slightly higher than the 5% goal metric. There seems to be more false positives and negatives in the flowering plants than the pea shoots. This may be due to the larger range of colors present in the pansy flower plants that may contain the HSV parameters similar to those of yellow-brown colors of withering leaves.

TABLE IV. WITHERING DETECTION

Image	# of False Positives	# of False Negatives
Peashoots 1, Day 15	1	1
Peashoots 1, Day 16	0	0
Peashoots 1, Day 17	0	0
Peashoots 1, Day 18	1	0
Peashoots 1, Day 19	0	0
Peashoots 2, Day 15	1	0
Peashoots 2, Day 16	1	0
Peashoots 2, Day 17	0	0
Peashoots 2, Day 18	0	0
Peashoots 2, Day 19	0	0
Blue Pansy View 1	2	1
Blue Pansy View 2	0	0
Blue Pansy View 3	0	0
Blue Pansy View 4	1	0
Blue Pansy View 5	1	1
Purple Pansy View 1	0	1
Purple Pansy View 2	0	0
Purple Pansy View 3	2	1
Purple Pansy View 4	1	1
Purple Pansy View 5	0	0
	True Positive: 105	True Negative: 127
	False Positive Rate (%): 7.97	False Negative Rate (%): 5.41

For the stem binding, 5 different side view images of the blue pansy, purple pansy, pink snapdragon, and red seed geranium flowers are analyzed. For each image, we count the number of stems which have a fruit or flower on the top and are connected to the plant center on the bottom. Then, we measure the real angle that the flower stem is relative to the vertical axis that goes through the plant center and count how many are categorized as greater than or less than 45°. The image goes through the CV application, and we calculate the stem angles from the y-axis and count how many stems are categorized as greater than or less than 45°. For each image, the largest angle difference between a real stem angle and the CV's stem angle measurement is calculated, and we compare the number of stems categorized greater than or less than 45° between the real data and the CV analysis output. If the numbers do not match, we add that to the number of stems erroneously calculated. We then divide the number of errors by the total number of stems. The stem bending measurer has around a 9% error rate, and the average largest angle difference is 5°. The error rate comes from the stems that are around 45° which may be categorized incorrectly due to the small differences in measured and predicted angles. The slightly off flower centers also lead to slight differences between the angle measurements.

TABLE V. STEM BENDING DETECTION

Image	# Stems (Flower/ Fruit Clusters to the Plant Center)	# Errors	Highest Angle Difference (degrees °):
Blue Pansy View 1	5	1	3
Blue Pansy View 2	4	1	2
Blue Pansy View 3	2	0	6
Blue Pansy View 4	3	0	7
Blue Pansy View 5	5	0	3
Purple Pansy View 1	5	0	4
Purple Pansy View 2	4	1	4
Purple Pansy View 3	3	0	4
Purple Pansy View 4	4	0	6
Purple Pansy View 5	3	0	5
Snap Dragon Flowering 1	2	1	10
Snap Dragon Flowering 2	2	0	9
Snap Dragon Flowering 3	3	0	2
Snap Dragon Flowering 4	4	0	7
Snap Dragon Flowering 5	3	0	9
Seed Geranium Flowering 1	3	1	3
Seed Geranium Flowering 2	2	1	8
Seed Geranium Flowering 3	2	0	2
Seed Geranium Flowering 4	3	0	2
Seed Geranium Flowering 5	2	0	4
		Error Rate (%): 9.38	Avg. Angle Diff.: 5

For the latency, an object was placed near the plants and we recorded the time it took between the object being placed and the object appearing in the live stream. The average latency time was 8.77 seconds, which is less than the 10 second metric goal we had set.

TABLE VI. LATENCY

Delay (seconds)	9.12	8.74	9.32	7.69	6.48	10.54	7.42	11.58	7.79	9.01
Average: 8.77										

B. Results for Data Transmission

There are two components to our data transmission: between hardware and web application, and between hardware and DynamoDB. Between hardware and web application, we had to time how long it takes for the hardware to receive 3-bits signal from our web application backend. To determine this, we printed out the time stamp on the backend when it sent the signal and did the same on ESP32 once it received the data. Table 7 is our result. We conducted 10 tests and achieved an average of 1.76 seconds to send/receive data between hardware and web application.

