Robotany – Our Fourth Rodeo

Eleanor Ozer, Jason Ashley, Noelle Law, and Zachary Hicks

December 10th, 2020

Capstone Design ECE 4440 / ECE4991

Signatures

[illegible]

Noelle Law

[illegible]

[illegible]

Statement of work:

Eleanor Ozer

My tasks focused on the software side of the project, ranging in nature from designing and developing the mobile application, connecting the front-end mobile app UI to the server-side back-end through API endpoints, and helping reverse-engineer and reconstruct the OV2640 ArduCam driver on the ESP32. The majority of the semester was spent building the mobile application, which was built from scratch to become a commercial-capable app with professional formfactor. The mobile application was programmed in JavaScript using the React Native web-app framework and shipped using Expo; having never programmed in JavaScript, a significant segment of time was dedicated to learn and understand the language's lifecycle and syntax. Both tools were approachable and extremely capable: getting the application off-the-ground and general construction did not present a sizeable challenge.

Competent mobile-application authentication was critical to ensure a secure product and protect user data, and is where the primary hardship was faced whilst building the app. We used an industry-standard strategy to secure authentication, which utilized hashing to send user data and tokens to-and-from the database, and prescribed a complex API and navigation configuration. Once completed, time was spent to thoroughly test and ensure that the application was behaving as intended.

Noelle Law

The pieces I have contributed to have been twofold: the control algorithm and computer vision algorithms. The control algorithm was designed in Code Composer Studio to integrate the plant signals, light sensor signals, bump switches, and IR distance sensors through a task scheduler. The C code would use the bioelectrochemical signals produced by the plant as a 'wake-up' for the controller and the difference in signals from the two light sensors were analyzed to identify and moved towards an optimally lighted position. The computer vision algorithms included image background and glare removal and machine learning models to track plant growth. The image background and glare removal algorithms were written in Python. They were then packaged up as functions to be placed on the server for use in the application. A neural network architecture for image classification was designed using OpenCV and TensorFlow, it was then trained on two separate datasets for the following classification purposes: if electrodes can or cannot be placed on the plant and if the leaves of the plant can or cannot be harvested. These models are placed into the server and enable the user to be notified of when important growth indexes have taken place.

Beyond this, I helped with writing test programs to ensure there is proper communication between the MSP432 and ESP32-WROOM so that needed data could be sent between the two microcontrollers via UART communication. I also spent some time trying to get a minimized version of OpenCV on the ESP32-WROOM before it was determined that the computer vision algorithms would be better placed on the server, due to space constraints of the microcontroller.

Jason Ashley

My work focused on the ESP32, API, and mobile application software. On the ESP32 side of the software, I developed the application for ESP-IDF based on FreeRTOS that initializes the camera, captures images, and reads the moisture sensor. This application was also responsible for setting up the wireless communications with the ESP-IDF SoftAP provisioning library, initializing the NVRAM when necessary, and synchronizing the clock to ensure communication may be performed securely over HTTPS. Beyond this, the provided camera driver for the selected camera module had no driver for the chosen platform, so I worked with Eleanor to reverse-engineer the provided driver library for the module, determine needed functionality, and port that to our application. I also developed some methods for making the driver more CPU and storage efficient (at the cost of some unused RAM), speeding up the driver significantly. With all this complete, an image and moisture data could be securely uploaded to some location after a user initializes the device on their account. Thus, I then needed a location to send this data.

The API I developed has both a user and robot facing section. The user side allows a user to securely create an account and login, register their mobile phone for notifications, add a plant to their account, and view details about the plant, including previous images and moisture sensor readings. I also created event handlers for, in certain circumstances, notifying the user to attend to their plant. This includes in the cases where it might need to be rotated, water is running low, or one of our machine learning models determines it is ready to be harvested or electrodes need to be attached. All information is stored in a secured database. Users are limited to only details they can access thanks to authentication provided by information in this database, including tracked ownership relations, session storage, and login credentials.

The robot facing side of the API was developed to allow the robot to upload images and sensor readings using a unique identifier on the device in a secure manner. At a regular frequency when the Wi-Fi network is available, the robot will send an image and moisture reading. The API call instantly responds, but asynchronously saves this image, processes it through Noelle's packaged computer vision and machine learning models, then uploads it to a secure storage location. Accessing these processed images now becomes impossible without the API first signing the URL, granting temporary access to the resource for the mobile application, whenever it properly requests it.

Zach Hicks

The work that I contributed to this project includes the hardware design, the CAD design for the 3D printed parts, and writing the firmware drivers for the light sensors and the ADC channels for the plant signals.

My first task was to create the hardware schematics for the analog signal processing hardware, MSP432 sensor connections, and ESP32 sensor connections. These tasks included designing the instrumentation amplifiers and filters to process the plant signals, ensuring that the ESP32 pins were correctly configured to allow for programming and operation, and configuring the hardware for the communications busses as well. Once the schematics were finished, I completed the original PCB layout and the necessary revisions as well. The analog signal processing work that I completed also involved the background research on the plant signals themselves, including creating a waveform used to simulate the mimosa action potentials

I also completed all of the necessary CAD to design the 3D printed parts used on the Robotany. These parts include the front and rear enclosures, the plant pot holder and light sensor mounts, and the post to attach the camera as well. These parts were designed using Fusion 360, which was a new software package for me this semester so some time was spend familiarizing myself with it as well.

The final major task that I completed was writing the firmware to integrate the light sensor modules with the MSP432. I first wrote and tested an I2C driver on two of the MSP432 eUSCI modules, and then implemented the necessary functions to calibrate and retrieve the light data from the sensors.

Table of Contents

Contents

Table of Figures

Table of Tables

Abstract

The Robotany is an environmentally-aware, autonomous plant-hybrid robot that tracks moisture levels, monitors growth, and moves according to the plant's lighting needs. The project uses a camera module to collect image data needed to identify both when a user is able to attach electrodes to the plant and when a plant is ready to harvest, making the Robotany ideal for growing herbs such as basil or mint. Electrodes attached to leaves of the plant monitor the bioelectrochemical signals of the plant, which will act as a "wake-up" for the system, signaling the activation of light sensors on opposite sides of the plant to find and move the Robotany to more optimal lighting conditions. Two motors, an array of bump switches, and cliff sensors are attached to the base of the system to allow it to maneuver safely around any surface. A mobile application allows the user to monitor and be notified of moisture, growth, and lighting levels. The Robotany provides the groundwork for many future plant-based sensing applications because it utilizes the biosensor qualities of the plant to directly monitor plant health and changes in the environment, proving to be a more efficient sensor in the natural world.

Background

Over the past decade, starting with an increase in the popularity of succulents in 2012, society's youth has grown an inordinate love for potted plants [1]. Stemming from a desire to nurture, young plant owners become emotionally attached: often naming their plants, caring for them intimately, and even treating them as conversational counterparts. It should be noted, however, that plants cannot speak.

The Robotany, a dual-motor smart-pot, was conceived to mitigate that issue: to give the common houseplant a voice. Equipped with a moisture sensor and camera, the device allows the plant to notify the user (via mobile application) when it should be watered and update the user on its growth. Additionally, two electrodes attached to the plant can measure certain plants' bioelectrochemical signals and give the plant its own free-will to move around as desired.

When a plant is exposed to variation of temperature or light, drought, soil pollution, or any other perturbation, the plant produces electrical potentials to communicate between plant tissues and organs plants [2]. These potentials, known as bioelectrochemical signals, resemble nerve impulses in humans and animals, and can be measured using electrodes [3]. These signals are relatively weak, typically in the range of 1-50 mV, depending on the stimulus and the plant [4]. A study done by Cai & Qi, researchers from Zhejiang University in Ningbo, China, consisted of the design of a two-stage amplifier for these signals using integrated op-amps, amplifying a differential voltage similar to the function of an instrumentation amplifier [4]. A product called the Plant SpikerBox, produced by Backyard Brains, a neuroscience education company, allows users to connect electrodes to a plant and measure the bioelectrochemical signals using a mobile application [5]. The Plant SpikerBox uses an ATMEGA32U4 MCU and a connection to a smartphone process and display the signals from the plant.

The primary motive for measuring these signals has been to develop environmental biosensors, but several projects have given plants their own free-will: most notably the Elowan. This project, developed at MIT, positioned a potted Homalomena plant on a two-wheel drive robot and allowed the plant's bioelectrochemical excitations to control the onboard motors [6]. When placed between a lit and unlit lamp, the Elowan was able to move towards the lit lamp within 120 to 180 seconds. The phytosensing and phytoactuating system produced by Cybertronica Research is another example of a project that measures and records electrical responses of plants to an external stimulus [7]. This product is equipped with sensors to measure signals from the plant, and actuators to perform a defined action as a result of the measured electrochemical activity from the plant.

Similar to the Elowan, the Robotany is equipped to use a specific plant's signaling to move the plant to a location with more sunlight - in effect, the plant's leaves can be used as photoelectric biosensors. Unlike the Elowan, the Robotany was developed as a consumer-facing product instead of a laboratory experiment. Thus, significant time was spent to ensure that the user experience is as pleasant as possible. Additionally, safety precautions, such as cliff sensors and bump sensors, were implemented to prevent the plant from driving itself off a high ledge or into obstacles.

Although several customer-facing "smart garden" products currently exist on the market, we believe the integration of bioelectrochemical measurement makes this project novel. Most "smart garden" products have some implementation of autonomy, such as hydroponics or artificial light. The Robotany, however, lacks automation, a design choice made to help foster the relationship between plant and owner. The Robotany does not make the plant self-sufficient, it simply provides the user an improved communication structure with the plant.

To measure the bioelectrochemical signals inside the plant, small signal amplification and external noise removal were necessary, which necessitated the signal processing techniques learned in ECE Fundamentals III, ECE 3750, taken by all four groupmates. Noelle, Zach, and Eleanor have two semesters of experience with the robot platform used to move the plant, the TI-RSLK MAX, and the microcontroller, an MSP432, which were both used in the Embedded and Robotics course sequence, ECE 3501 and ECE 3502. Jason has prior experience with the MSP430 series through the Introduction to Embedded Computing Systems course, ECE 3430.

The printed circuit board and other necessary CAD were designed primarily by Zach, who has experience from his projects in Gizmologists Engineering Club and in the ECE Fundamentals classes. A mobile application was built as the user interface, stemming from Eleanor's app development knowledge from Advanced Software Development, CS 3240, and Jason's from Advanced Software Development, CS 3240, and Mobile Application Development, CS 4720. An ESP32 Wi-Fi/Bluetooth module will be used for communication from the robot to the app, leaning on Jason's work in personal projects involving the particular hardware that is planned to be used.

Noelle's primary task was the control algorithm that allows the plant's positioning to be updated according to its lighting needs. She used her experience in the Autonomous Mobile Robots course, ECE 6501, to build out a control algorithm. Further, cliff sensors and bump sensors will be integrated into the algorithm, having a greater priority than changes in lighting to ensure that the robot is able to safely maneuver any surface it is on. Determining interrupt priorities and integrating multiple sensors to determine the system's movements will pull from coursework in Embedded Computing and Robotics, ECE 3501 and ECE 3502.

Constraints

Design Constraints

Two constraints were placed on the design of the Robotany because of the hybrid composition of both electrical and computer engineers on the team. As such, the course required that a team-designed, custom Printed Circuit Board (PCB) be included in the system in addition to either a microcontroller or a National Instruments myRIO [8].

CPU Limitations

Two microcontrollers were chosen for this project: the MSP432P401R [9] and the ESP32-WROOM [10]. The MSP432 was chosen as a CPU due to the large number of GPIO pins, especially as it pertains to ADC channels. The ESP32 was selected due to its Wi-Fi capabilities, community support, and low cost. However, it required the use of an FTDI programmer [11] to allow for debugging support and could only be used on a Linux machine. The ESP32 and the MSP432 constrained the team to a 40MHz clock speed and a 48MHz clock speed, respectively. While these clock speeds limited the speed of peripheral clocks, it proved to be more than sufficient for the system.

Software Availability

UVA provides an active license to National Instruments' Multisim [12] and Ultiboard [13], and as such, these tools were employed in board design. Embedded code was written with Code Composer Studio (CCS) [14] for the MSP432 and with PlatformIO [15] for the ESP32, due to their respective compatibility with the microcontroller. The aforementioned development environments were additionally selected due to their availability at no cost.

Manufacturing Limitations

The PCB manufacturer, Advanced Circuits, imposed manufacturing constraints to meet the criteria required for a student special on 2-layer boards. The 2-layer boards had the following requirements: 62 mil thickness and a 39-mil core. The most significant constraints to system design were as follows:

- Maximum board size: 30 in^2
- Minimum 5 mil line per space
- Minimum 10 mil drill hole size
- Maximum 50 drilled holes per in^2

Beyond these constraints the TI-RSLK MAX robot base [16] limited the width of the PCB to the width of the gap between the motors. As such, the PCB was designed to match the size of the MSP432 to cleanly fit on top of the microcontroller and connect to the robot base.

Economic and Cost Constraints

Our Fourth Rodeo was provided a budget of $500 to complete the project. As such, multiple cameras or a more advanced camera could not be purchased for the system. Beyond this, it limited the ability for the team to purchase back up items or multiples of the same item, such as microcontrollers, to be used for synchronous development. The impact of these constraints was somewhat limited by virtue of equipment donations from other professors in the ECE department.

External Standards

1. Wireless communications on the robot adhere to IEEE 802.11 standards for local area networks. Most specifically, the ESP32 utilizes 802.11b, 802.11g, and 802.11n for Wi-Fi based networking. The ESP32 is certified with the Wi-Fi Alliance for such interoperability of communications [17]. The FCC has also granted the ESP32 a certification for both Wi-Fi and Bluetooth communications [18].
2. The PCB was designed in adherence to IPC standards. Standards for track and part spacings are determined by IPC-2221A [19]. The acceptance criteria of the PCB are set by IPC-A-600J, which includes, but is not limited to, material, board edges, solder mask, and plating [20].
3. Surface Mount Device (SMD) components have standard sizes as defined by the Surface Mount Technology (SMT) industry standards. JEDEC [21] is the leading standardization group, which simplified the PCB design because these footprints are readily available in the Multisim master database. The board uses the 1206, 0805, and 0402 rectangular passive components.
4. The Universal Asynchronous Receiver-Transmitter (UART) circuitry block is responsible for serial communication implementation. UART transmits data via asynchronous communication between devices, a transmitter and receiver [22]. This allowed information between the MSP432 and ESP32 to be sent between devices at a standard baud rate of 115,200 bits/second. This also allowed for communication between a serial terminal and the MSP432 for debugging capabilities.
5. The Serial Peripheral Interface (SPI) communications protocol is used for synchronous communication between a master device and its selected peripheral [23]. SPI was employed for image data communication between the OV2406 camera and the ESP32 microcontroller.
6. The Inter-Integrated Circuit (I2C) communications protocol is used for synchronous communication between a master device and its selected peripheral [24]. I2C was employed for communications between the VEML7700 light sensors and the MSP432, and the interface for sending commands between the ESP32 and the camera.

Tools Employed

Various tools were used to complete this project, specifically as it pertains to programming, design, and testing. Tools for each category are described in further detail below.

Hardware

National Instruments' simulation and design tools, Multisim [12] and Ultiboard [13], were used for board design and layout. Zach was familiar with both tools due to prior coursework in the ECE curriculum. However, he improved his skills throughout the course of the project to meet the level of proficiencies required for both the creation of the schematics and the layout of the board. Autodesk Fusion 360 [25] and Ulitmaker Cura [26] were used to design and 3D print the necessary parts. The hardware test plan was conducted using the National Instruments Virtual Bench [27] and the Analog Discovery 2 [28].

Firmware

Firmware for the MSP432 [9] was written in C using Code Composer Studio (CCS), Texas Instruments' integrated development environment [14]. The developed MSP432 code used certain functions and header files provided with the TI-RSLK MAX learning kit [16], which was developed by Dr. John Valvano for the TI-RSLK MAX education curriculum [29].

Firmware for the ESP32-WROOM was developed using PlatformIO [15] and the ESP-IDF [30], which is Espressif's C-language IoT development framework. The IDE employed for ESP32 code development was Visual Studio Code [31]. An FTDI programmer was used for debugging ESP32 firmware [11].

Software

Python 3.7 was installed using an Anaconda Distribution [32]. The algorithms for the computer vision system were prototyped in Jupyter Notebook and packaged for deployment on the server [33]. The following packages were installed to using pip [34]:

- OpenCV, to process image data [35]
- TensorFlow, for neural network creation [36]
- Numpy, for powerful n-dimension array use [37]

Beyond these packages, a structured-forest edge detection model was used to aid in the background removal algorithm [38].

The mobile application was written in JavaScript [39] using the React Native [40] development framework, which was chosen over similar web application development frameworks because it incorporates native components and cross-compatibility between iOS, Android, and the web, so that a single app can be developed that supports all three platforms. Expo [41], a tool designed to be used in conjunction with React Native, was also used to facilitate development by providing a framework and bundler for deployment and rapid testing the application on iOS, Android, and browser-based platforms.

The application was entirely written using Visual Studio Code [31], a cross-platform and well-featured text editing and integrated development environment. Version control was maintained using Git [42] and GitHub [43], which hosted our codebase and allowed easy collaboration between the software developers.

The server was hosted on an Amazon Web Services EC2 instance [44], with photo storage on a provisioned Amazon S3 bucket [45]. The EC2 instance was responsible for running the API server as it was developed, before being run through nginx's [46] reverse proxy to ensure HTTP communications were upgraded to utilize HTTPS. Let's Encrypt [47] was used to generate the necessary certificate for these secured communications.

The API server had two major components. The first was the Node [48] application designed to process all requests made to the server. The second was the MongoDB database [49] that stored user and plant information permanently, as well as temporary session information.

The API was tested using Postman [50] to make arbitrary requests to ensure proper functionality and security. This was useful for testing partial functionality throughout the entire development process. To ensure database consistency, MongoDB Compass [51] was used to view, modify, and analyze the database to ensure consistency and proper behavior of the API calls that worked with the database.

Axios [52], an HTTP client component for React Native, was used to create async GET and POST requests to the database based on the pre-tested calls.

The proprietary OV2640 camera driver was also written in Visual Studio Code [31], using the PlatformIO [15] extension, an IDE and configuration manager across environments. The camera driver itself was written with and for the ESP-IDF [31] framework, developed by Espressif and based on FreeRTOS [53]. Esptool [54], also developed by Espressif, was used as a simple tool for testing the ESP32 protocol implementations.

ArduCAM_ESP32_UNO [55], an Arduino library for the ArduCAM camera modules for Arduino and the ESP32, heavily influenced the structure of our camera driver. This driver worked to port the major functionalities of the ArduCAM_ESP32_UNO library for the ESP-IDF with added efficiency improvements.

Ethical, Social, and Economic Concerns

The primary components used in the Robotany system are a printed circuit board, various sensors, electrical components, a power system, a plant, electrodes, and physical housing for the electronics and plant. The components for the roaming base of the system are readily available, creating a cost-effective system. The mechanical housing is be inexpensive and easy to produce. The plant, electrodes, the server, the camera modules, and the moisture and cliff sensors are also readily available and cost-effective.

The main environmental and sustainability impact will be from the power system, which is composed of rechargeable batteries. The PCBs and other electronic parts can have significant environmental impacts if improperly disposed of, so these components should be properly recycled at their respective end of life, which allows for the reclamation of copper and other harmful metals [56]. In addition, safety concerns can arise as a result of moisture data being spoofed, causing the user to be notified too often or too little when the plant is to be watered. The major ethical concern that arises from the project is that image data used for determining the growth of the plant may also contain parts of the user's household, which may pose a risk if exposed.

Environmental Impact

The rechargeable batteries pose the biggest concern to the environment because, while lithium-ion batteries are less carbon-intensive than other energy sources, they are made with toxic chemicals. The use of rechargeable batteries extends the lifetime of them by over 3 times their non-rechargeable counterparts, allowing the system to minimize harmful effects to the environment. The housing was 3D printed using PLA, a bioplastic, that has a reduced long-term impact on the environment because it is compostable [57]. The production of this type of bioplastic also decreases CO_2 emissions. Printed circuit boards can have negative environmental effects during their manufacturing, but these effects are largely unavoidable to the project team. PCBs used on the Robotany should be recycled to avoid adding harmful metals to landfills at the end of life of the device.