Between hardware and DynamoDB, we also did a similar test by printing out a time stamp of ESP32 once it sends sensor data and compared that time stamp to the one that is stored within DynamoDB. The time stamp within DynamoDB indicates when it received the data. We again conducted 10 tests. From Table 8, our result showed the average of 1.75

seconds. We actually have conducted another 10 other tests twice on different days, and we got an average of 0.29 seconds and 12.6 seconds. While we are not sure the exact cause of this fluctuation, we assumed this was because of our internet upload speed and operating conditions of AWS.

Because our initial goal was to transfer data within an hour, we achieved this goal as the transmission took less than 2 seconds on average in both cases.

TABLE VII. DATA TRANSMISSION FROM WEB APP TO HARDWARE

Time stamp (web)	Time Stamp (hardware)	Time Difference (sec)
13:56:32.438969	13:56:35.777	3.338031
13:57:46.985887	13:57:49.653	2.667113
13:58:6.27398	13:58:07.430	1.15602
13:58:19.935543	13:58:20.785	0.849457
13:58:35.108173	13:58:37.047	1.938827
13:58:47.736421	13:58:48.966	1.2233579
13:58:59.339266	13:59:00.747	1.407734
13:59:12.373659	13:59:14.064	1.690341
13:59:23.139495	13:59:24.460	1.32055
13:59:32.858402	13:59:34.856	1.997598
		Avg =1.76

TABLE VIII. DATA TRANSMISSION FROM HARDWARE TO WEB APP

Time stamp (web)	Time Stamp (hardware)	Time Difference (sec)
14:09:11.509	13:56:35.777	3.338031
14:09:26.276	13:57:49.653	2.667113
14:09:41.025	13:58:07.430	1.15602
14:09:55.819	13:58:20.785	0.849457
14:10:10.585	13:58:37.047	1.938827
14:10:25.363	13:58:48.966	1.2233579
14:10:40.128	13:59:00.747	1.407734
14:10:54.874	13:59:14.064	1.690341
14:11:09.650	13:59:24.460	1.32055
14:11:24.440	13:59:34.856	1.997598
		Avg =1.75

C. *Results for Web Application*

For the web application, we decided to test the usability on our web UI. We randomly chose 10 people and let them navigate through the website without any explanation.

On our first survey, 4 out of 10 people found our website confusing. This was because even when the users changed the settings of the greenhouse, they were not able to confirm whether the change was saved or not. Because we were aiming to have 80% ease-of-use, we failed on the first survey.

After improving this issue, we gave out the survey again to the same 10 people. As a result, 10 out of 10 people answered that it was much easier to understand the functionality of the website, and therefore they could flawlessly navigate through. Therefore, we eventually succeeded in this test by having 100% positive feedback.

D. *Results for Heating and Watering System*

We tested the heating and watering system of our greenhouse to make sure that we met our previously set requirements. When the current greenhouse temperature was 74°F, we set the goal temperature to 80°F. The greenhouse internal temperature reached 84°F in 10 minutes, and the heater automatically turned off after reaching 84°F. This testing result satisfied our requirements of maintaining the target temperature with a 5°F tolerance, and reaching the new target within an hour. The heater that we are using in our greenhouse is effective enough to raise the internal temperature of the greenhouse 1°F per minute.

In order to test the greenhouse's watering system, we set the target soil moisture percentage higher and lower than current sensor value. The water pump turned on when the target was set higher than the current sensor value, and stayed off when the target was set lower than the current sensor value.

IX. PROJECT MANAGEMENT

A. *Schedule*

Overall, we were on time with our schedule. Kanon built the overall web application, including the login and registration page, the greenhouse controls page, and the setup for the DynamoDB. Hiroko was in charge of the hardware; she connected the sensors to the ESP32, as well as the appliances that control the greenhouse environment to the ESP32. Sarah implemented the HSV Color Detection and edge detection, and applied that to the growth stage classifier and withering detection. For specifics of the schedule, please refer to Appendix I: Schedule Chart.

As we researched more about greenhouses and plants, we made some changes to the schedule. After deciding on pea shoots as our testing subject, we updated the schedule to include when to start planting pea shoots for our tests. We gave the pea shoots a week head start before the testing period, because a week is needed for their germination.

We also had to factor in when Sarah would be bringing the Raspberry Pi camera and Raspberry Pi to Pittsburgh, so we pushed back the CV integration into the physical greenhouse and live stream work until her arrival.

B. *Team Member Responsibilities*

We divided the work such that Hiroko worked on most of

the hardware, specifically arranging the relays, setting up a feedback loop with the sensors, the MQTT protocol connection between the ESP32 and the website, and the assembling of the sensors, greenhouse, and equipment. Kanon worked on the cloud database setup with AWS, integrating DynamoDB and Twilio with the website, managing the data received and sent between the hardwares and software, and creating a web platform with Django for users to interact with the greenhouse, and assisting in the hardware setup. Sarah focused on creating a remote CV application and was responsible for setting up the day and night vision RPi Camera with the RPi and implementing analysis on recognizing growth stages, defects such as withering, diseases, or deforms, and extreme vine curvature that requires plant staking. She also completed a 24/7 live stream of the greenhouse and a SMS notification system when attention is needed on the plants.

C. Budget

The Bill of Materials is located at the bottom of the report, in Appendix II: Budget and Parts List. With Sarah not being in the same location as Hiroko and Kanan, we decided that it would be best to work with several boards instead of integrating all the hardware together, so Sarah's CV implementation will be integrated to the RPi and will work remotely from the ESP32 system. In the Bill of Materials, the first column contains the web development service and API credits and all the hardware components bought the first week after the project proposal. The second column contains the AWS services that have prices dependent on how much of the service we use, so we estimated how much we would use these web development tools and gave enough room in the remaining AWS credit for any changes in the future.

D. AWS Credits

Our main use for the AWS credits was for DynamoDB. With the total write/read within the database, we consumed $0.62 out of our $50 credits. We would like to thank the ECE department and professors for giving us an opportunity to learn more about AWS products with these credits.

E. Risk Management

The risks we had in mind changed considerably between when we pitched our idea and when we began implementing the components and designing the project. For the hardware, we were concerned about the response time being too low for the communication between the hardware, cloud, and software, the feedback loop appropriately adjusting conditions without overshooting or taking too much time, and placing hardware such that it will all fit in the greenhouse while being safely away from the water and organic material we are working with, and the instability of the connection between hardware and software. Plans for mitigating such challenges included setting a threshold for the frequency of the feedback loop updates and changes and putting the hardware in a container. For the software, we were aware of the potential issues with the latency of the live stream monitoring, the lag in the website, the proper automation of multiple plant types, overlapping leaves and plants that may hinder CV analysis of defects, and high measures of false negatives for classifying growth stages and defects. To mitigate this, we chose the RPi to run the CV application and live streaming script as it has the computing power to run both components and in the future we may add more cameras or moving cameras to get all angles of the plant.

Further into the project, we also had to consider the risk of receiving the wrong data from the database, and to mitigate this issue we planned to either regain the data before outputting the value to the website or to notify the website if drastic changes take place. We also hoped that the night vision of the IR-Cut cameras would work properly, but in the case that it did not, we planned to reconfigure the LED lights to turn on for the night vision to work. After testing the OpenCV module, we found it very important that the subject we are analyzing contrasts with the background or the unnecessary components of an image, so we looked into making a monochrome backdrop in the greenhouse that provides the best contrast. We also discovered that parts of the soil moisture sensor were fragile and not waterproof, so Hiroko used heat shrink tubes to cover the sensor's outer electrical components.