Sustainability

The system uses a sustainable design because the primary environmental sensor is a plant, which is both a more efficient sensor and readily available in nature. The modular design uses easily attachable electrodes, which allows the plant to be replaced by a new plant at its end of life. The modular design also enables batteries to easily be recharged or replaced, and will therefore not affect the lifetime of the system as a whole. The current design limits the system to monitor a singular plant at a time. However, the technology associated with monitoring the bioelectrochemical signals of a plant, the easily placed electrodes, and the image processing technology to identify when a plant is ready for harvest are able to be applied to larger-scale agriculture.

Health and Safety

A safety issue that arises from the system design is the potential for moisture data to be spoofed by a malicious actor. This could result in the user being notified to water the plant either too often or too infrequently, leading to the death of the plant. This potential issue will scale with the size of the system, causing more damage to be done as the amount of agriculture involved increases.

Depending on the use case, there is the potential for injury and damage related to the use of rechargeable batteries, especially over time. Employing some methods for battery maintenance could reduce this risk. If modifications to the battery system are made or there are external sources of damage, there could be a heightened potential for damage or injury. Thus, modification or damaging the battery will be strongly discouraged.

Due to the mobile nature of the system, there is a risk that a user could stumble and fall over the device if it is improperly placed on the floor in a walkway. This risk is minimized by clear instructions regarding the proper placement of the system, emphasizing use on an elevated surface, such as a desk or countertop. This minimizes the risk of stumbles and falls through safe operation. If the user does intend on placing the pot on the ground, there will be an inherent risk of tripping over the pot. Some additional indicators will be recommended for this case, with further solutions possible in later iterations of the design.

Manufacturability

The system will be relatively easy to reproduce, with the primary constraint resulting from a single printed circuit board and the custom mechanical housing for the electronics and the plant. The PCB is only two layers and does not require any advanced manufacturing techniques, and the mechanical enclosures are able to be 3D printed. For a larger-scale production, injection molding the enclosures would decrease costs and increase manufacturing speed as well. All sensors and other electronics will be available off-the-shelf, and the rechargeable batteries will be AA standard. The server for the system will have to be increased to scale for more customers, but for now the system employs a $12/month server that can be utilized by all current clients.

Ethical Issues

One ethical issue the Robotany presents is the use of image data of the plant that could showcase features of a user's home in the background. The wireless transmission could allow bad actors to expropriate image data and use it for malicious purposes. However, the system ensures that after a successful, secure request, the image data retrieved through the API will only be available for a short span of time before requiring a new signature, eliminating the risk that bad actors can extract large numbers of images. Methods will be provided for controlling if images are taken and for wiping such images from our database. To further minimize risk, image data will be taken and transmitted only once per hour.

Intellectual Property Issues

The plant growth system described in [58] consists of "a system for controlling plant growth conditions comprising [of] at least one detector, [a] central detector data processing means", and a "portable detector communication device". The patent details a mobile application used to receive data from the various sensors used within a greenhouse setting, including measurements from an irrigation system. This mobile application "may further be arranged to: receive detector data…[and] can allow a user to check detector data relating to an output or of a status of a detector in the growing area" [58]. This mobile application is similar to the one created for the Robotany in that it also allows a user to view data transmitted from sensors on the plant. However, this is not likely to pose an intellectual property conflict with the Robotany due to the differences in uses. The Robotany mobile application is used to monitor and display the health of a plant in a stylized format, rather than simply presenting data to the user.

The plant observation device and method presented in [59] outlines a system used to record and analyze images of the plant. This system uses a robot that moves among stationary plants to collect various pieces of information on the plants, including growth and light data, and transmit them wirelessly to a server. A specific independent claim of this patent similar to the Robotany is a "plant growing state measurement unit that measures the growing state of a plant", and a dependent claim is "the growth environment measurement unit further comprises of…a light quantity measuring unit for measuring the quantity of light in the measurement area" [59]. This device is similar to the Robotany in that unlike most greenhouse monitoring systems, the device moves around within the greenhouse area in order to record measurements. The Robotany differs from the claims made in this patent due to the plant being placed on the moving robot itself in order to reposition the plant location, rather than the measurement device moving around a physical location to record information on many plants. The imaging component of this system also focuses on adjusting the height of the camera in order to normalize the growth measurement based on distance between the sensor and the plant, whereas the Robotany positions the camera at a fixed height and uses a computer vision algorithm to interpret the images.

[60] describes a plant biosensor system that uses "plant biosensors that indicate the presence of contaminants". This patent consists of modifying a plant itself in order to produce a system that can monitor environmental qualities over "large and/or remote territories" [60]. The

specific claims in this patent cover the modified nucleic acid and the transgenic plant that responds to an environmental contaminant. The Robotany also seeks to use plants as biosensors, but without modifying the plant itself. The biosensing capabilities also differ between this patent and the Robotany, as the Robotany seeks to measure lighting conditions using the *Mimosa pudica*. These two patents have both different implementations and different application spaces and pose no intellectual property issues.

Based on the survey of patents completed, it is not likely that the Robotany would experience difficulty receiving a patent as a novel product.

Detailed Technical Description of Project

The proposed project was a cybernetic plant robot that both allows for increased human awareness of the plant's needs and grants a form of self-autonomy to the plant. Electrodes measure electrochemical signals induced in the plant by various environmental stimuli, and that information can be used to allow the system to "wake up". Once the system has woken up, conventional light sensor data allows the plant to move itself to an area of optimal lighting. The system also collects data on the plant using conventional sensors such as moisture sensors to notify when the human should water the plant and a camera transmitting images to notify the human when electrodes should be attached to a growing or recently harvested plant's leaves, or when leaves from an herb are ready to be harvested.

The system is built on the TI-RSLK MAX robot platform, shown below in Figure 1.

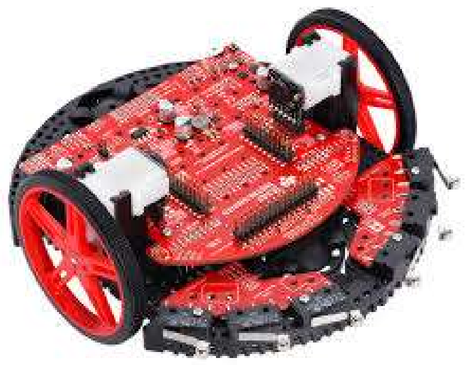


Figure 1: TI-RSLK MAX Robot Platform

The system utilizes the existing hardware on this robot platform for power management and locomotion. The necessary additional hardware and the plant itself are attached to expansion plates located above the main robot chassis. The CAD model for the robot was created using Autodesk Fusion 360 [25], and can be seen in Figure 2. The components that were designed and 3D printed include the enclosures on the front and rear of the chassis, the pot holder and light sensor mounts, and the camera mount.

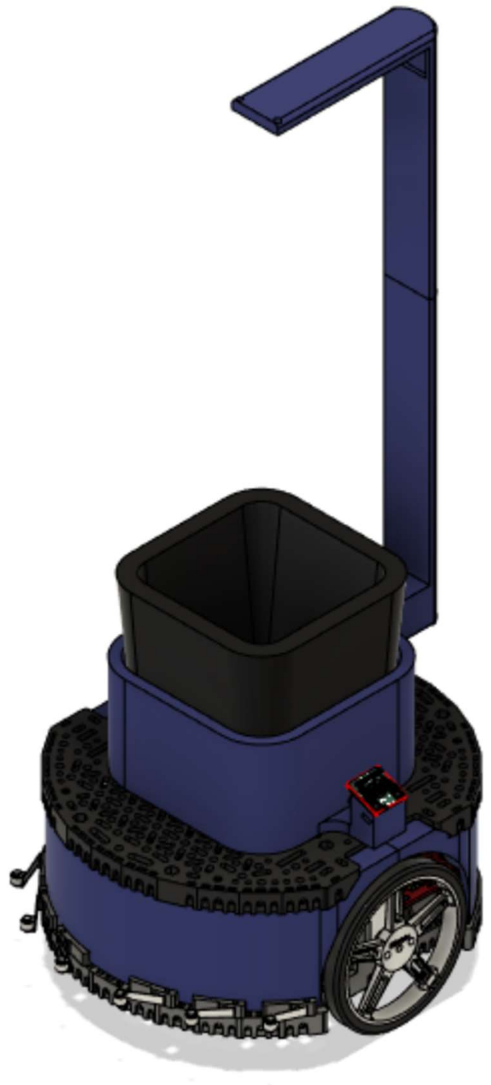


Figure 2: Robotany CAD Model

When a plant experiences a physical stimulus, such as sunlight, water, or wounding, it produces a measurable electrochemical signal. Electrodes attached to the leaves of the plant measure the electrochemical signals generated by the leaves in response to a stimulus, which then are amplified and filtered. These signals are used to "wake up" the control algorithm running on the MSP432. Data from two VEML7700 light sensors placed on opposite sides of the plant will provide light data to the microcontroller, driving the robot in the direction of greatest light. A conventional moisture sensor is used to measure soil moisture and notify the mobile application that the plant should be watered.

Various conventional sensors are used to assist the robot's autonomous movement. The TI-RSLK MAX platform has six bump switches on the front of the robot which ensure the robot does not continuously drive into a wall or other objects that would cause overload damage to the motors. Two infrared (IR) distance cliff sensors were attached to the bottom of the robot chassis to prevent the robot from driving off of a ledge in search of the strongest sunlight.

A camera module is used to capture photos of the plant to monitor plant growth over time and notify the user when to harvest mature herb leaves or when to attach electrodes to new growing leaves. The captured images and other data from the robot are transmitted to a mobile application using the Wi-Fi networking capabilities of an ESP32-WROOM module and are processed on the server. The image is first passed through a background isolation and removal algorithm. This ensures that any background of a user home or other personal space is removed from the image prior to it being stored in the database. Due to memory constraints on the ESP32-WROOM module, the images being processed will be low resolution and are captured once every hour. A glare removal algorithm is applied to the updated image and it is then passed into two separate machine learning models to classify plant growth. If a plant has reached an appropriate size for electrode placement or for harvesting purposes, the mobile application will notify the user. The neural networks for growth classification were designed using TensorFlow and trained using a dataset of 5000+ RGB images [9].

Figure 3 shows the block diagram for the overall system, where the two microcontroller units, peripheral devices, and wireless network components can be seen. Individual components of this system are described in further detail in the following subsections.

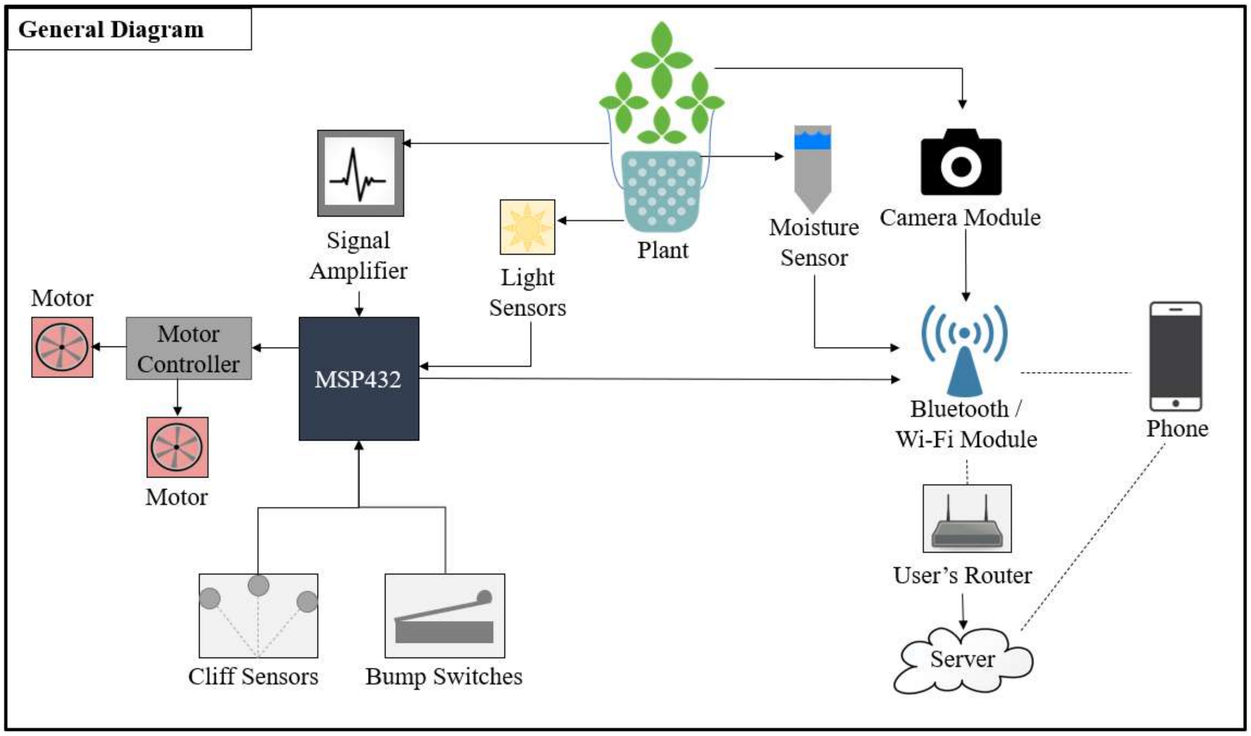


Figure 3: System Block Diagram

Hardware

The hardware for the Robotany uses a PCB designed as a booster-pack for the MSP432 LaunchPad. The PCB contains the analog signal processing hardware used for measuring the electrochemical signals generated by a Mimosa pudica plant, the ESP32-WROOM32D module used for wireless communications, and headers and passive components necessary for the various sensors used by the robot. The robot uses two microcontrollers, and the peripherals were divided between the MSP432 and the ESP32 based on their uses. Sensors used to aid the robot's navigation are connected to the MSP432, and sensors used to collect data from the plant are connected to the ESP32.

The TI-RSLK MAX robot chassis includes hardware that was integrated with the MSP432, which can be seen in Figure 4 below. The DRV8838 motor drivers [61], the motor encoders, and the bump switches interface with the MSP432 through the mounting headers on the robot chassis, so it was not necessary to redesign the hardware connections for these components. The cliff sensors are attached to two ADC channels on the MSP432, and the light sensors were connected via two I2C busses. The VEML7700 sensor modules [62] that were selected have fixed I2C addresses, so two separate busses were implemented on the MSP432 eUSCI communication modules. The hardware connections to the MSP432 microcontroller are shown in Figure 5 below.

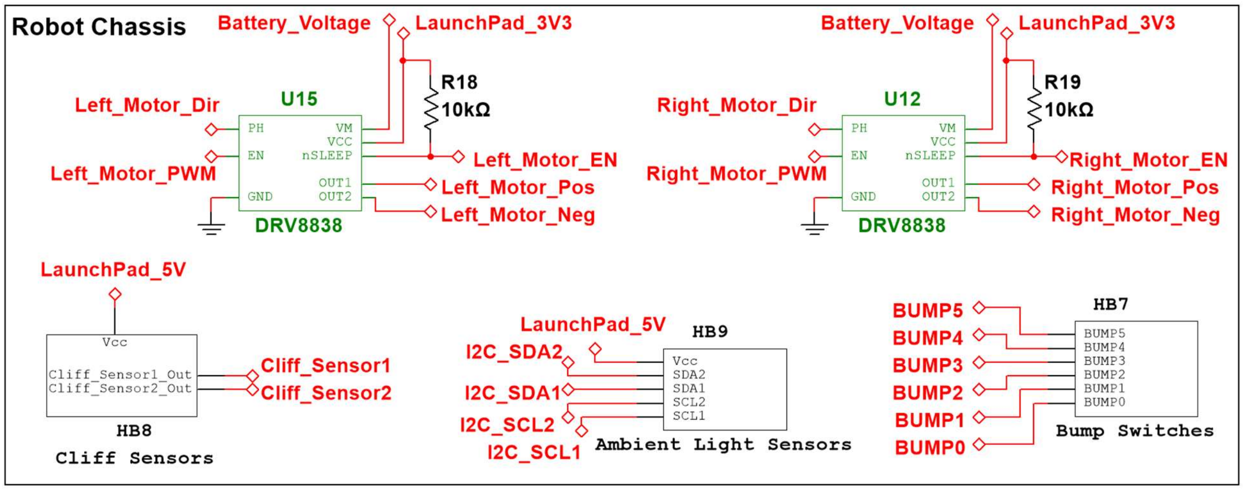


Figure 4: Robot Chassis Hardware Schematics

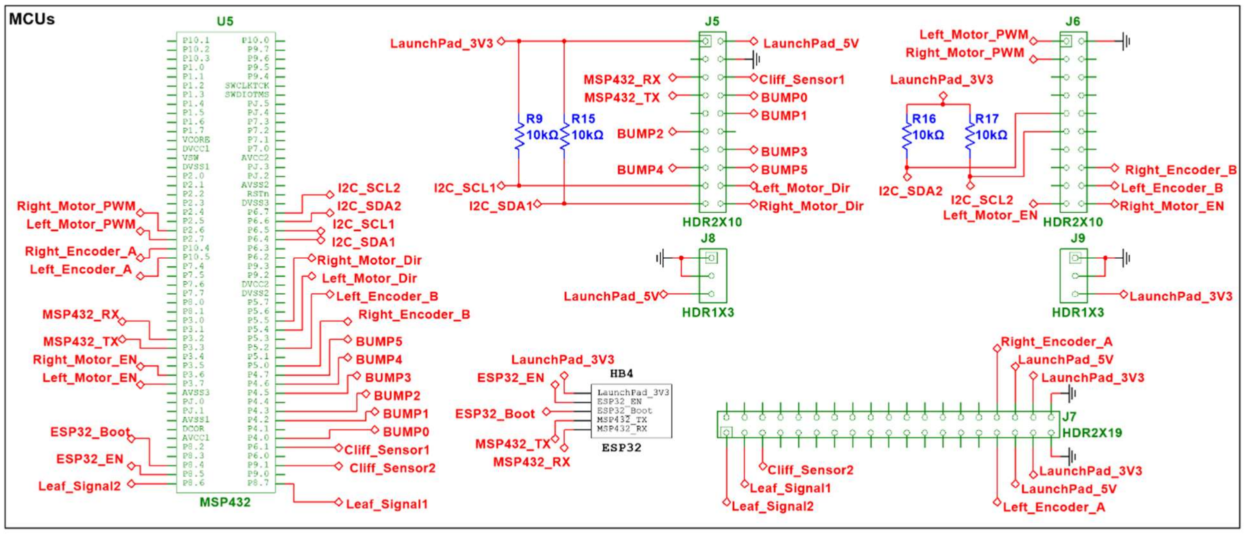


Figure 5: MSP432 Hardware Schematics

When the leaves of the *Mimosa pudica* experience a physical stimulus, the leaflets fold inwards towards the stem. This generates an action potential in the plant which can be measured. Due to the difficulties encountered in acquiring a grown, healthy mimosa plant that experimental data could be measured from, a simulated waveform was created, as shown in Figure 6. The amplitude, duration, and overall shape of this waveform was modelled based on research on plant electrophysiology published by Jorg Fromm and Silke Lautner [63] and research on signal transduction in Mimosa pudica by Alexander Volkov et. al. [64].

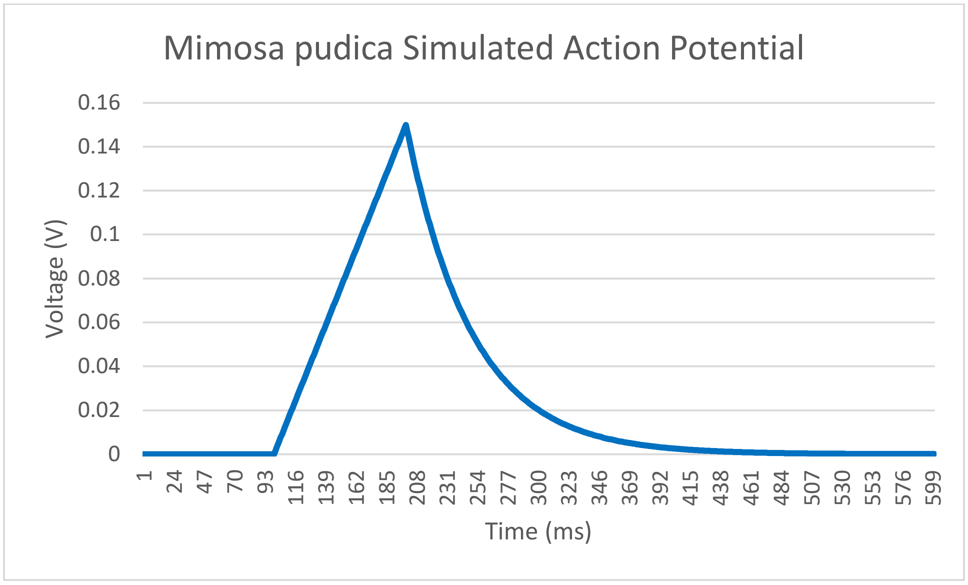


Figure 6: Simulated Mimosa Pudica Action Potential

Figure 7 below shows the overall block diagram of the analog signal processing circuit used to measure the signals generated by mimosa plant.

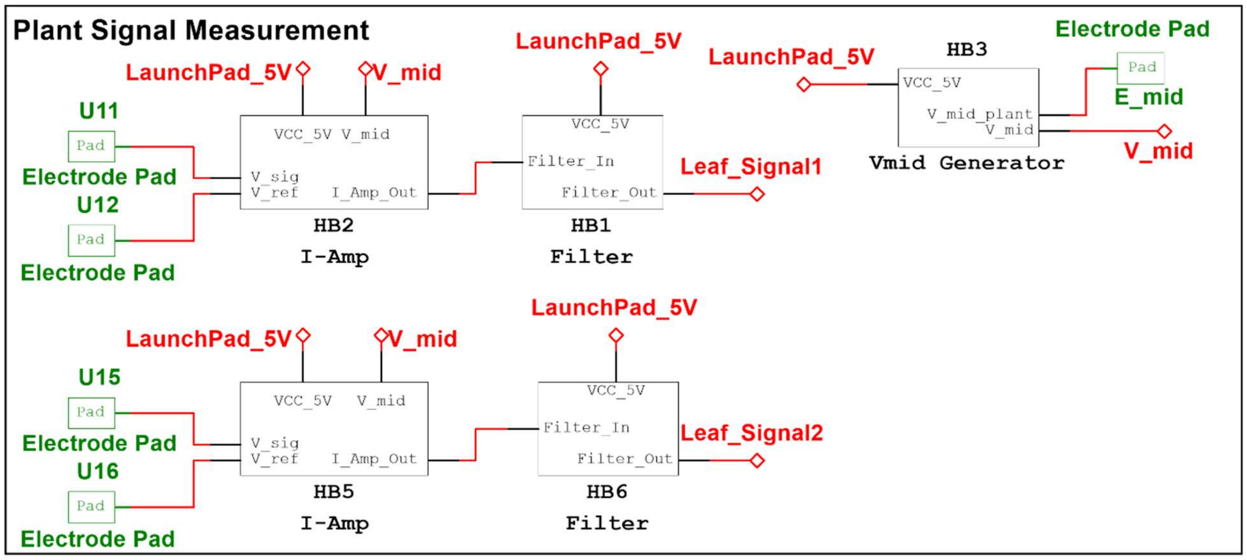


Figure 7: Analog Signal Processing Circuit Block Diagram

This subsystem uses two AD632 instrumentation amplifiers [65] and antialiasing filters constructed using LMC6582 operational amplifiers [66], providing the user with the ability to attach three electrodes to a plant in order to measure any generated signals. First, the electrodes should be attached to the plant and the electrode gel should be placed surrounding the electrode. Two of the electrodes are used to measure the differential signal from the plant, and the third electrode is connected to a DC voltage that drives the plant to a midpoint voltage between the instrumentation amplifier supply rails. The signals measured from the electrodes are then inputted to an instrumentation amplifier circuit, which can be seen in the schematic shown in Figure 8.

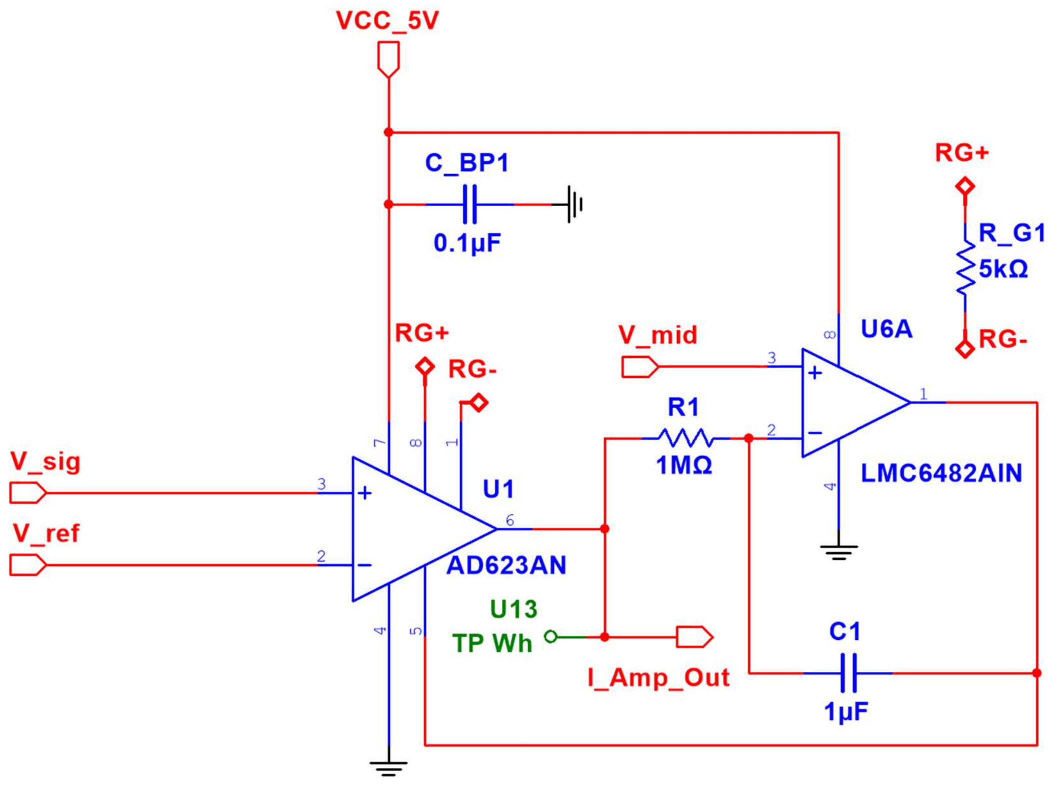


Figure 8: Instrumentation Amplifier Circuit Schematic

The gain of the instrumentation amplifiers was determined using the amplitude of the simulated input waveform and the gain equation listed in the AD623 datasheet. The output of the plant signal chain is inputted to a 3.3V referenced ADC channel on the MSP432, so the gain was set to 21 in order to produce a maximum voltage amplitude of 3.15 V, which is within the configured ADC range. This design was modified while conducting the test plan due to an error discovered in the design beyond the point of hardware revisions. The V_{mid} generator circuit that centers the instrumentation amplifier output to 2.5 V should have been designed to center the output at 1.65 V instead for use in the 3.3V ADC channel.

The DC voltage used as the V_{mid} input to the integrator op-amp, U6A, was generated using a voltage divider and an op-amp configured as a unity gain buffer. The output of the V_{mid} generator was also buffered by the B-side of the op-amp in order to be used as the DC offset voltage for the plant as well. The circuit schematic for the V_{mid} generator can be seen in Figure 9.

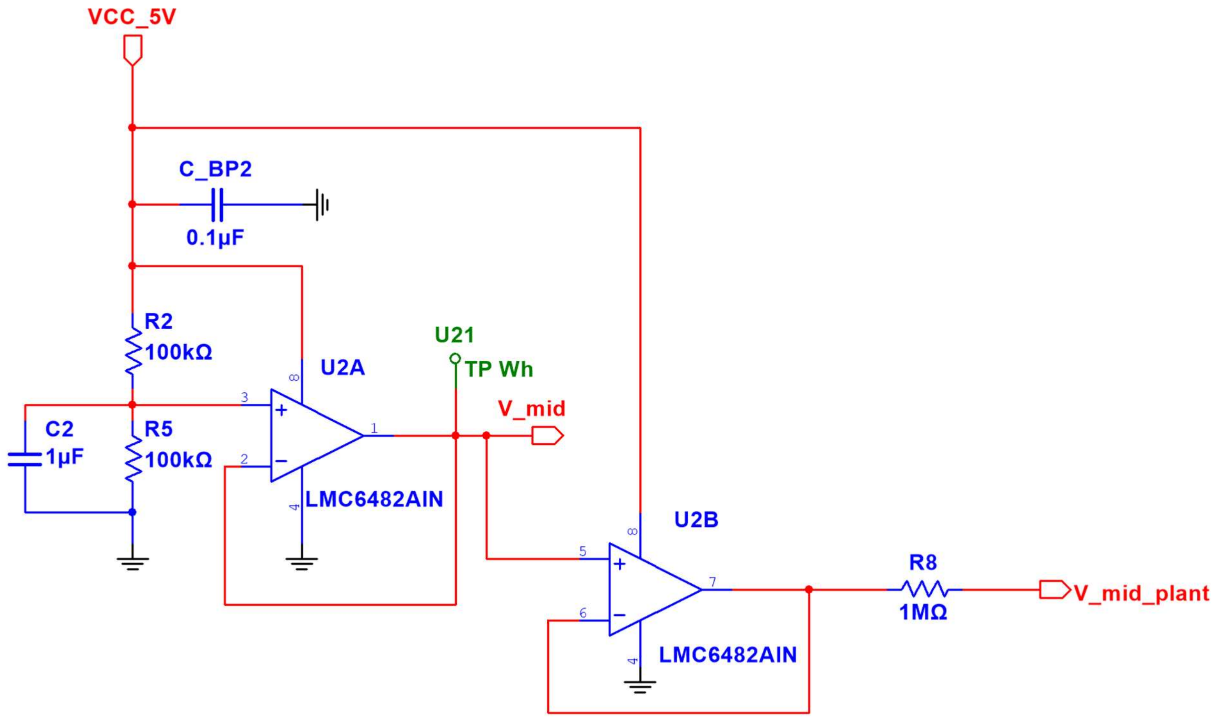


Figure 9: V_{mid} Generator Circuit Schematic

The antialiasing filters for both blocks are identical, and their schematic can be seen in Figure 10. The filter was designed as a cascade of two Sallen-Key low pass filters. Due to uncertainty in the frequency of the plant signals at the time of design, the corner frequency of each stage was designed to be 100 Hz in order to avoid any potential attenuation of the desired signal. The cascaded filters have an overall Butterworth response.

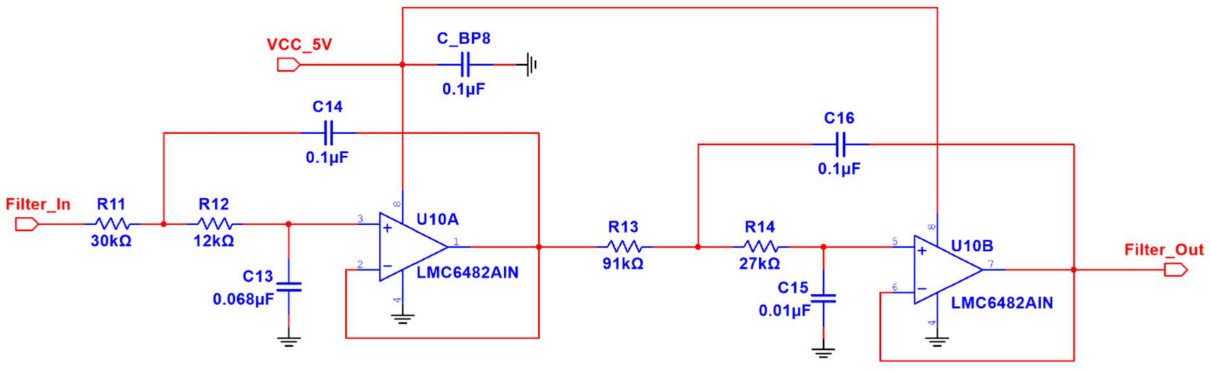


Figure 10: Antialiasing Filter Schematic

The outputs of the antialiasing filters are connected to two ADC channels on the MSP432. The filters produce an attenuation of approximately -80 dB at 1 kHz, so the ADC channels were configured to sample at a rate of 2 kHz by the Nyquist rate.

The ESP32 hardware design included connecting the necessary pins for powering and programming the device, along with the various sensor connections. The schematic diagram can be seen in Figure 11.

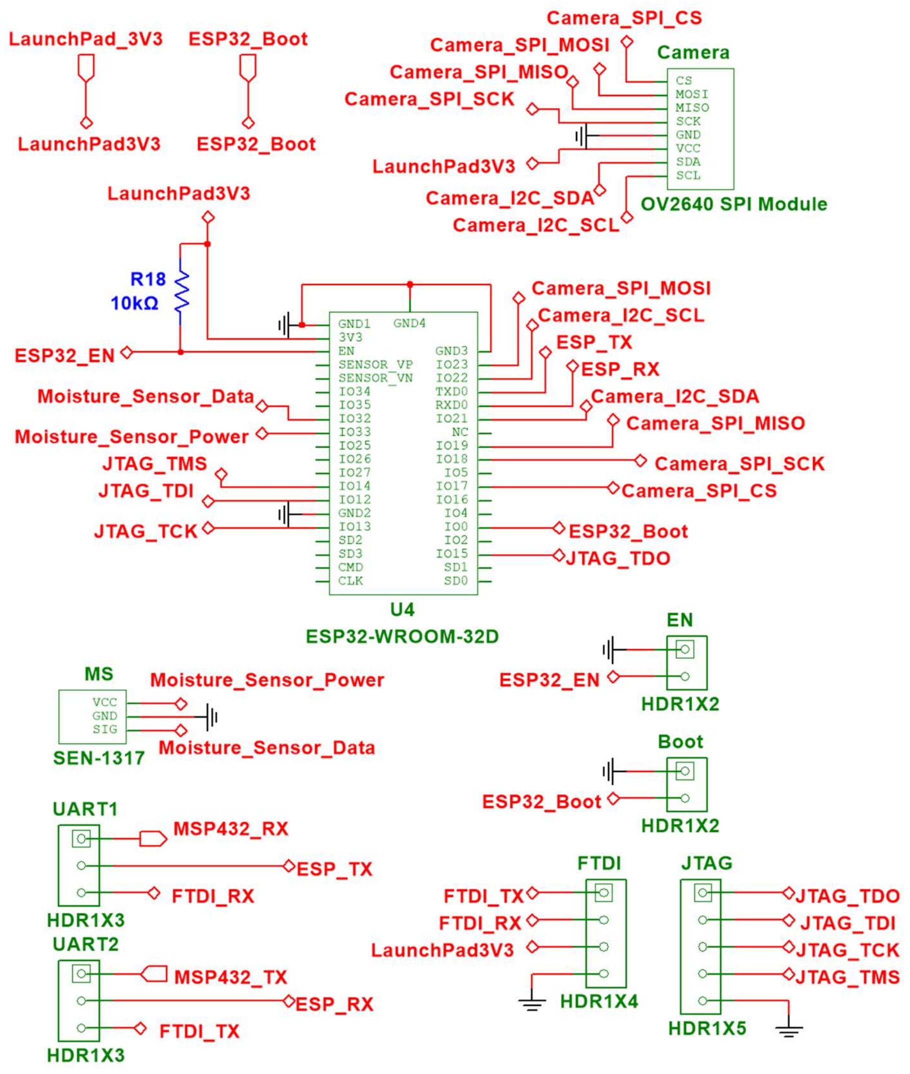


Figure 11: ESP32 Hardware Schematics

The primary programming interface for the ESP32 is an FTDI header, and a JTAG header was also included as a backup interface. The device is powered from the 3.3 V supply from the robot chassis, and the programming mode was controlled using an external pin header. The moisture sensor data is connected to an ADC channel on the ESP32, and the device is powered from a GPIO output pin in order to prevent corrosion from the device being constantly powered. The camera header includes both an I2C interface for sending commands and an SPI interface for receiving the image data. Designated internal pullup resistors on the ESP32 were used on the I2C bus.

The PCB was designed to attach to the MSP432 LaunchPad expansion header pins, and was limited to the same board footprint as such. Figure 12 shows the completed PCB layout and trace routing. The board was designed using two layers in order to simplify manufacturing and decrease costs. Various test points were also added at the output of each stage for debugging purposes.

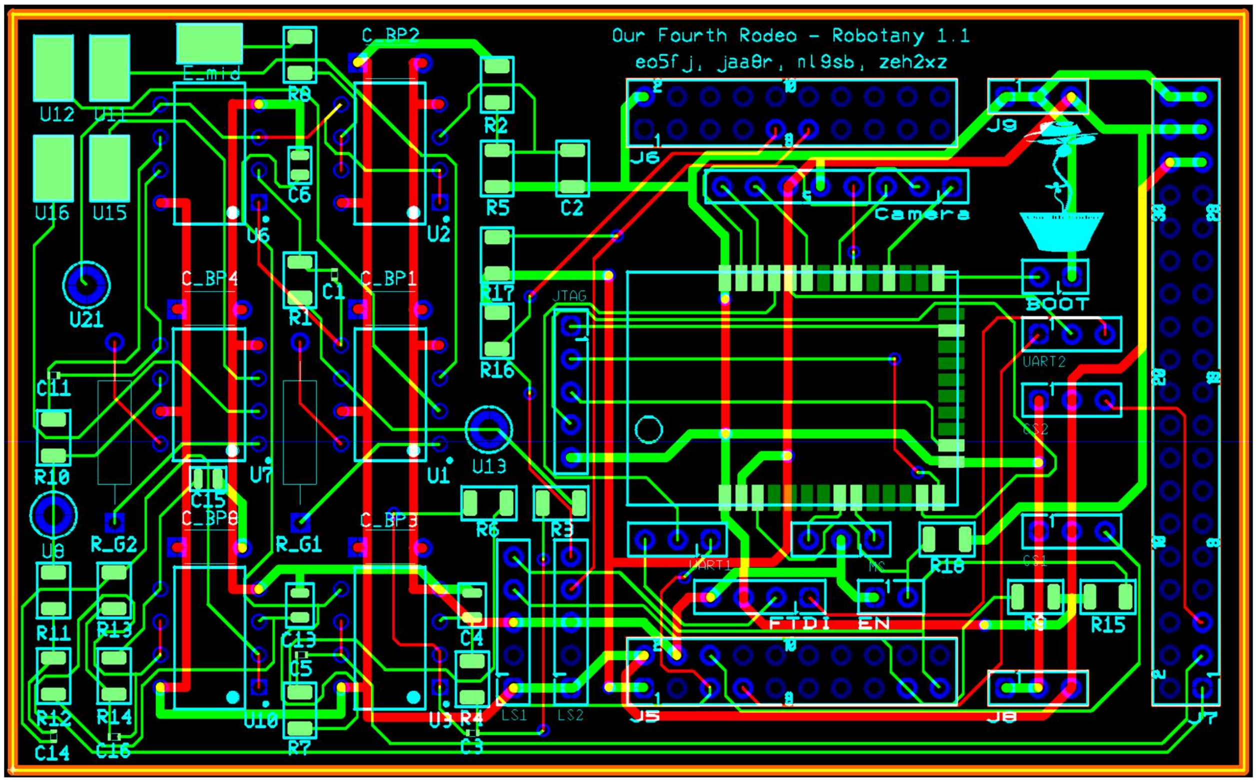


Figure 12: PCB Layout

Firmware and Control

The control algorithm flow is displayed in Figure 13 below. The control algorithm was designed to integrate the plant signals, light data, bump switches, and IR distance sensors through a task scheduler, in which all associated sensors were connected to the MSP432. The control algorithm first checks to see if a lighting flag is set, which occurs when ADC channel 18 reads in signals from the plant that are greater than a certain threshold voltage. This comparison is made using the raw ADC value, rather than a voltage value, to avoid any unnecessary floating-point conversion on the microcontroller. If the lighting flag is set, the control algorithm will "wake up" and initiate I2C communications with the two VEML-7700 light sensors [62] to determine the system's next movement based on the ambient light measurements on either side of the robot. The algorithm uses integral control, in which the controller is run at a Δt equal to the sampling rate of the sensor. The error between the desired difference threshold and the actual difference is first calculated, and then the system updates its position based on that value. If the difference is less than the negative threshold, the system will turn right until it is within the threshold and the motors will stop. If the difference is greater than the positive threshold, the system will turn left until the signals are within the threshold and the motors will stop.