X. Ethical Issues

Our automatic greenhouse seeks to help people grow plants at home, since most greenhouse technologies that are available on the market are for industrial uses and require a lot of investment in infrastructure and technical expertise to operate. An ideal user would be a home gardener who needs help growing plants, but there are users that may be vulnerable to failure or misapplication of our project. There is a possibility that someone might try to use the greenhouse to grow illegal plants, even though we do not intend on encouraging illegal activities. One might try to grow plants that can be used as ingredients for illegal drugs such as cannabis or opium poppy, or grow illegal or restricted plants classified as invasive species that can threaten the ecosystem of the region.

In order to prevent this kind of misapplication of our greenhouse, it may be helpful to create a instruction manual that contains recommendations for ideal plants to grow using the greenhouse, as well as warnings that give a list of illegal plants to grow and explain the consequences of growing them. Furthermore, it may be useful to implement a plant detection algorithm for identifying common illegal plants using the already existing CV detection system and send alerts or warnings when it detects the growth of illegal plants. However, it is ultimately the user's decision and responsibility if they decide to grow illegal plants using the greenhouse. Therefore, there will always be a small possibility that a user may misuse the greenhouse no matter how many preventative measures are taken.

XI. Related Work

Agricultural technology is one of the most rapidly

developing fields, and we took some inspiration from the industrial greenhouses that are already automated at large farms. The standard sensing systems for these greenhouses are LED lights, hydrometers, and thermometers, so we decided to include these components to our project and include a heating system, irrigation system, and lighting system to adjust such environmental parameters. There are many smaller projects online that use sensors compatible to the Raspberry Pi, but none of them seem to implement lighting, watering, and temperature altogether, so we hoped to scale down the size of industrial greenhouses, combine all the sensors and equipment through an ESP32 instead, and create a cheaper, smarter system with convenient control through a website and CV analysis.

XII. Summary

Our team has been successful in completing the deliverables on time, and creating detailed, thorough designs for our subsystems. We also had enough room in our budget to attend to any unexpected issues and risk mitigation.

A. Future work

Beyond the end of the course, we hope to scale the types of plants the greenhouse accounts for, so ordering a larger greenhouse and more sensors and equipment are in the bigger picture. Adding a camera that moves around the greenhouse and could take both side and top views of a plant would also make our CV analysis much more accurate.

Due to financial and technical constraints, we could not include an air cooler, but if the greenhouse were to be used in a more tropical environment, an air cooling system would be necessary. Depending on the future scale of the greenhouse, we may need more compact and customized containers for the hardware, especially if the greenhouse is subject to high levels of humidity and moisture. A customized PCB board could also simplify the wiring and complications that come with breadboards.

Some of our members hope to use the systems in the greenhouse to grow plants in an urban setting, and depending on the living situation the systems may be adjusted to run without an enclosed greenhouse.

B. Lessons Learned

Through this project, we learned how to set ambitious yet realistic goals and how to collaborate and take time out of our schedules in order to complete a complex project. We found that clear communication and honesty about our progress in the project prevented time wasting and misunderstanding when we would coordinate tasks. In doing so, we found more time to research and implement our individual tasks. Unlike some Capstone experiences, we did not feel rushed to finish our project at the end of the semester because the amount of time spent on it was evenly distributed throughout the semester. Always showing up to meetings, maximizing productivity during the allotted Capstone time, and asking well-developed questions to the course staff are a few of the habits that we developed early on in the semester that helped our team work efficiently.

Glossary of Acronyms

ADC – Analog to Digital Converter
AWS – Amazon Web Services
CV – Computer Vision
GPIO – General-Purpose Input/Output
HSV – Hue Saturation Value
IR-Cut – Infrared Cut
LED – Light Emitting Diode
MQTT – Message Queuing Telemetry Transport
OBD – On-Board Diagnostics
PAR – Photosynthetically Active Radiation
RPi – Raspberry Pi
UI – User Interface

References

[1] Aufranc, Jean-Luc. “Know the Differences between Raspberry Pi, Arduino, and ESP8266/ESP32.” *Embedded Systems News*, CNX Software, 24 Mar. 2020, www.cnx-software.com/2020/03/24/know-the-differences-between-raspberry-pi-arduino-and-esp8266-esp32/.