Once the system has oriented itself towards the light source, it will move forward a short distance and check its current lighting value against its previous lighting value to determine if it is going in the optimal direction. If the previous value is significantly less than the current lighting value, the system will drive forward another set distance. If the previous light value is greater than the current value, the system will turn around 180-degrees and then drive forward a set distance. When the system determines that there is not a significance difference between the current and previous lighting signals, it will stop the motors and clear the lighting flag. As only the front of the Robotany has bump switches and cliff sensors, the system is designed to only drive forward to ensure maximum safety. If the bump switches are triggered or if an ADC reading of above a threshold value is read from the cliff sensors, the Robotany will reverse a short distance, stop the motors, and clear the lighting flag. The cliff sensors, bump switches, plant signals, and light sensor signals are assigned interrupt priorities in descending order. The user will be notified once per week to rotate their plant, preventing the plant from growing asymmetrically in a single direction due to the system being designed to only drive in the forward direction.

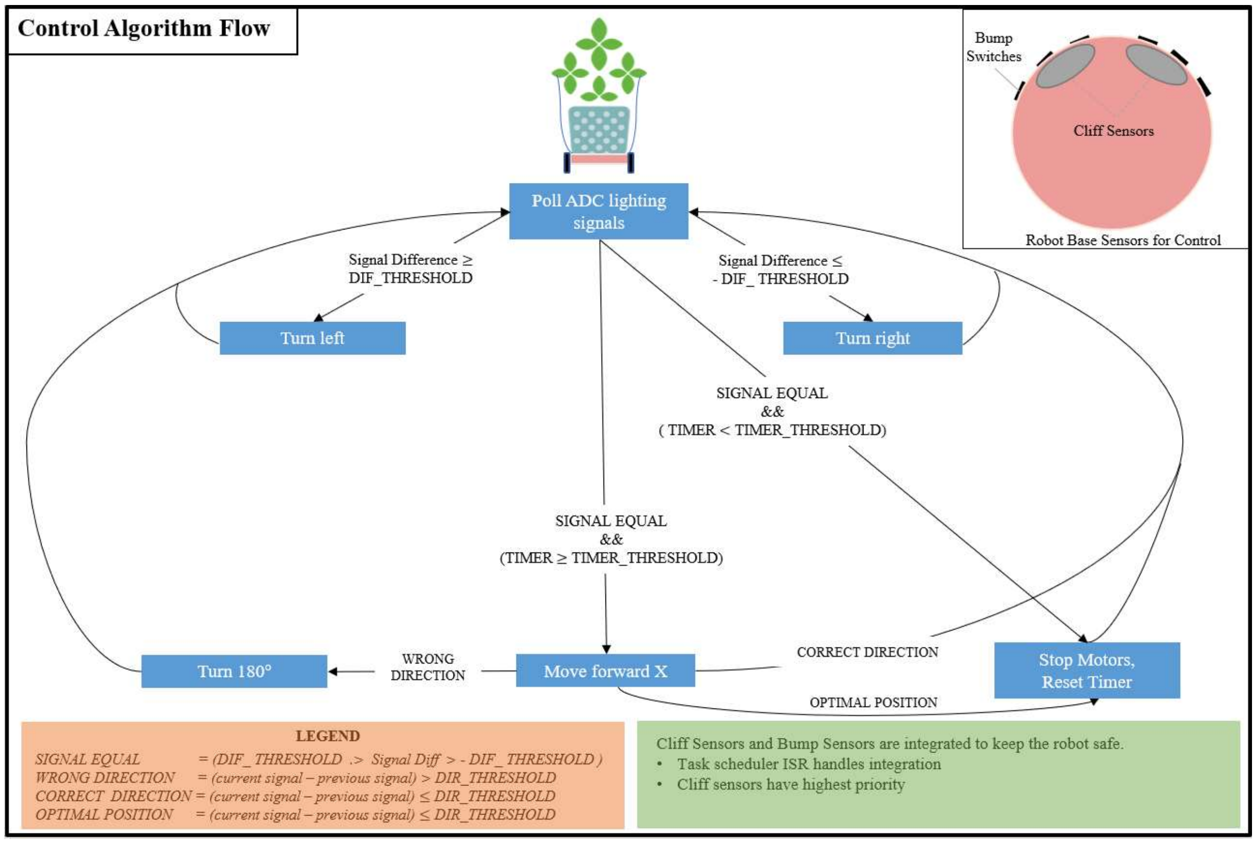

Control Algorithm Flow
Bump Switches
Cliff Sensors
Robot Base Sensors for Control
Poll ADC lighting signals
Signal Difference ≥ DIF_THRESHOLD
Signal Difference ≤ - DIF_ THRESHOLD
Turn left
Turn right
SIGNAL EQUAL && (TIMER < TIMER_THRESHOLD)
SIGNAL EQUAL && (TIMER ≥ TIMER_THRESHOLD)
Turn 180°
WRONG DIRECTION
Move forward X
CORRECT DIRECTION
Stop Motors, Reset Timer
OPTIMAL POSITION
LEGEND
SIGNAL EQUAL = (DIF_ THRESHOLD .> Signal Diff > - DIF_ THRESHOLD)
WRONG DIRECTION = (current signal – previous signal) > DIR_THRESHOLD
CORRECT DIRECTION = (current signal – previous signal) ≤ DIR_THRESHOLD
OPTIMAL POSITION = (current signal – previous signal) ≤ DIR_THRESHOLD
Cliff Sensors and Bump Sensors are integrated to keep the robot safe.
• Task scheduler ISR handles integration
• Cliff sensors have highest priority

Figure 13: Control Algorithm Flow Chart

The SEN-13637 moisture sensor [67] is connected to the ESP32. From this sensor, the ADC data is collected to be sent from the microcontroller to the server, in which low ADC values map to dry soil and high ADC value map to wet soil. Once the data uploaded to the server, the value is checked against a predetermined threshold to see if the user should be notified to water the plant.

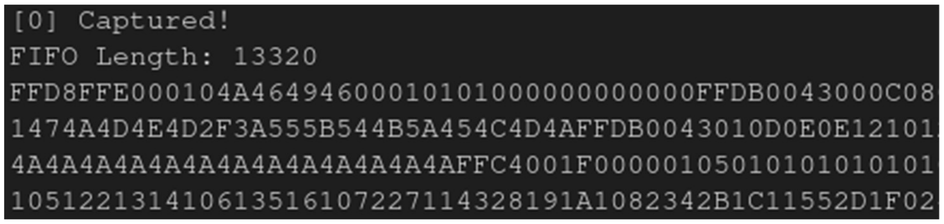
Computer Vision and Growth Prediction

The machine learning model and computer vision algorithms are integral aspects of the system, as they are used to monitor growth levels and inform the user when they can place electrodes on the plant and when they can harvest the leaves of the plant. The latter model primarily applies to herbs, such as basil or mint. The purpose of the computer vision algorithms is to both remove the background of the image, preventing any personal information regarding a user's location from being stored, and to remove glare from an image prior to it being passed into the machine learning model. These items were implemented through use of TensorFlow [36], OpenCV [35], and Python.

The overarching information flow of this subsystem is as follows: an image is captured by the OV2640 camera [68] and sent to the server via the ESP32. The image is then passed into the background and glare removal algorithms, described in further detail below, and the resulting image are saved onto the server. Next, the image is passed into the neural network model(s) for growth classification. The results of the classification and the saved image are displayed to the user on the mobile application.

In order to implement the computer vision system, a camera driver for the ESP32 was developed and tested. A reference driver was available for Arduino [55], but this required conversion to function with the ESP-IDF. Unfortunately, there are major structural differences between the provided library for the camera module and the design of ESP-IDF for similar operations, so a direct, function-to-function port was not possible. Each function was roughly replicated with an ESP-IDF equivalent where possible, or completely rewritten when not.

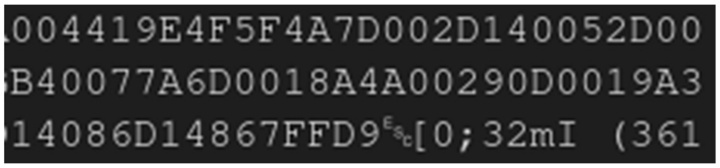
To read the images from the camera, a message is sent to capture and store in FIFO buffer. A FIFO length is then received. Unfortunately, this is aligned to approximately the kilobyte, so there is additional unneeded data in the buffer. As a result, it is necessary to parse the received buffer, searching for the "magic bytes" of a JPEG, the bytes FF D8, and continue reading until we encounter the end of the JPEG image, as specified in the Arduino library and JPEG standard format, bytes FF D9.



```
[0] Captured!
FIFO Length: 13320
FFD8FFE000104A46494600010101000000000000FFDB0043000C08
1474A4D4E4D2F3A555B544B5A454C4D4AFFDB0043010D0E12101
4A4A4A4A4A4A4A4A4A4A4A4A4A4AFFC4001F00000105010101010101
1051221314106135161072271143281 91A1082342B1C11552D1F02
```

Figure 14: Start of the FIFO Buffer

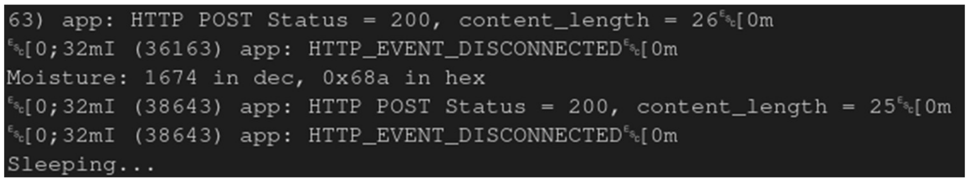
Figure 14: Start of the FIFO Buffer shows the start of the FIFO buffer, while Figure 15: End of the FIFO Buffer shows the end of the FIFO buffer, indicating that the camera was functioning as desired, providing JPEG images. Further, this action occurs as a single, hardware-based transaction, compared to the series of transactions performed by the Arduino library. This increases the speed of the capture process significantly from the reference design at the expense of additional memory requirements to pre-allocate the necessary transmit and receive buffers.



```
004419E4F5F4A7D002D140052D00
B40077A6D0018A4A00290D0019A3
14086D14867FFD9␛[0;32mI (361
```

Figure 15: End of the FIFO Buffer

Communicating the image data and the moisture sensor data requires POST requests to the API over HTTPS. As such, the ESP-IDF's HTTP client [69] is utilized for this purpose. All data was transmitted with an identification number as an octet stream to the API endpoint. Next, several tasks are completed, including 1) determining the type of data that was sent, 2) determining which device sent the data, and 3) the necessary actions required based on the received data. The ESP32 waits for a period before sending any additional data that it collects, entering a low power state in the meantime. Figure 16 shows these requests being sent successfully (a response code of 200 indicates the successful upload of all data and matched hashes), then disconnecting from the server. The server then handles much of the processing, though the ESP32 does not wait for this to occur.



```
63) app: HTTP POST Status = 200, content_length = 26␛[0m
␛[0;32mI (36163) app: HTTP_EVENT_DISCONNECTED␛[0m
Moisture: 1674 in dec, 0x68a in hex
␛[0;32mI (38643) app: HTTP POST Status = 200, content_length = 25␛[0m
␛[0;32mI (38643) app: HTTP_EVENT_DISCONNECTED␛[0m
Sleeping...
```

Figure 16: Successful HTTP Requests from ESP32

Several image processing steps occur before the image is saved to the server, starting with background removal. The background removal algorithm first applies a gaussian blur to the image to reduce the noise and detail of an image. The image is then passed through a pre-trained structured forest edge detection model to identify prominent edges [38]. Next, the image is passed into a median filter to remove impulse noise that can occur from sharp disturbances in the image signal [70]. The image is then passed through a pre-trained structured forest edge detection model to identify prominent edges [38]. Next, the image is passed into a median filter to remove impulse noise that can occur from sharp disturbances in the image information [70]. The most significant contour is then located on the image to create a background mask, allowing for the filled in region of the contour to be extracted. This allows for the potted plant to be isolated from the background of the image and, therefore, prevents any security hazards that could occur from having the user's private space stored in the server. The background removal is demonstrated in Figure 7 below, in which the leftmost image is the image passed in, the next image is after edge detection is performed, the center-right image is after the most significant contour is identified, and the right-most image is the final result.

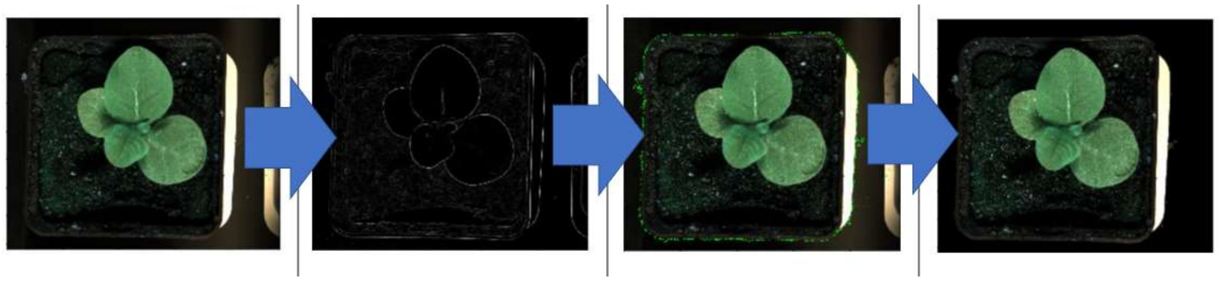


Figure 17: Background Removal Algorithm

The glare removal algorithm first converts the image to Hue Saturation Value (HSV) color-space. From this, the values in the range of light to dark white were determined to create a glare mask. The glare mask and the original image are then passed into OpenCV's inpainting method, where the Navier-Stokes method of inpainting is applied and used to update the image intensities of the region with a partial differential equation, and the image Laplacian is used to determine smoothness [71]. This allows for a more normalized image to be passed into the machine learning model and prevents any adverse lighting conditions from negatively impacting the results.

The growth detection of the plant is implemented using two separate machine learning models. Both models have identical architecture and differ only in training set data and classification fields. The neural network has a goal of mapping the input, an image, to a classification field: NOK, ElectrodesOK, and HarvestOK, in which NOK indicates that insufficient growth has taken place for electrode placement or harvesting to take place. The neural network is designed to receive an integer that represents the number of green pixels found in the image. This not only optimized the space the model took up but also allowed for a more flexible dataset to be used for training. Determining the proportion of greenery in the image is accomplished by first resizing an image passed in to 150 by 150 pixels. It is then converted to the HSV color space and the number of pixels in the range of light green to dark green is calculated. This calculation is then passed into the model.

A 3-layer feed-forward, or sequential, model is used for growth detection. The first layer takes in the input as a 1x1 array and uses a dense structure with 8 neurons and a rectified linear activation function. The activation function allows the neurons to "fire" [72]. The middle layer is a hidden layer and follows a similar dense structure to the first layer, with 8 neurons and a rectified linear activation function as input parameters. The dense output layer specifies the number of classifications (two) and uses a SoftMax activation function. This activation function treats the output as a probability distribution, so that each classification field is assigned a respective probability and the maximum value is used to classify the image [72]. The model was compiled using an Adam optimizer, an extension of stochastic gradient descent, and uses sparse categorical cross entropy to calculate loss. Once the model was compiled, it was trained on 10 epochs with a batch size of 2.

This neural network architecture was trained on separate datasets to produce two models that will satisfy different purposes: one model to classify the difference between insufficient growth and sufficient growth for electrode placement, and another to classify the difference between insufficient growth and sufficient growth for plant harvesting. This structure was designed with the intention that a plant only needs to be classified as harvestable in the case that the user is growing an herb. The dataset used for training was composed of over 6,000 images, with approximately 2,000 images in each class (NOK, ElectrodesOK, HarvestOK). All images were scaled to 150x150 and converted to the HSV color-space to calculate the amount of greenery in the image, as described above. Greenery calculation and scaling will occur to all images captured using the system before this image data can be passed into either of the neural network models. After the two models were trained with their respective datasets, they were placed into the server.

IoT Software

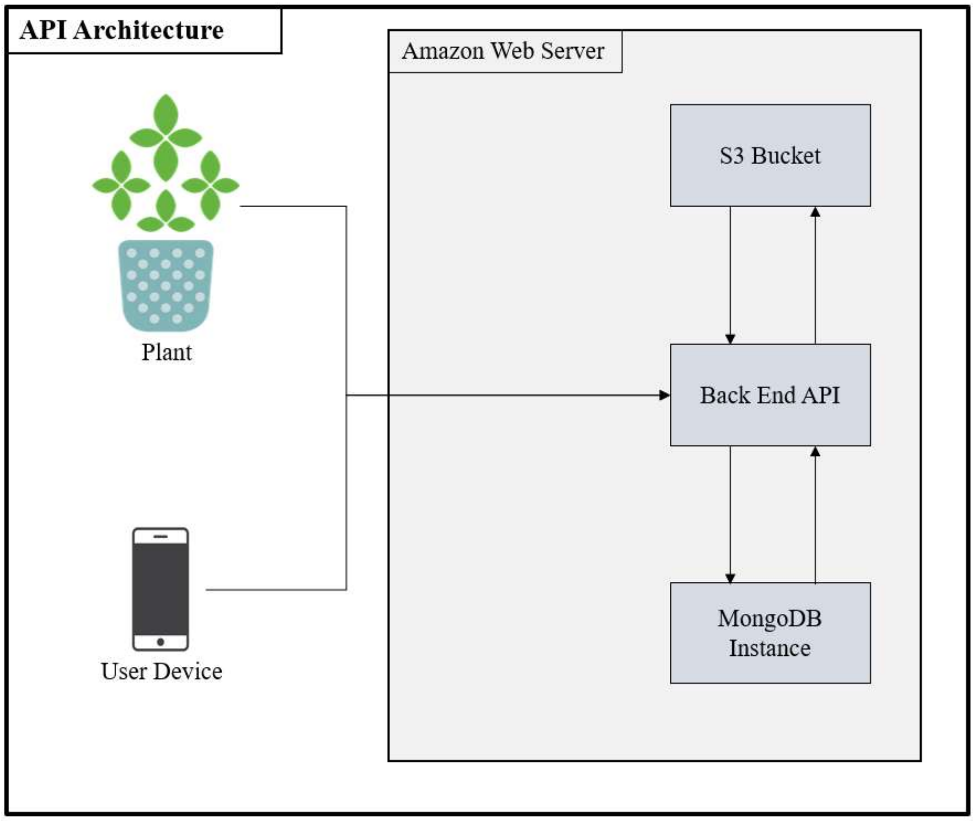


Figure 8: API Architecture

The mobile application was developed to provide a user interface to the Robotany on either Android, iOS, or the web, and serves as the primary point of communication between the user and his or her plant. The multi-faceted application performs several functions:

1. Securely associates a user with a Robotany

2. Connect the Robotany to the internet
3. Notifies the user of action events
4. Displays moisture metrics and growth data with interesting visual approaches

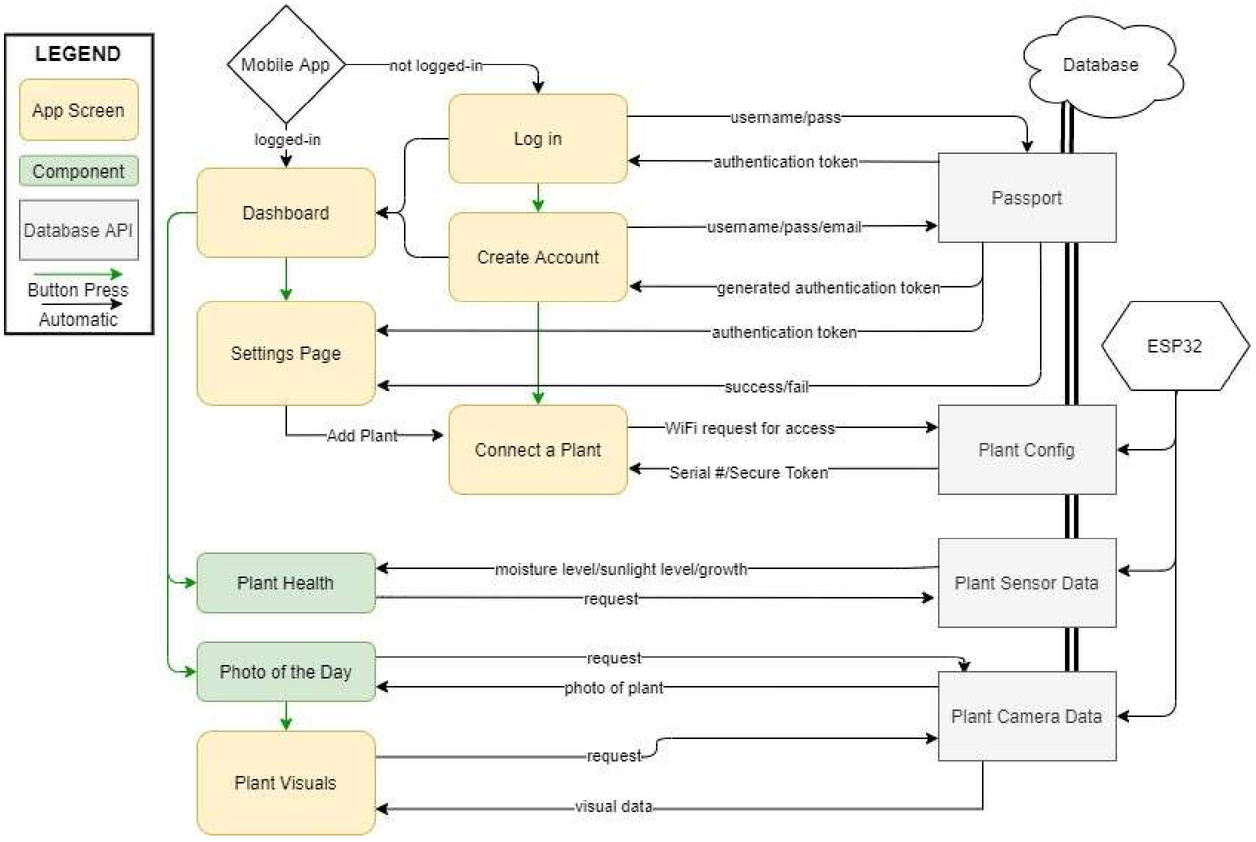


Figure 18: Mobile Application Use Flow Chart

#1. Securely associate a user with a Robotany.