[2] Badgery-Parker, Jeremy. “Light in the Greenhouse.” NSW Agriculture, *Agnote*, Sept. 1999, https://www.dpi.nsw.gov.au/__data/assets/pdf_file/0007/119365/light-in-greenhouse.pdf.

[3] “BH1750 (GY-302), measure the lighting quality of your home (Arduino / ESP8266 / ESP32).” diyprojects.io, 12 Feb, 2021, https://diyprojects.io/bh1750-gy-302-measure-lighting-quality-home-arduino-esp8266-esp32/#.YJ58iBNKj6a.

[4] Campbell, Scott. “TURN ANY APPLIANCE INTO A SMART DEVICE WITH AN ARDUINO CONTROLLED POWER OUTLET.” Circuit Basics, https://www.circuitbasics.com/build-an-arduino-controlled-power-outlet/.

[5] Cat, Invisible. “DIY a Webcam for MacBook with Raspberry Pi [Tutorial].” *Medium*, Medium, 19 July 2020, www.medium.com/@invisiblecat233/diy-a-webcam-for-macbook-with-raspberry-pi-tutorial-2efc8213d9e7.

[6] Chris. “Image Background Removal Using OpenCV.” *Medium*, Medium, 21 June 2020, www.chris-s-park.medium.com/image-background-removal-using-opencv-part-1-da3695ac66b6.

[7] Datta, Sumon, et al. “Understanding Soil Water Content and Thresholds for Irrigation Management.” Oklahoma State University, Aug. 2018, https://extension.okstate.edu/fact-sheets/understanding-soil-water-content-and-thresholds-for-irrigation-management.html.

[8] David, Christopher. “Soil Moisture Sensor Tutorial for Arduino, ESP8266 and ESP32.” DiyI0t, https://diyi0t.com/soil-moisture-sensor-tutorial-for-arduino-and-esp8266/.

[9] Hassan, Murtaza, director. *LEARN OPENCV in 3 HOURS with Python | Including 3xProjects | Computer Vision*. *Murtaza's Workshop*, YouTube, 25 Mar. 2020, www.youtube.com/watch?v=WQeoO7MI0Bs.

[10] Hattersley, Lucy. *Monitor Plant Growth with AI and OpenCV*. The MagPi Magazine, www.magpi.raspberrypi.org/articles/monitor-plant-growth-ai-opencv.

[11] Miles, Carol A, et al. “Pea Shoots.” Pacific Northwest Extension, http://pubs.cahnrs.wsu.edu/publications/wp-content/uploads/sites/2/publications/PNW567.pdf.

[12] nathancynathancy. “How to Get the Size of an Object Using OpenCV Python?” *Questions*, Stack Overflow, 11 Feb. 2020, www.stackoverflow.com/questions/60171143/how-to-get-the-size-of-an-object-using-opencv-python.

[13] Pennisi, Svoboda Vladimirova, and Robert Westerfield. “Care of Holiday and Gift Plants.” University of Georgia Extension, 1 June 2006, https://extension.uga.edu/publications/detail.html?number=B1318&title=Growing%20Indoor%20Plants%20with%20Success.

[14] Rizza, Matteo. *Greenhouse Monitoring with Discovery Kit IoT and Android*. Hackster.io, 6 June 2019, www.hackster.io/matteo-rizza/greenhouse-monitoring-with-discovery-kit-iot-and-android-333430.

[15] Rosebrock, Adrian. “Measuring Size of Objects in an Image with OpenCV.” *Tutorials*, PyImageSearch, 28 Mar. 2016, www.pyimagesearch.com/2016/03/28/measuring-size-of-objects-in-an-image-with-opencv/.

[16] Santos, Rui. “ESP32 DS18B20 Temperature Sensor with Arduino IDE (Single, Multiple, Web Server).” Random Nerd Tutorials, https://randomnerdtutorials.com/esp32-ds18b20-temperature-arduino-ide/.