When a user opens the app, the authentication protocol is launched. The protocol will ping the database to identify whether an active session is present on the device, with the API viewing the session token held by the device and comparing it to stored information in the database. If the session is present and fresh (not expired), the API responds with user account information and the app automatically logs the user in. If the session token is not present or is expired, the app prompts the user to either create an account or log in.

The app cannot be accessed without a valid session token, so a user is forced to login or create an account to work with the application. This is primarily to ensure arbitrary access is not permitted. With the creation of an account, the user must provide some identifying credentials (username, email, password) and connect a Robotany to the account. The credentials are passed to the API to handle account and session creation and storage with Passport, while the plant details are stored in the database to allow the Robotany to begin to report information. To associate a plant with the account, the user must follow a series of tasks displayed on the app – which serve to connect the Robotany to the user's home wireless network.

Once the user has supplied the required information, a token will be created and passed to the application, allowing the user to access to the app's main content. The user's session will be stored in the database to allow automatic login upon app entry.

If the user submits a log-in request, the submitted username and (hashed) password are compared to the version stored in the database to confirm these details with the existing credentials. If an existing account is found, a valid session token is returned. If not, the user is prompted to try again, and a session is not established.

Once logged in, the user gains access to the main content of the application. The "Profile" screen allows users to add a new plant, delete a plant, and/or log out, while there are several other screens that breakdown the health and history of the plant.

#2: Connect the Robotany to the internet.

As a user account is being created, or the user attempts to add a new Robotany to their account, they will be requested to connect to the Robotany itself, which creates a Wi-Fi network on its first boot through the ESP-IDF's SoftAP and Wireless Provisioning API. From there, the ESP32 opens a simple HTTP server with endpoints for listing capabilities, establishing sessions, scanning for wireless networks, and sending credentials, all to connect the ESP32 to a nearby Wi-Fi network. Upon receiving credentials, the SoftAP and server shut down and the ESP32 attempts to connect to the internet with the given credentials. If the network credentials are invalid after attempting to connect, or there is no internet connection, the credentials will be flushed and the provisioning services will restart, allowing a user to re-enter the credentials.

Sending these credentials must be done in a simple, fast manner, which is where the mobile app is involved. The "Add plant" screen requests that a user switches wireless networks to the Wi-Fi network being hosted by the ESP32 (it will always start with "Robotany_"). Upon the Wi-Fi network being detected, the mobile app will allow the user to enter the proof that they own the Robotany. The MAC address is used for simplicity in the demo, though proofs could be generated in other manners. The Wi-Fi credentials are then requested. A byte buffer of these details is then generated and submitted to the ESP32's provisioning endpoint, as well as a confirmation request. This sends the credentials and requests a connection be attempted. The mobile application will then wait for the ESP32's wireless network to disappear and for the phone to reconnect to some other network. After a short wait to ensure it does not reconnect to the Robotany's wireless network in the event of a failure, the mobile application will log the connection of the plant to the API, adding it to the user's account.

The Wi-Fi credentials are stored in the ESP32's NVRAM and will be used upon future reboots of the Robotany. To disassociate from a network, the NVRAM could be reset through some signal to the ESP32, or the Robotany can be kept away from the network for some time.

#3: Notify the user of action events.

Push notifications were configured using Expo's notification library, expo-notifications [73], which provides a pre-built structure to present and schedule notifications, as well as fetch notification tokens. When the user logs into the app for the first time, a notification permissions request will be sent to the user's phone, which will allow the user to either opt in or out of notifications.

If the user opts in, the user will be presented with notifications whenever the moisture readings drop below a certain threshold (with a reminder if not watered in a certain timespan) and a weekly notification to remind the user to rotate the Robotany 90 degrees to ensure the plant grows evenly strong on all sides.

#4. Display interesting moisture and growth metrics through visual approaches.

After the user token has been identified or created, the application pulls moisture, growth, and image data from the database using API endpoints. The current moisture data is displayed to the user on the Home page, and 24-hour historical data is displayed on the Plant Details page.

The most recent photo of the plant, having been processed for background removal on the server, is also displayed on the Home page. The photo is also analyzed server-side to identify whether the plant should have electrodes attached or be harvested. This growth information is stored in the database, accessed by HTTPS request via the API, and displayed on the Home page of the mobile application upon a plant's details being requested.

Problems and Design Modifications

The team experienced the following problems that led to modifications in the design: cliff sensor selection and the implementation of plant signals into the project.

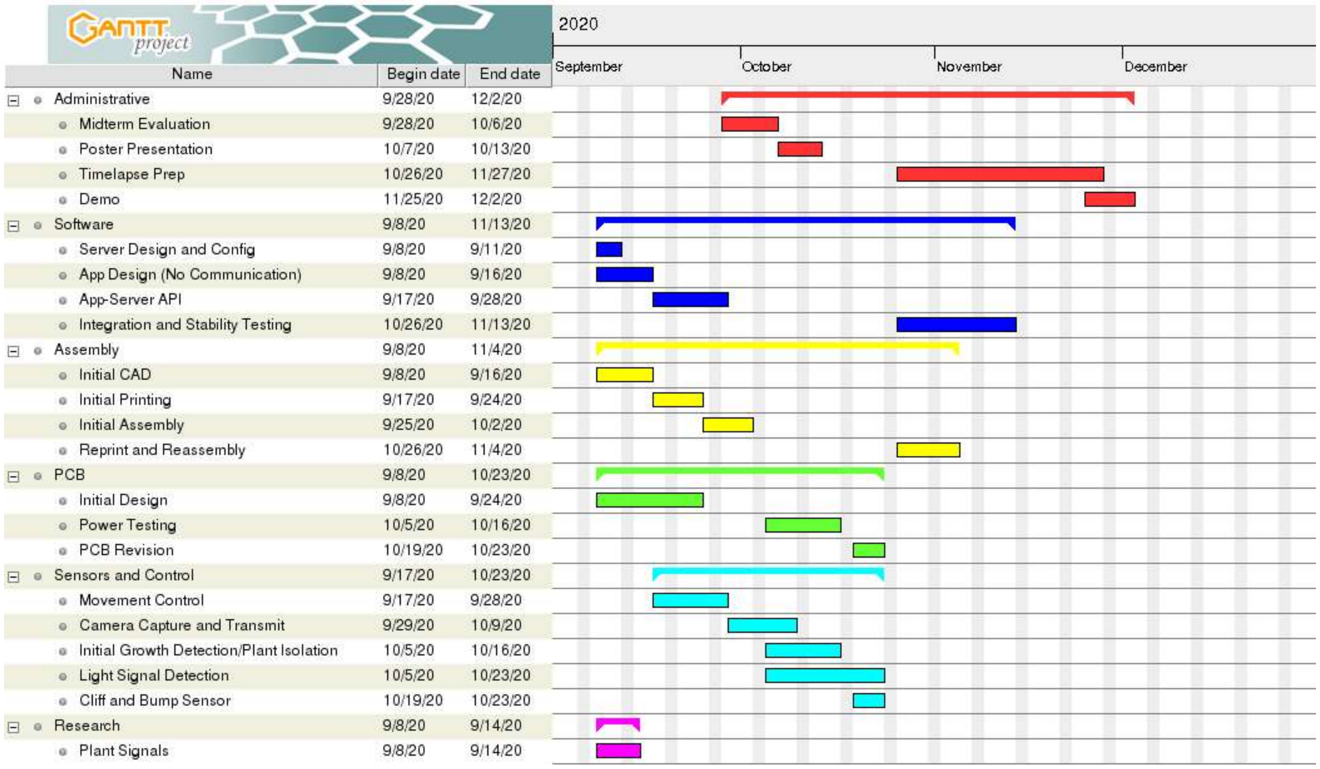
The team initially designed the system with 3 reflectance sensors to sense cliffs or ledges that would be attached to the robot base. However, the reflectance sensors required a separate calibration for each surface and lighting conditions that the system was used on. This resulted in a non-scalable design, and so the team swapped the reflectance sensors for two IR distance sensors that were bulkier, but did not require calibration. Two cliff sensors were used due to the aforementioned bulkiness of the new sensors and were spaced towards the front base of the system to ensure an optimal position for responding to ledges.

The V_{mid} generator was originally designed to center the signals from the plant at 2.5 V, half of the Vcc rail for the amplifiers. Due to the ADC configuration for the cliff sensors, the ADC channels were referenced to 3.3 V, and when this was discovered, it was too late to change the voltage offset for V_{mid}. In order to successfully demo the project, the V_{mid} generator was removed from the circuit and the integrator input was attached to ground in remove the DC offset from the signal and prevent clipping.

Originally, two electrodes were going to be placed on opposite sides of the plant and used solely for the control algorithm by analyzing the difference between the two signals and updating accordingly. However, it was determined that the frequency of the plant signals was too low to provide significant data to accomplish this. As such, the plant signals were used to "wake up" the control algorithm if a signal was registered at above a specific threshold value, rather than act as the input data being tracked by the controller. The difference between light signals was then determined through values from two VEML7700 light sensors placed at opposite ends of the plant.

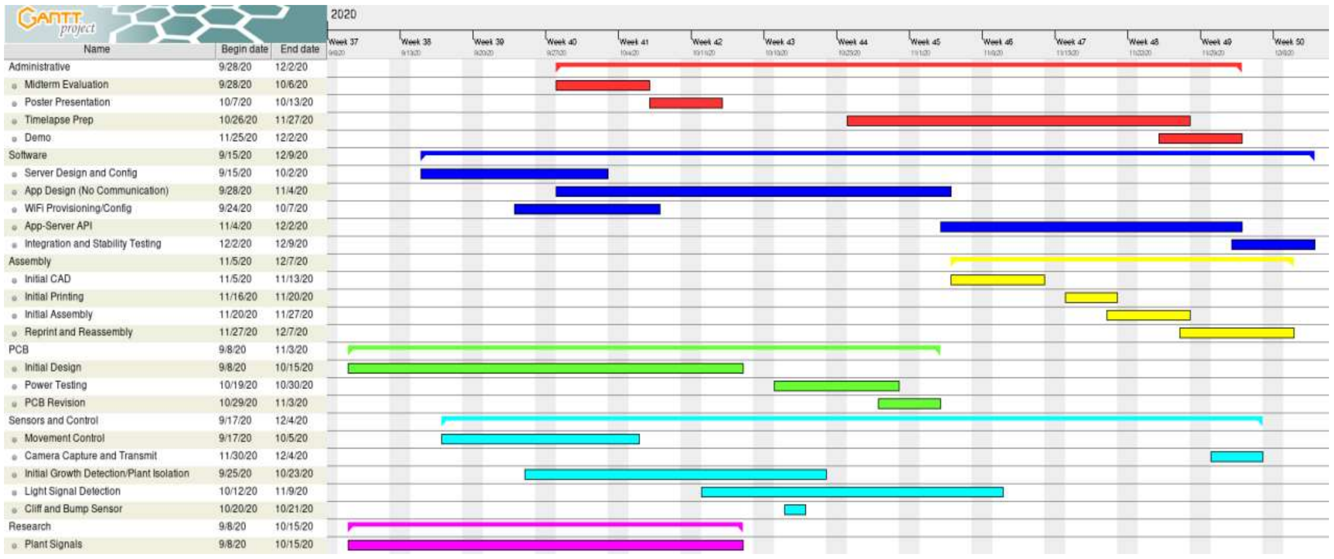
Project Timeline

The first proposed Gantt chart can be seen be in Figure 19. The original timeline sought to complete the software development for the mobile application and server prior to the delivery of the necessary hardware components. Once those components were delivered, the plant signal retrieval circuitry was to be developed in parallel with the wireless communications and the sensor integration on the TI-RSLK MAX robot. The signal retrieval circuit was planned to be completed before the PCB can be designed and printed. The actual project timeline followed the Gantt chart displayed in Figure 20. While this Gantt chart differed in many components, including software timeline being shifted back and the amount of time dedicated to the camera module expanded, it was still structured in accordance with major deadlines in the course. Specifically, the updated timeline remained structured around deadlines pertaining to the proposal, the midterm design review, and the final project submission. The expansion of the tasks was determined necessary following midterm design review, when the team realized that certain items would take longer than planned. The CAD for the necessary enclosures needed to occur after the PCB design to ensure that all of the hardware fits within the enclosure. This CAD work needed to be followed by a 3D print of the design, which had to occur before final assembly. At final assembly, the project was at the point of final tuning and qualification of combined parts. While the timeline shift pushed integration back, team members had incrementally tested their parallelized project components well enough to make final integration a digestible task. The team aimed to have a final working system by December 10th, and this deadline was met, allowing us to film a state-of-the-art video demoing Robotany.



Name	Begin date	End date
Administrative	9/28/20	12/2/20
Midterm Evaluation	9/28/20	10/6/20
Poster Presentation	10/7/20	10/13/20
Timelapse Prep	10/26/20	11/27/20
Demo	11/25/20	12/2/20
Software	9/8/20	11/13/20
Server Design and Config	9/8/20	9/11/20
App Design (No Communication)	9/8/20	9/16/20
App-Server API	9/17/20	9/28/20
Integration and Stability Testing	10/26/20	11/13/20
Assembly	9/8/20	11/4/20
Initial CAD	9/8/20	9/16/20
Initial Printing	9/17/20	9/24/20
Initial Assembly	9/25/20	10/2/20
Reprint and Reassembly	10/26/20	11/4/20
PCB	9/8/20	10/23/20
Initial Design	9/8/20	9/24/20
Power Testing	10/5/20	10/16/20
PCB Revision	10/19/20	10/23/20
Sensors and Control	9/17/20	10/23/20
Movement Control	9/17/20	9/28/20
Camera Capture and Transmit	9/29/20	10/9/20
Initial Growth Detection/Plant Isolation	10/5/20	10/16/20
Light Signal Detection	10/5/20	10/23/20
Cliff and Bump Sensor	10/19/20	10/23/20
Research	9/8/20	9/14/20
Plant Signals	9/8/20	9/14/20

Figure 19: Proposed Gantt Chart



Name	Begin date	End date
Administrative	9/28/20	12/2/20
Midterm Evaluation	9/28/20	10/6/20
Poster Presentation	10/7/20	10/13/20
Timelapse Prep	10/26/20	11/27/20
Demo	11/25/20	12/2/20
Software	9/15/20	12/9/20
Server Design and Config	9/15/20	10/2/20
App Design (No Communication)	9/28/20	11/4/20
WiFi Provisioning/Config	9/24/20	10/7/20
App-Server API	11/4/20	12/2/20
Integration and Stability Testing	12/2/20	12/9/20
Assembly	11/5/20	12/7/20
Initial CAD	11/5/20	11/13/20
Initial Printing	11/16/20	11/20/20
Initial Assembly	11/20/20	11/27/20
Reprint and Reassembly	11/27/20	12/7/20
PCB	9/8/20	11/3/20
Initial Design	9/8/20	10/15/20
Power Testing	10/19/20	10/30/20
PCB Revision	10/29/20	11/3/20
Sensors and Control	9/17/20	12/4/20
Movement Control	9/17/20	10/5/20
Camera Capture and Transmit	11/30/20	12/4/20
Initial Growth Detection/Plant Isolation	9/25/20	10/23/20
Light Signal Detection	10/12/20	11/9/20
Cliff and Bump Sensor	10/20/20	10/21/20
Research	9/8/20	10/15/20
Plant Signals	9/8/20	10/15/20

Figure 20: Updated Gantt Chart

The team serialized and parallelized the tasks in such a way that the hybrid semester would be more manageable. Figure 21 aims to visualize how team tasks were distributed.

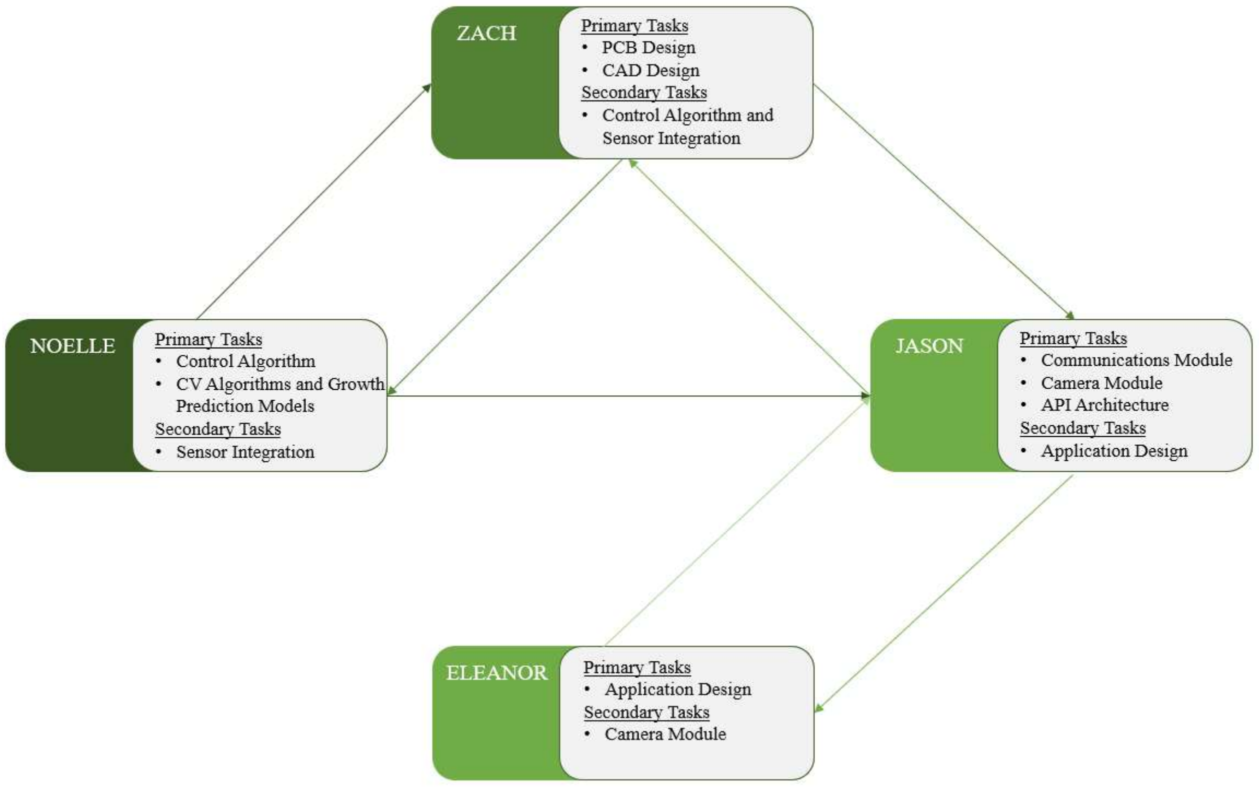


Figure 21: Visualization of Team Tasks

Zach was the primary group member for the signal measurement circuitry from the plant and the CAD design required for the project. He acted as the secondary group member for sensor integration, specifically taking lead on initializing MSP432 I2C communications and calibrating and retrieving light data from the VEML7700 light sensors.

Eleanor chiefly worked on building the mobile application, specifically on designing the user interface, notification system, and data representation. Additionally, she assisted Jason on server development and provide support with camera implementation and computer vision.

Jason was be the primary group member for wireless communications and protocol, the camera module, and server development. Beyond this, he worked on the mobile application and integrating the moisture sensor. Once the initial CAD work was completed, he also was responsible for 3D printing the necessary models and casings.

Noelle was the lead group member for designing and implementing the control algorithm, including integrating the bump and cliff sensors into the software to ensure the system will maneuver over surfaces safely. She was also the primary group member for the computer vision algorithms and neural network design for growth prediction. She acted as the secondary group member for sensor integration into the code base.

Test Plan

The Robotany was divided into multiple systems for testing, including hardware, firmware, and IoT software. Once each system was tested independently, the systems were integrated for final testing.

Hardware

The first component of the hardware test plan was the power system, the flow chart for which can be seen in Figure 22 below.

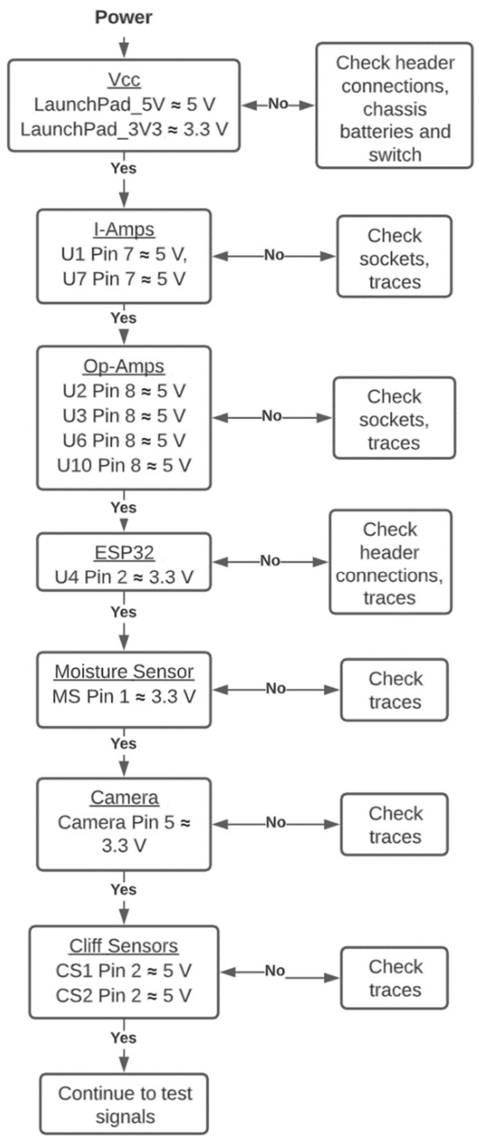


Figure 22: Power System Test Plan Flow Chart

Each of the Vcc pins was verified to have the correct voltage, and each of the ground pins were verified to be at ground potential. Screenshots of the measurement from each pin can be seen in Appendix A. Next, the analog signal processing hardware was tested according to the flow chart shown in Figure 23.

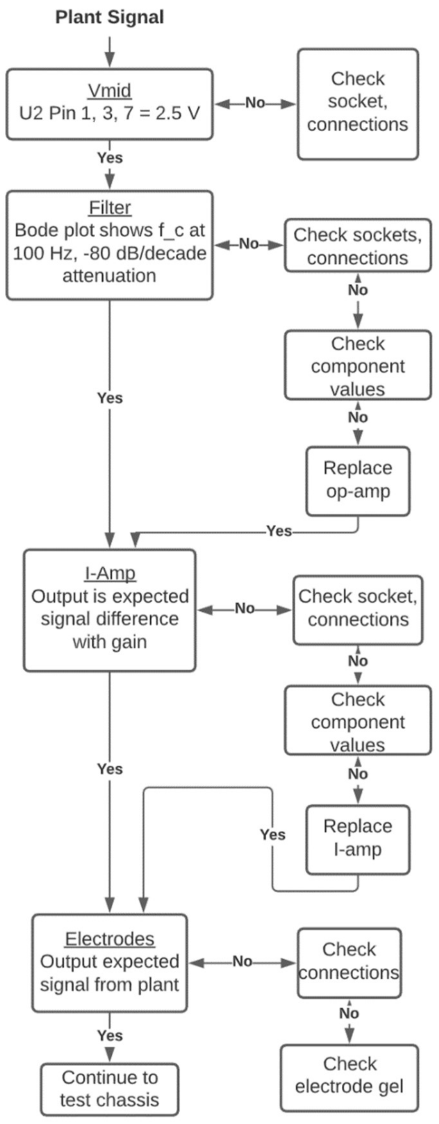
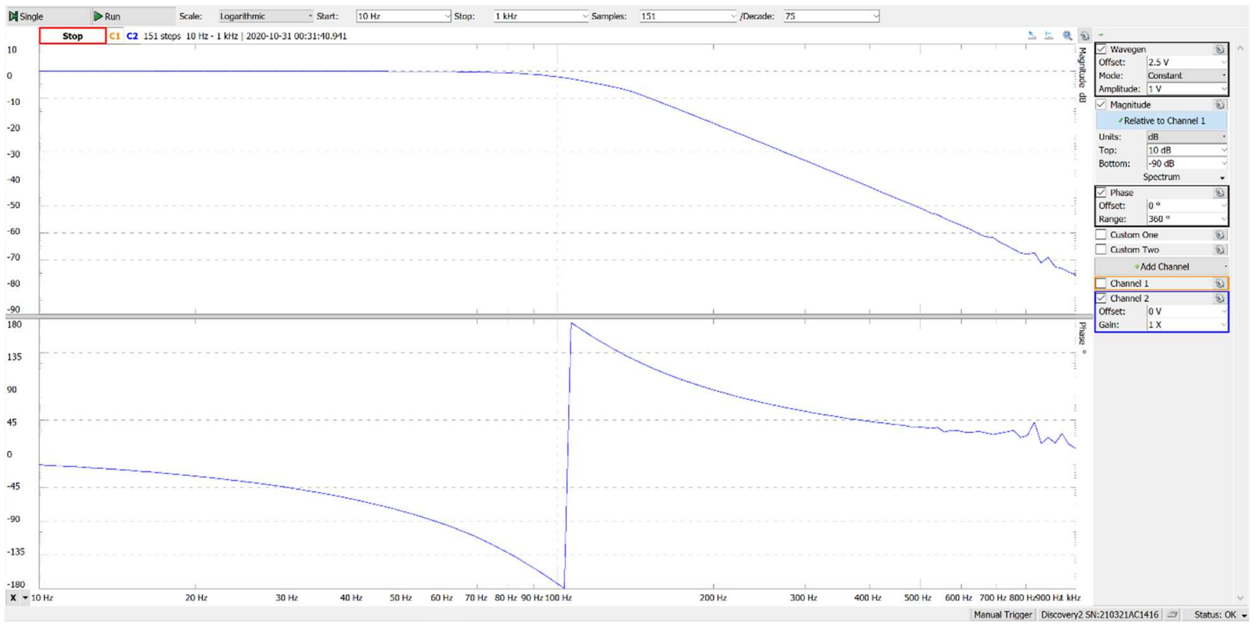


Figure 23: Analog Signal Processing Hardware Test Plan

First, the DC voltage generated by the Vmid buffer was verified, which can also be seen in Appendix A. Next, the frequency responses of each antialiasing filter were verified, as shown in Figure 24 and Figure 25.


Scale: Logarithmic Start: 10 Hz Stop: 1 kHz Samples: 151 /Decade: 75

151 steps 10 Hz - 1 kHz | 2020-10-31 00:31:40.941

Waveген
Offset: 2.5 V
Mode: Constant
Amplitude: 1 V

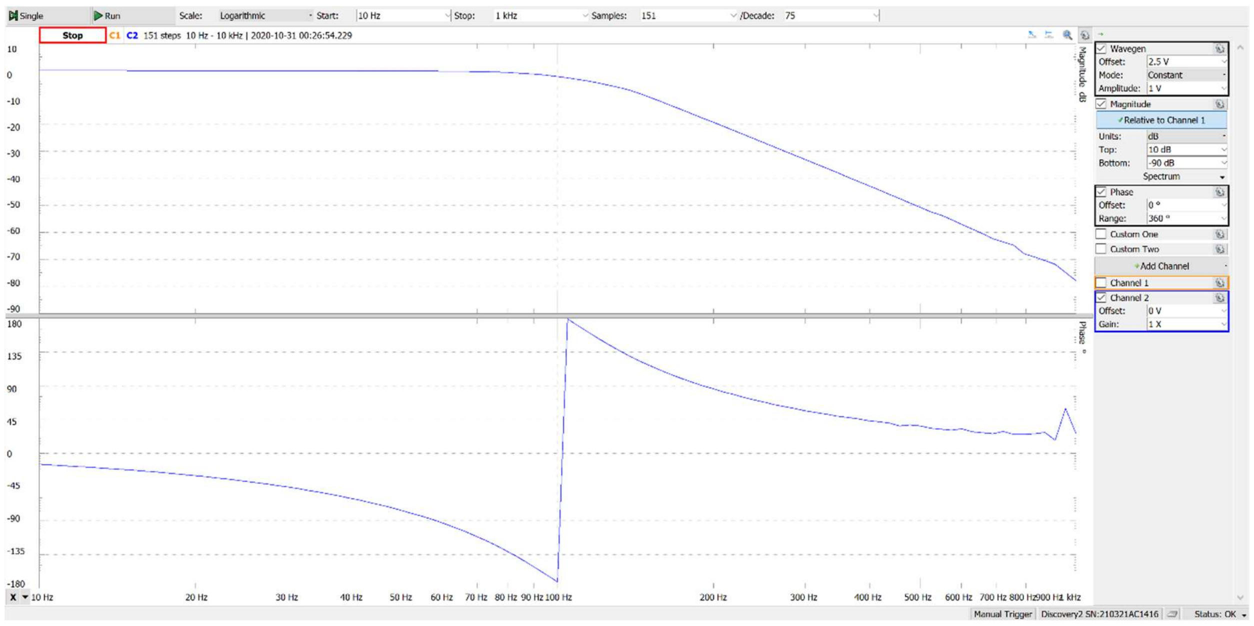
Magnitude
Relative to Channel 1
Units: dB
Top: 10 dB
Bottom: -90 dB

Phase
Offset: 0 °
Range: 360 °

Channel 2
Offset: 0 V
Gain: 1 X

Manual Trigger Discovery2 SN:210321AC1416 Status: OK

Figure 24: Filter 1 Frequency Response Verification


Scale: Logarithmic Start: 10 Hz Stop: 1 kHz Samples: 151 /Decade: 75

151 steps 10 Hz - 10 kHz | 2020-10-31 00:26:54.229

Wavegen
Offset: 2.5 V
Mode: Constant
Amplitude: 1 V

Magnitude
Relative to Channel 1
Units: dB
Top: 10 dB
Bottom: -90 dB

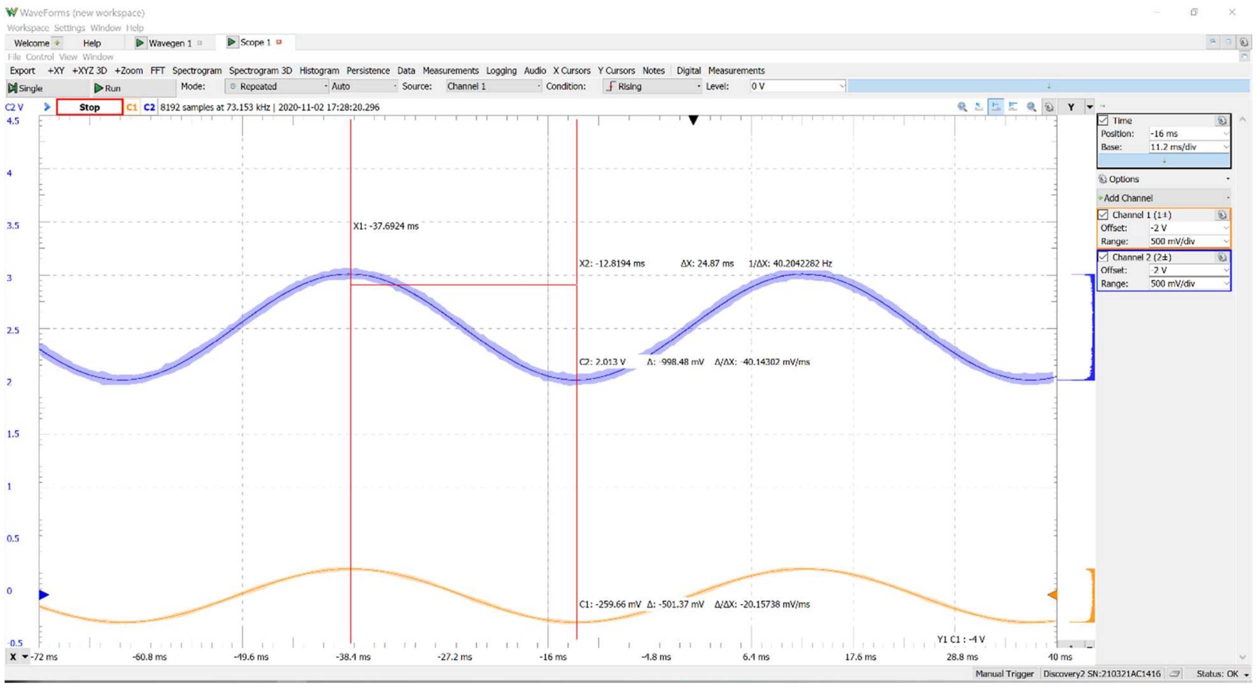
Phase
Offset: 0 °
Range: 360 °

Channel 2
Offset: 0 V
Gain: 1 X

Manual Trigger Discovery2 SN:210321AC1416 Status: OK

Figure 25: Filter 2 Frequency Response Verification

The filters reach an attenuation of approximately -80 dB at 1 kHz. Figure 26 and Figure 27 show the verification of the instrumentation amplifier behavior. The gain was set to 2 and a 500 mV sine wave was inputted into one input while the other input was grounded. It can be seen that the output from the instrumentation amplifiers has the correct amplitude and is centered at the DC voltage offset from the Vmid generator.



8192 samples at 73.153 kHz | 2020-11-02 17:28:20.296

Mode: Repeated Auto Source: Channel 1 Condition: Rising Level: 0 V

Time Position: -16 ms Base: 11.2 ms/div

Channel 1 (1±) Offset: -2 V Range: 500 mV/div

Channel 2 (2±) Offset: -2 V Range: 500 mV/div

X1: -37.6924 ms

X2: -12.8194 ms ΔX: 24.87 ms 1/ΔX: 40.2042282 Hz

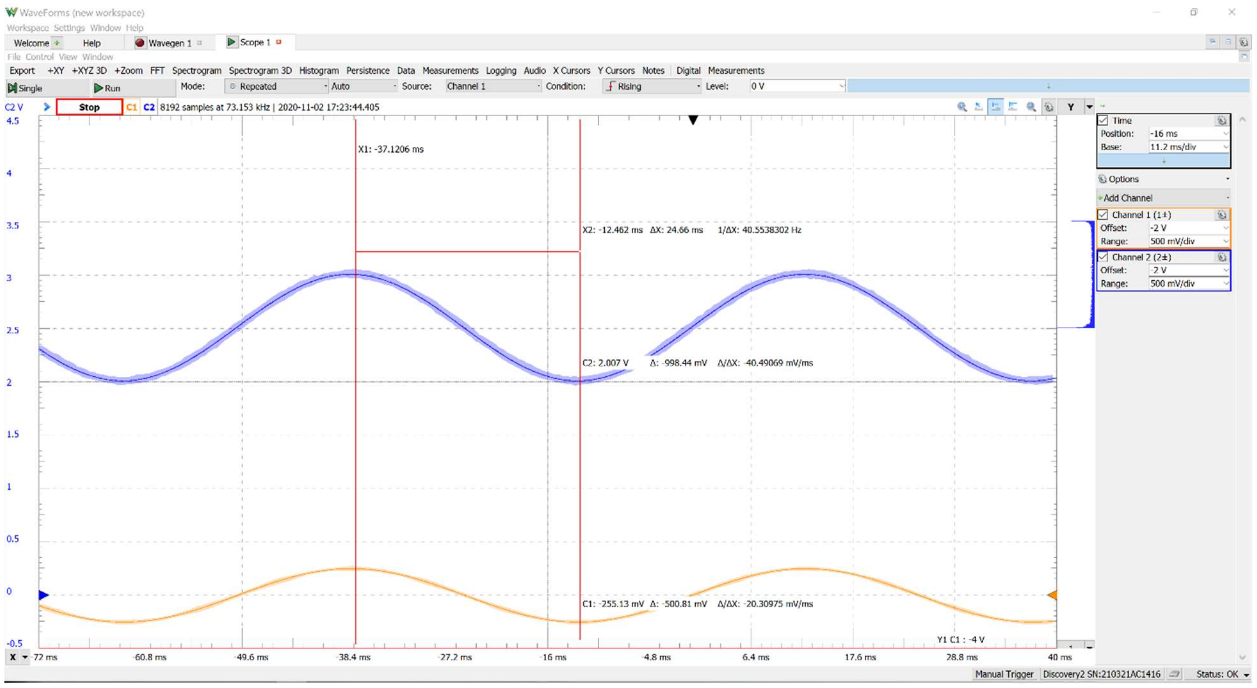
C2: 2.013 V Δ: -998.48 mV Δ/ΔX: 40.14302 mV/ms