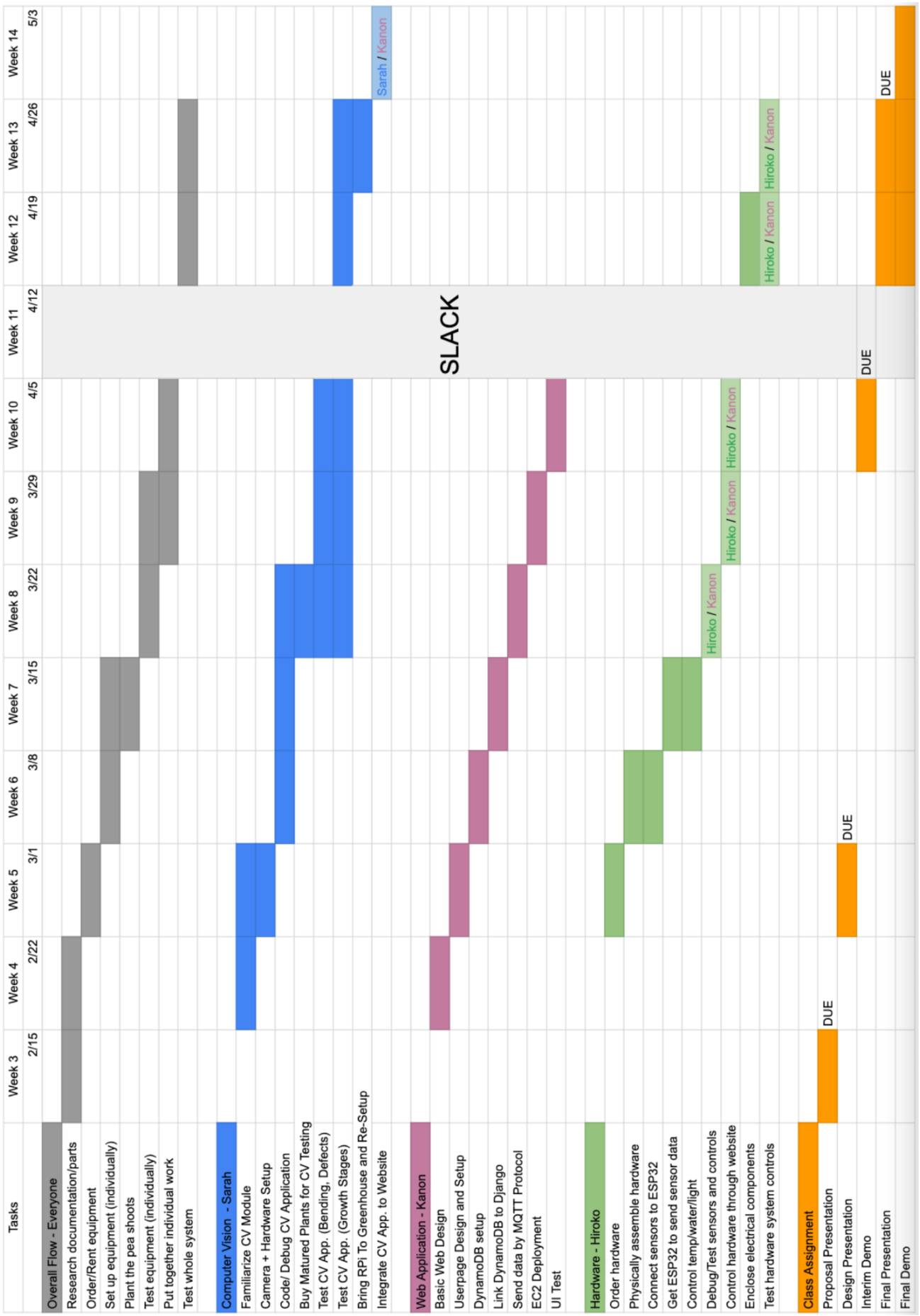
[17] “The Twilio Python Helper Library.” Twilio Docs, www.twilio.com/docs/libraries/python.