C1: -259.66 mV Δ: -501.37 mV Δ/ΔX: -20.15738 mV/ms

Y1 C1 : -4 V

Manual Trigger Discovery2 SN:210321AC1416 Status: OK

Figure 26: Instrumentation Amplifier 1 Output Verification



8192 samples at 73.153 kHz | 2020-11-02 17:23:44.405

Mode: Repeated Auto Source: Channel 1 Condition: Rising Level: 0 V

Time Position: -16 ms Base: 11.2 ms/div

Channel 1 (1±) Offset: -2 V Range: 500 mV/div

Channel 2 (2±) Offset: -2 V Range: 500 mV/div

X1: -37.1206 ms

X2: -12.462 ms ΔX: 24.66 ms 1/ΔX: 40.5538302 Hz

C2: 2.007 V Δ: -998.44 mV Δ/ΔX: -40.49069 mV/ms

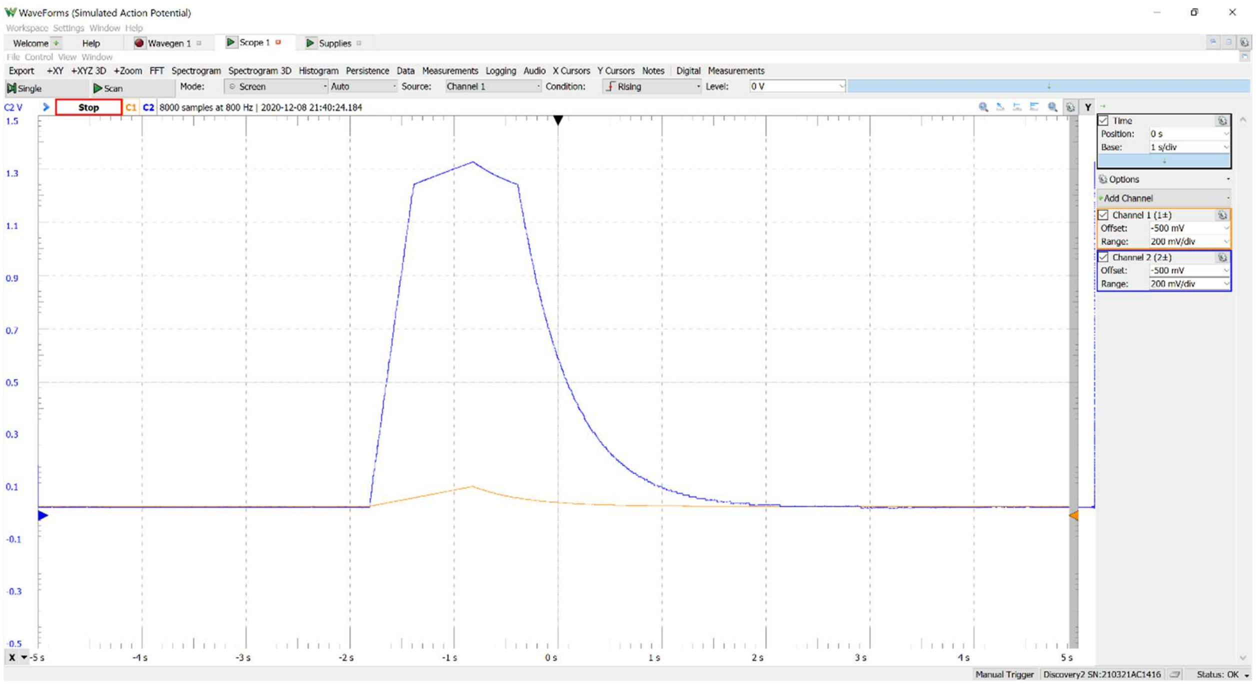
C1: -255.13 mV Δ: -500.81 mV Δ/ΔX: -20.30975 mV/ms

Y1 C1 : -4 V

Manual Trigger Discovery2 SN:210321AC1416 Status: OK

Figure 27: Instrumentation Amplifier 2 Output Verification

The instrumentation amplifier was also verified using the simulated mimosa action potential as well. Figure 28 shows the complete signal chain of the simulated waveform used in the video demo of the project. It can be seen in Figure 28 that the output is slightly distorted due to an error in the design of the offset voltage generator, but this was able to be solved in software by decreasing the ADC wakeup threshold voltage.


WaveForms (Simulated Action Potential)

Mode: Screen, Auto Source: Channel 1 Condition: Rising Level: 0 V

8000 samples at 800 Hz | 2020-12-08 21:40:24.184

Time — Position: 0 s Base: 1 s/div

Channel 1 (1±) — Offset: -500 mV Range: 200 mV/div

Channel 2 (2±) — Offset: -500 mV Range: 200 mV/div

Manual Trigger Discovery2 SN:210321AC1416 Status: OK

Figure 28: Plant Signal Chain Verification

Once the PCB hardware was verified, the existing robot chassis hardware was verified using the steps shown in Figure 29 below, and the sensors and communications test plans were verified using the steps shown in Figure 30 and Figure 31, respectively.

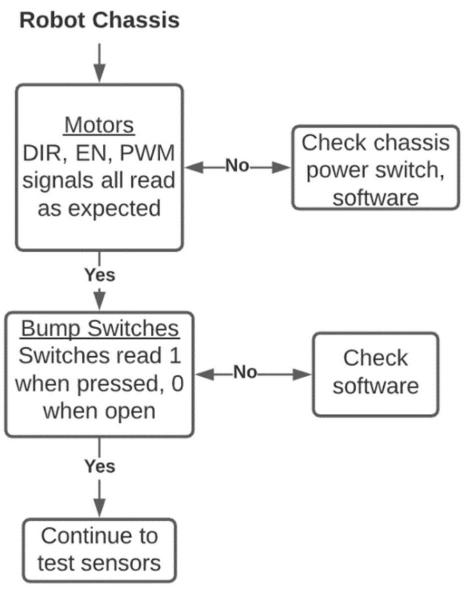

Robot Chassis
Motors
DIR, EN, PWM signals all read as expected
No
Check chassis power switch, software
Yes
Bump Switches
Switches read 1 when pressed, 0 when open
No
Check software
Yes
Continue to test sensors

Figure 29: Robot Chassis Hardware Test Plan

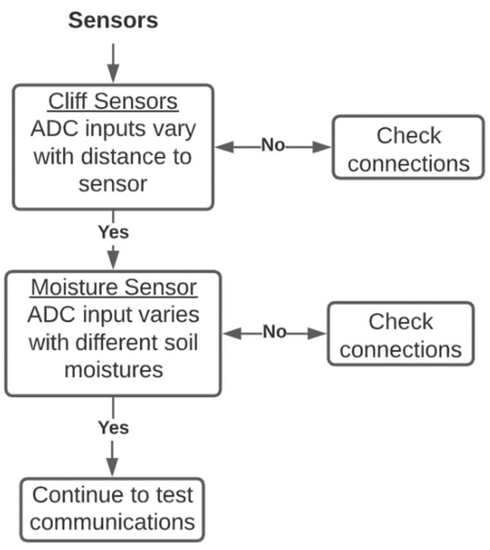

Sensors
Cliff Sensors
ADC inputs vary with distance to sensor
No
Check connections
Yes
Moisture Sensor
ADC input varies with different soil moistures
No
Check connections
Yes
Continue to test communications

Figure 30: Sensors Test Plan

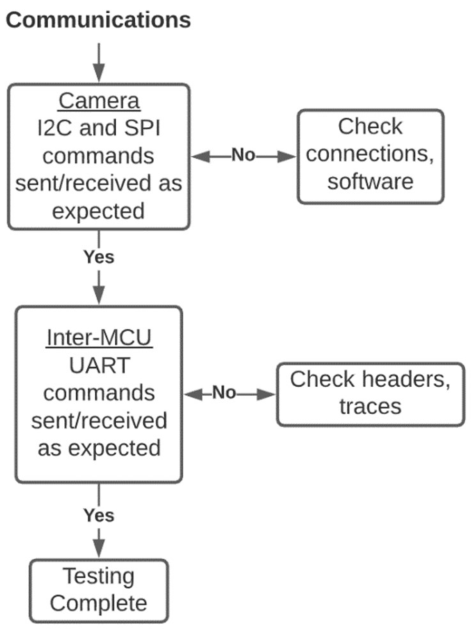

Communications
Camera
I2C and SPI commands sent/received as expected
No
Check connections, software
Yes
Inter-MCU
UART commands sent/received as expected
No
Check headers, traces
Yes
Testing Complete

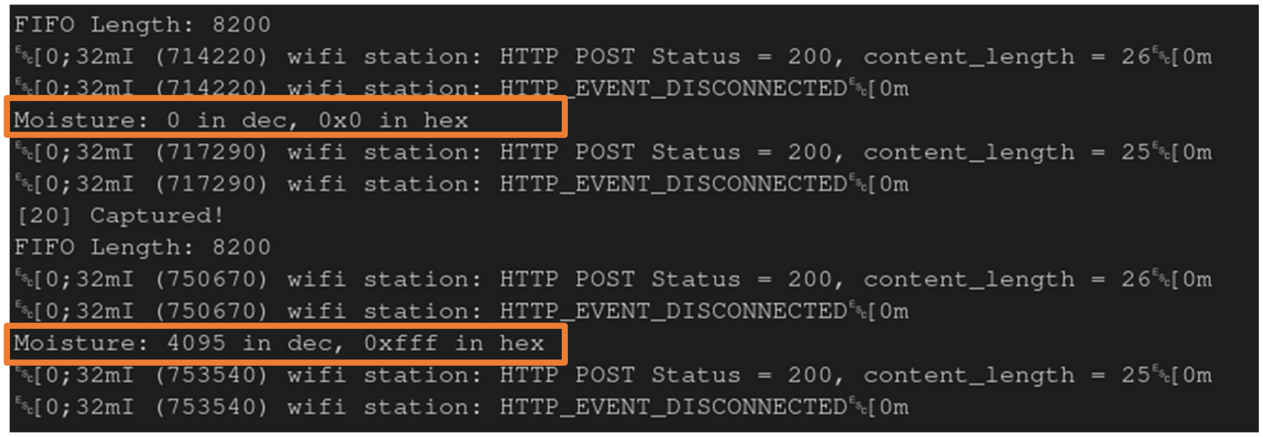
Figure 31: Communications Test Plan

The robot chassis hardware, sensor hardware, and communications hardware were all tested alongside the firmware detailed in the following section.

Firmware

The firmware was tested with the TI-RSLK MAX base, which is faceted with an array of bump switches and cliff sensors used to detect ledges. The control algorithm was first tested incrementally by imitating the expected lighting data and using IR distance sensors as cliff sensors in lieu of hardware that was in the process of being shipped. This helped identify any errors in logic and ensure that the task handler would assign correct priority to the bump switches, the cliff sensors, and lighting data. When needed hardware arrived, the less bulky reflectance sensors replaced the IR distance sensors, and it was discovered that the reflectance sensors did not have the scalability needed for our project due to constant calibration requirements. As such, IR distance sensors were reintegrated into the system for the final design. This is explained in further detail in the Detailed Technical Descriptions: Problems and Design Modifications section of the report. Next, the light sensors were integrated into the control algorithm, replacing previously imitated data, and the algorithm was tuned to match the sensor data. Finally, the firmware associated with the MSP432 was tested in the following set-up configurations: light source placed to the right of the system, light source placed to the left of the system, and light source placed behind the system. All aforementioned tests passed and results are demonstrated in the submitted video.

The ESP32 firmware first involved testing the moisture sensor. The moisture sensor was first connected to the ESP32 and then ADC data was collected from the sensor. The moisture sensor was tested in ranges of very dry to very wet soil to get a gauge of the associated ADC readings. It was confirmed that a high value correlated with a wet soil value and a low ADC reading correlated with a dry soil value. Figure 32 below shows the moisture data being read in through the ESP32. From these tests a threshold value was determined regarding when a user would be notified that their Robotany needs to be watered.

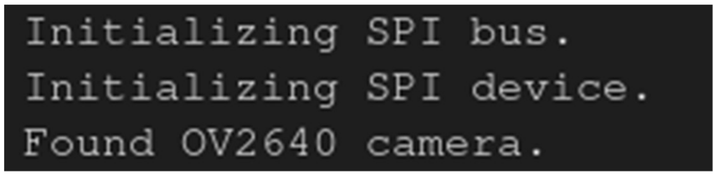


```
FIFO Length: 8200
␛[0;32mI (714220) wifi station: HTTP POST Status = 200, content_length = 26␛[0m
␛[0;32mI (714220) wifi station: HTTP_EVENT_DISCONNECTED␛[0m
Moisture: 0 in dec, 0x0 in hex
␛[0;32mI (717290) wifi station: HTTP POST Status = 200, content_length = 25␛[0m
␛[0;32mI (717290) wifi station: HTTP_EVENT_DISCONNECTED␛[0m
[20] Captured!
FIFO Length: 8200
␛[0;32mI (750670) wifi station: HTTP POST Status = 200, content_length = 26␛[0m
␛[0;32mI (750670) wifi station: HTTP_EVENT_DISCONNECTED␛[0m
Moisture: 4095 in dec, 0xfff in hex
␛[0;32mI (753540) wifi station: HTTP POST Status = 200, content_length = 25␛[0m
␛[0;32mI (753540) wifi station: HTTP_EVENT_DISCONNECTED␛[0m
```

Figure 32: Moisture Sensor Verification

As previously discussed, the camera sensor driver was tested throughout development in order to ensure that it behaved as desired for the computer vision requirements for the project. As such, the camera driver was tested to ensure it would read out valid JPEG images at the specified size. The results of each image were uploaded to the server and are displayed within the application, as demonstrated in the mobile application section. A simple attempt to write into the FIFO buffer over SPI was used to check for the presence of the buffer as a test before any image requests were performed. If the FIFO buffer responds with the values written, the sensor is then checked using the camera's I2C interface by reading in the model number. If the model number matches, the initialization was successful. Otherwise, the camera cannot be used. Next, several bytes were to uploaded to the camera to initialize and configure the sensor. The necessary instructions are stored as an array and are listed as standard across OV2640 modules.

Images in 640x480 pixel JPEG format were used for testing by uploading an array of the proper bytes to set this particular configuration. Throughout this process, an Analog Discovery 2 was used to intercept and confirm that the uploaded configuration and received data matched the expected behavior. Once initialized, it was possible to start reading images from the camera using the FIFO buffer. Figure 33 shows that the software was able to read the camera module's FIFO buffer and model number and fully initializing the device, verifying that the camera is functional on each run of the device.



```
Initializing SPI bus.
Initializing SPI device.
Found OV2640 camera.
```

Figure 33: Initialization of OV2640

Computer Vision and Growth Prediction

The computer vision algorithms and machine learning models were tested as a part of the IoT software test plan. The computer vision algorithms can be divided into two categories: glare removal and background removal. These subcategories were first tested separately by passing in various images. It was seen that the glare removal struggled to create an effective mask on brighter backgrounds, as such the algorithms were then combined, with background removal occurring before glare removal. The same images were passed in for testing, and it was seen that the glare removal algorithm was more robust, as is seen in Figure 34 below, in which the left-most figure is the original image, the center figure is after background removal, and the final image is after glare removal.

The machine learning models were tested in a similar fashion, in which a Django test application was quickly created to allow a user to upload various images and see what the models would predict. This allowed for various iteration of the model to be tested to determine whether the dataset used for training needed to be built out to handle more testcases or if the architecture of the model needed to be updated. Originally a single model was used for predicting growth levels, but not only was the model large, it did not prove to be as effective. As a result, the model was broken down into two separate trained models: one to detect whether enough growth has occurred to place electrodes and one to determine if enough growth has taken place for a plant to be harvested. The machine learning models were tested with images of various plants taken from the system camera. This ensures that the view and the quality of the image used in testing is standardized against what will be used in the finalized system. A result of this test can be seen in Figure 35, in which the dying *Mimosa pudica* plant was captured from above as seen in the left most image, passed through background and glare removal, the number of green pixels in extracted as shown in the center image, and then that value is passed into the neural network for classification as seen in the right most image. These tests were conducted using a variety of plants at different stages of growth and varying soil compositions with a high degree of success.

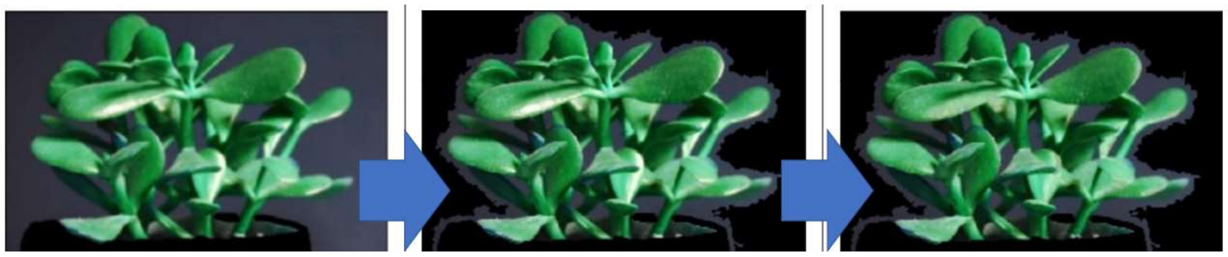


Figure 34: Combined Background and Glare Removal Test Results

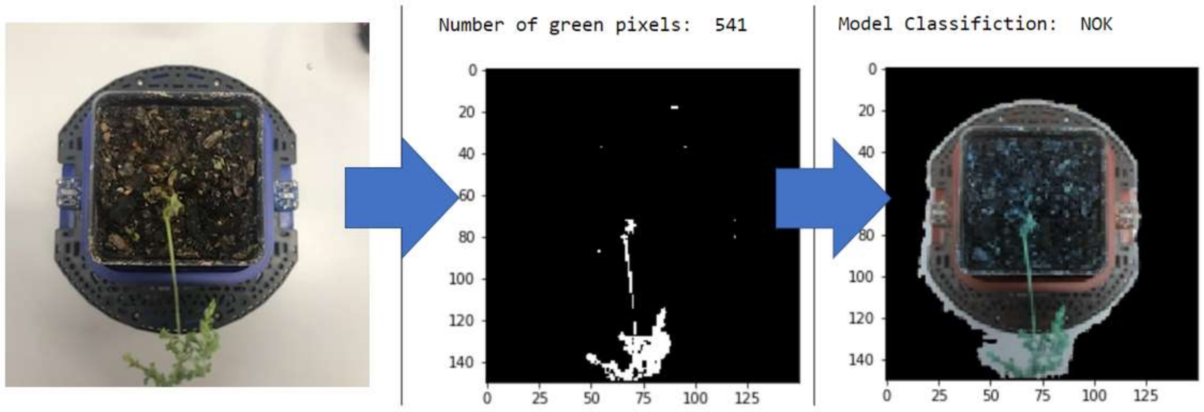


Figure 35: CV Algorithms and Electrode Growth Detection Model Results

IoT Software

Additionally, the IoT test plan included a testing strategy fir both the Wi-Fi provisioning and API portions of the project. The API testing procedure, shown in Figure 36 and Figure 40, focused on ensuring the proper configuration and storage/access behavior for the plant-side upload API and application retrieval API endpoints, respectively.