[18] “Types of Agricultural Water Use.” Centers for Disease Control and Prevention, www.cdc.gov/healthywater/other/agricultural/types.html.

[19] “Video Streaming Raspberry Pi Camera.” *Courses*, Random Nerd Tutorials, 1 Nov. 2019, www.randomnerdtutorials.com/video-streaming-with-raspberry-pi-camera/.

[20] Zara, Moheeb, “Building an AWS IoT Core device using AWS Serverless and an ESP32.” AWS Compute Blog, 3 Jan. 2020, https://aws.amazon.com/blogs/compute/building-an-aws-iot-core-device-using-aws-serverless-and-an-esp32/.

Appendix I: Schedule Chart



Appendix II: Budget and Parts List

Component	Price	Quantity	Total Price	Source
AWS Credit	$50.00			18-500 lab
Twilio Notification	$0.0075 / send		$1.105	Twilio
Greenhouse Shelf	$38.99	1	$38.99	Amazon
LED Grow Light (2 ft)	$44.85	1	$44.85	Home Depot
Zip Ties	$5.49	1	$5.49	Amazon
Planter Tray w/ Soil (2 Pack)	$34.99	2	$69.98	Amazon
Pea Shoot Seeds	$15.99	2	$31.98	Amazon
Water Tank (2.5 gallon)	$9.06	1	$9.06	Amazon
Mini Heater	$19.99	1	$19.99	Amazon
ESP32 (3 boards)	$16.99	1	$16.99	Amazon
Irrigation DIY Kit	$30.99	1	$30.99	Amazon
Breadboard (3 boards)	$0.00	1	$0.00	18-500 lab
Temperature Sensor (18B20)	$12.49	1	$12.49	Amazon
Light Intensity Sensor (BH1750)	$6.69	1	$6.69	Amazon
Extension Cord (8 ft)	$25.99	1	$25.99	Amazon
Micro USB to USB (10 ft)	$13.99	1	$13.99	Amazon
Scratch Awl	$3.99	1	$3.99	Amazon
Wire Cutter	$9.59	1	$9.59	Amazon
Outlet Box	$5.26	2	$10.52	Amazon
Duplex Receptacle	$1.29	2	$2.58	Amazon
Duplex Receptacle Wallplate	$2.98	2	$5.96	Amazon
NM/SE Clamp Type Connector	$3.99	2	$7.98	Amazon
Appliance Cord	$7.80	2	$15.60	Amazon
5V Relays	$8.49	1	$8.49	Amazon
Raspberry Pi IR-Cut Camera Day & Night Vision	$24.99	1	$24.99	Amazon
Raspberry Pi 3 Model B V1.2	$41.44	1	$41.44	Amazon
Raspberry Pi 3 Model B V1.2	$0.00	1	$0.00	18-500 lab
RF Transmitter and Receiver	$0.00	1	$0.00	18-500 lab
Raspberry Pi Power Adapter	$14.90	1	$14.90	Amazon
Junction Box	$23.75	1	$23.75	Amazon
Soil Moisture Sensor	$8.31	1	$8.31	Amazon
Snapdragon and Seed Germanium Flow	$6.38	1	$6.38	Home Depot
Flower Pots	$3.19	1	$3.19	Home Depot
Budget			$600.00	
Total Cost			$516.26	
Remaining Balance			$83.74	

AWS Component	AWS Price	Quantity	AWS Total Price
AWS DynamoDB	$1.5 / 1 million write+read	~0.5 million write+read	$0.62
AWS Credit			$50.00
AWS Total Cost			$0.62
Remaining AWS Credit			$49.38

Tool	Purpose
Arduino IDE	Program ESP32
BH1750 Sensor Library	Read data from light intensity sensor
One Wire, Dallas Temperature Library	Read data from temperature sensor
AWS IoT Core	Communicate with ESP32
AWS EC2	Cloud solution
AWS DynamoDB	Database
Twilio	Notification system
OpenCV Library	Plant detection