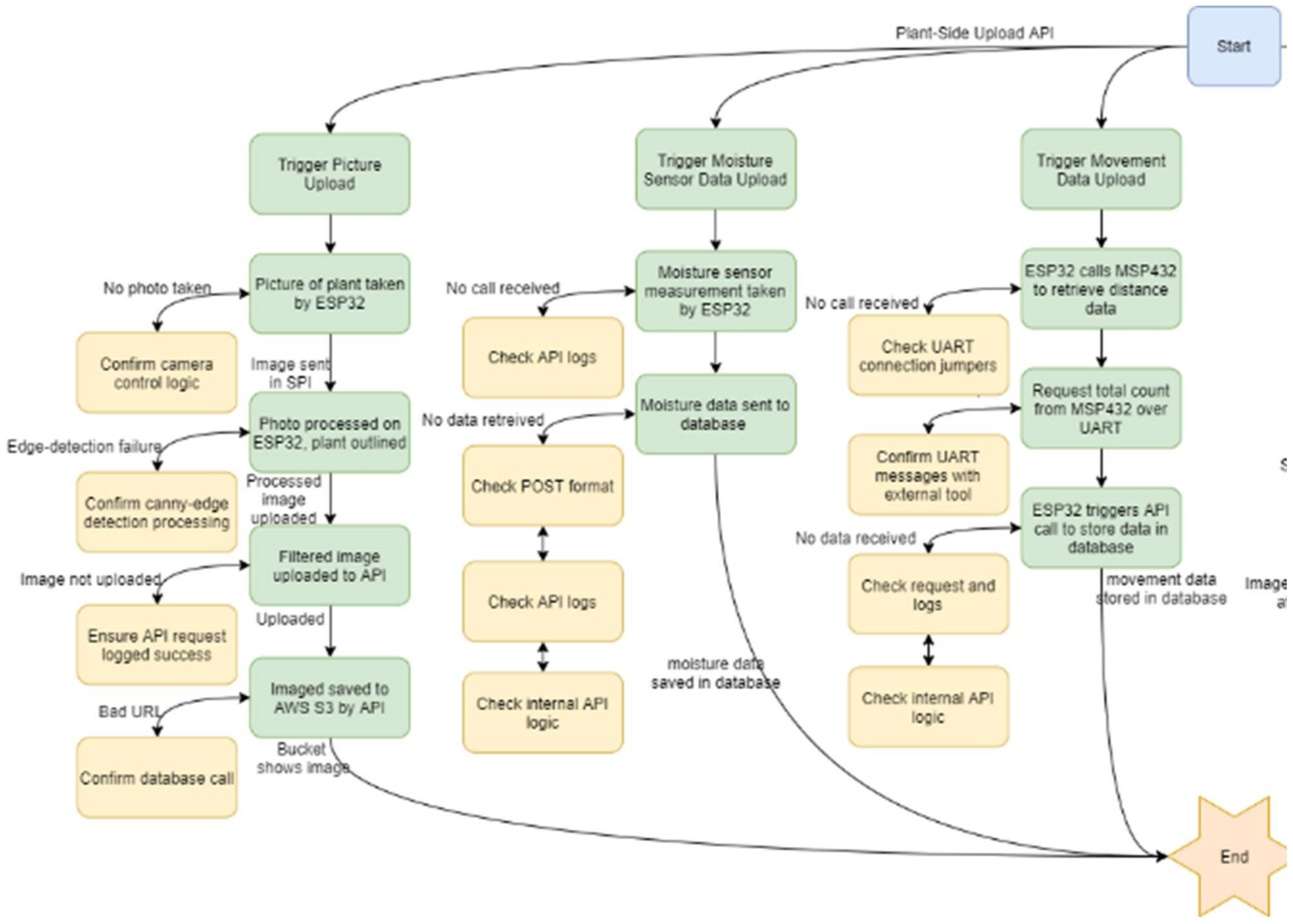
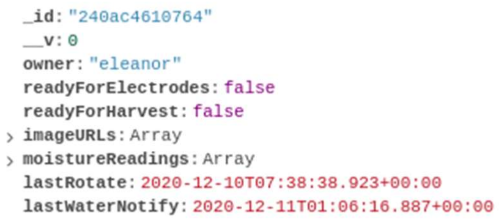

Plant-Side Upload API
Start
Trigger Picture Upload
Trigger Moisture Sensor Data Upload
Trigger Movement Data Upload
No photo taken
Picture of plant taken by ESP32
Confirm camera control logic
Image sent in SPI
Edge-detection failure
Photo processed on ESP32, plant outlined
Confirm canny-edge detection processing
Processed image uploaded
Image not uploaded
Filtered image uploaded to API
Ensure API request logged success
Uploaded
Bad URL
Imaged saved to AWS S3 by API
Confirm database call
Bucket shows image
No call received
Moisture sensor measurement taken by ESP32
Check API logs
No data retreived
Moisture data sent to database
Check POST format
Check API logs
Check internal API logic
moisture data saved in database
No call received
ESP32 calls MSP432 to retrieve distance data
Check UART connection jumpers
Request total count from MSP432 over UART
Confirm UART messages with external tool
No data received
ESP32 triggers API call to store data in database
Check request and logs
Check internal API logic
movement data stored in database
End

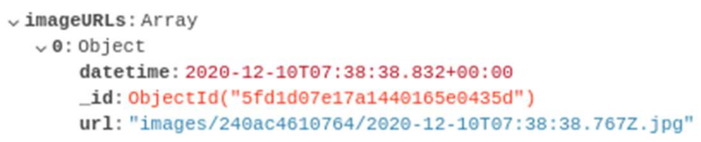
Figure 36: Plant-Side Upload API

Both the image upload and moisture data upload portions of the plant-side API test-plan were tested and passed, with some being demonstrated as part of firmware development. This was confirmed upon inspection of the database, where the triggered and uploaded data was retrieved. This is demonstrated by the database records shown in Figure 37, Figure 38, and Figure 39. Upon consideration of time constraints and several other feasibility concerns, the movement data portion was scrapped, and was not implemented. Otherwise, all intended information that the plant should upload was provided.



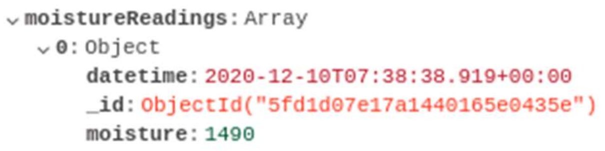
```
  _id: "240ac4610764"
  __v: 0
  owner: "eleanor"
  readyForElectrodes: false
  readyForHarvest: false
> imageURLs: Array
> moistureReadings: Array
  lastRotate: 2020-12-10T07:38:38.923+00:00
  lastWaterNotify: 2020-12-11T01:06:16.887+00:00
```

Figure 37: Overall Record



```
⌄imageURLs: Array
  ⌄0: Object
      datetime: 2020-12-10T07:38:38.832+00:00
      _id: ObjectId("5fd1d07e17a1440165e0435d")
      url: "images/240ac4610764/2020-12-10T07:38:38.767Z.jpg"
```

Figure 38: Record of Image



```
⌄moistureReadings: Array
  ⌄0: Object
      datetime: 2020-12-10T07:38:38.919+00:00
      _id: ObjectId("5fd1d07e17a1440165e0435e")
      moisture: 1490
```

Figure 39: Record of Moisture Reading

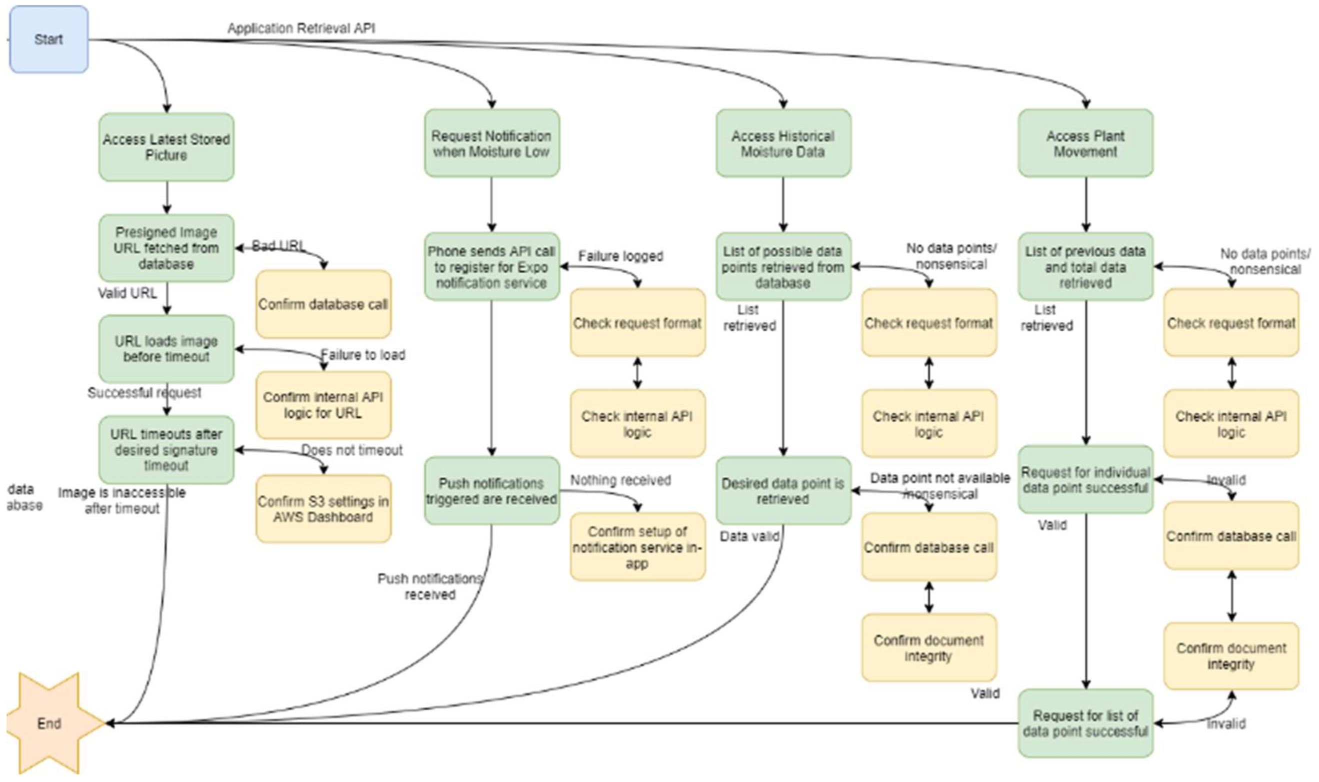
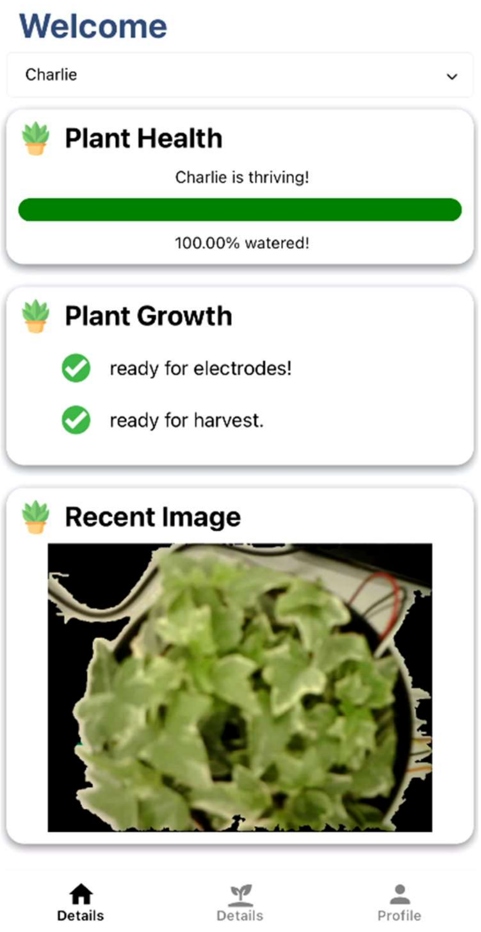

Start
Application Retrieval API
Access Latest Stored Picture
Presigned Image URL fetched from database
Bad URL
Confirm database call
Valid URL
URL loads image before timeout
Failure to load
Confirm internal API logic for URL
Successful request
URL timeouts after desired signature timeout
Does not timeout
Confirm S3 settings in AWS Dashboard
data abase
Image is inaccessible after timeout
Request Notification when Moisture Low
Phone sends API call to register for Expo notification service
Failure logged
Check request format
Check internal API logic
Push notifications triggered are received
Nothing received
Confirm setup of notification service in-app
Push notifications received
Access Historical Moisture Data
List of possible data points retrieved from database
No data points/ nonsensical
Check request format
Check internal API logic
List retrieved
Desired data point is retrieved
Data point not available nonsensical
Confirm database call
Confirm document integrity
Data valid
Access Plant Movement
List of previous data and total data retrieved
No data points/ nonsensical
Check request format
Check internal API logic
List retrieved
Request for individual data point successful
Invalid
Confirm database call
Confirm document integrity
Valid
Valid
Request for list of data point successful
Invalid
End

Figure 40: Application Retrieval API

Historical moisture data and photo download were configured properly and tested according to the application retrieval test plan. The data can be confirmed by viewing the mobile application and tracing API responses, as was structured in the test plan, though the database entries for the user confirms the linkage to the plant data, consistency confirmed with a schema analysis. Notifications were configured as Figure 40: once the user logs in and allows notification permissions, a push notification token is passed to the database. The API call was tested on the application side and confirmed in the database. Several test notifications were configured (and received on the user's phone), and a timed notification system was configured and confirmed as the final step. The culmination of all this work is shown in the screenshot of the mobile application dashboard, Error! Reference source not found., and the moisture history chart in the details screen, Figure 42.


Welcome

Charlie

Plant Health

Charlie is thriving!

100.00% watered!

Plant Growth

ready for electrodes!

ready for harvest.

Recent Image

Figure 41: Robotany Application Home Screen

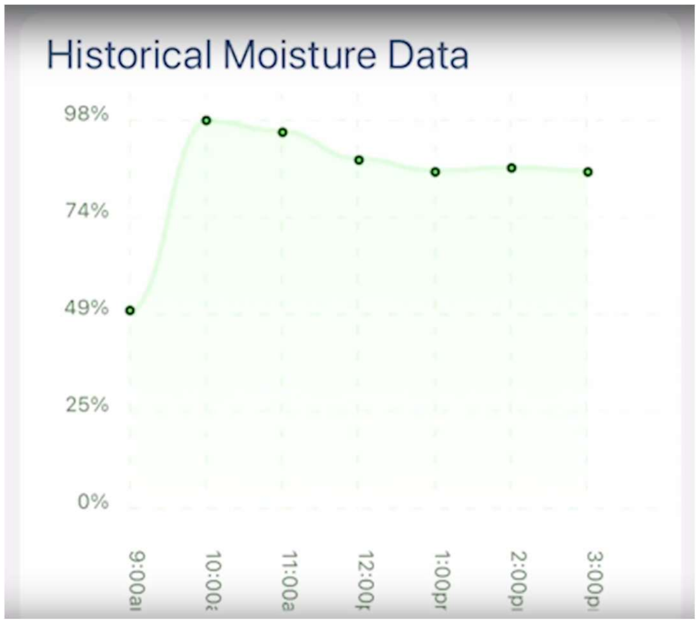


Figure 42: Robotany Application Moisture Data Display

Final Results

The final results for the Robotany included a functioning mobile plant robot that is capable of repositioning itself to the location of greatest light. Due to the difficulties experienced in sourcing a healthy mimosa plant, the signal that was to be measured from the plant was simulated instead using the function generators on the Analog Discovery 2. Each of the conventional sensors, including the cliff sensors, bump switches, light sensors, camera, and moisture sensor, all successfully collect and utilize data to control the robot's motion or provide information to the end user. As seen in the video demo, the bump and cliff sensors successfully prevent the Robotany from crashing into objects or driving off of a ledge. Further, the moisture sensor is able to log the moisture data over time and send notifications through the mobile application when the plant has low moisture. The computer vision model receives the image from the camera, removes glare and replaces the image background with a black mask, and classifies the plant growth level. As such, the finalized Robotany Product satisfies the proposed functionality shown in Table 1, with the only deviation being that the reconstructed *Mimosa pudica* signal was read in through a function generator rather than the plant itself, because of aforementioned sourcing issues. Even despite this difficulty, the functionality achieved in the system places the product in the A grade range as shown in Table 2: Points to Letter Grade.

Table 1: Grade Rubric

	Measuring Signals from Plant	**Using Moisture Sensor**	**Bump and Cliff Sensors**	**Mobile App Communication**	**Plant Boundary/ Growth Detection**

2	Can move plant based on the signals	Can indicate when moisture is low in pot	Prevent crashes with both	Can provide details and notify user	Can determine when a plant has reached some size
1	Can read signals from plant	Can read moisture level	Prevent crashes with one	Can either notify or give data	Images plant and can provide image
0	Unable to use plant signals	Unable to determine moisture level	Crashes	Cannot communicate with app	No growth detection

Table 2: Points to Letter Grade

Points	**Grade**
8-10	A
5-7	B
2-5	C
0-2	D

Further, the team had a few additional goals that were not met. The proposed project aimed to measure signals from various plants, rather than just the *Mimosa pudica* that has larger action potentials than most other house plants. This feature would allow the product to be more marketable to the potential customer base because they could grow a more diverse array of plants. The application also had the intention of allowing the user to select the specific species of plant that would be grown. This would allow the growth prediction model for determining harvest only used when most appropriate. It would further allow the models to be built out for more scalable growth prediction, as the model takes in the number of green pixels within a range of light to dark green to predict growth classes. However, adding the functionality for the user to specify which plant they are growing would enable the model used to be more customized. For example, a common house plant that does not showcase a high degree of greenery is the *Fittoniaa albivenis* (pink angel). If the system were to be aware of this information, it could rather predict growth on the amount of pin seen in the image.

The final result of the Robotany has demonstrated the value in determining the core functionality of a system. The additional features described above are an important part of making the system optimal, but expectations must be set in accordance with the core functionality of the system. Another key lesson from this project is how important communication is among team members. As soon as potential issues were determined with sourcing a *Mimosa Pudica* plant, it was communicated to all team members so that an ideal solution could be implemented. This enabled the final system to be demonstrated using reconstructed plant signals, and allowed the team to meet end goals.

Costs

The cost to produce a prototype of the Robotany this semester was significantly lower than a production model would cost thanks to Professor Dugan, who donated the TI-RSLK MAX robot and the MSP432 LaunchPad that were used. Table 3 shows a high-level breakdown of the total costs if the Robotany was produced in limited and large quantities.

Table 3: Robotany Costs

	Cost for 1 unit	Cost per unit for 10,000 units
TI-RSLK MAX Robot Chassis	$99.95	$77.83
MSP432	$23.99	$4.26
PCB	$33	$4.75
PCB Components	$26.43	$17.83
Camera	$25.99	$15.99
Total	$209.36	$120.66

It can be seen in Table 3 that the component contributing the most to the cost of the Robotany is the TI-RSLK MAX robot platform. If 10,000 units were to be produced, the total cost decreases by 42.37%, from $209.36 to $120.66. If this project were to be mass produced, designing and producing a new robot chassis would likely decrease the costs significantly. The Robotany also uses two separate microcontrollers due to the use of the existing robot chassis as well. A larger-scale production of this project could further decrease costs by consolidating the embedded system to run on a single microcontroller. The detailed breakdown of the component costs can be seen in

PhotoSynthesis-Server.zip

ESP32 Package

Robotany-ESP-IDF.zip

Mobile Application Package

RobotanyReactApp.zip

Appendix D.

The PCB designed for the Robotany has very few components, so automated equipment would likely not have a considerable impact on the costs. Depending on how a redesigned robot chassis would be implemented, the assembly of the Robotany could likely benefit from automated equipment.

Future Work

The team suggests that the implementation of the missing secondary features described in the Final Results section would improve the current version of the Robotany is such a way that brings it closer to its end goals: a proof of concept for larger scale agriculture or sensor network applications. These future improvements are decomposed into advancements for biosensor and computer vision applications.

Improvements for Biosensor Applications

Using plants as biosensors can have applications in both agriculture and sensor networks. The Robotany attempts to utilize the action potentials generated by the *Mimosa pudica* when opening and closing as a wakeup signal for the control algorithm, which receives data from conventional sensors to relocate the plant to the position of brightest sunlight. If more sensitive hardware were to be implemented, future work for the Robotany could include utilizing the phototropic (bending towards or away from light) response of plants. [2] reports that an action potential was measured in a soybean plant as a response to moving the location of a light, and this could be used as an input to the control algorithm to allow the plant to choose the direction of travel. However, this study characterizes these light-induced action potentials as an "all-or-nothing" response, meaning that the amplitude does not vary based on the intensity of light, so conventional light sensors would likely still be necessary to indicate when the robot should stop moving.

A larger and more impactful use of biosensors is in agricultural systems. Plants also exhibit measurable electrochemical responses to negative stimuli such as wounding from insects harming the leaves and environmental contaminants [2]. A large-scale implementation of biosensing in an agricultural system could use signals measured from the plants themselves to detect any potentially harmful agents in the environment automatically, without requiring frequent human inspections of problems such as pests and soil quality. The Robotany attempts to implement a more basic, consumer-level demonstration of using plants as biosensors, but more meaningful applications of the technology can be implemented in the future as well.

Improvements for Computer Vision Applications

Robots developed for agricultural purposes can analyze health of a crops, detect weeds, determine the amount of time needed for crop harvesting, and determine when a crop is ready for harvest [74]. While the current system uses the number of green pixels seen in an image to determine when a green plant is ready for electrode placement or harvest, this limits the growth prediction model's scalability to green plants. As such, the future system should rather take in the type of plant and select an electrode or harvest model accordingly. For non-green plants, such as mushrooms, our system would improperly classify its growth. However, if a model were trained for each class of specific plants, it would be a more scalable system. This could further be applied beyond our product and into larger scale agriculture systems to help drones or other sorts of agriculture robots inform their user that their crops are ready for harvest. The development of these additional models would be simple, as the basic architecture of the algorithm is the same, with only changing the range of color that is passed into the model and the dataset the model is trained on.

Improvements for Mobile Application

While the mobile application can efficiently connect the user to the plant, display plant data, and notify the user of events, it does not provide much additional functionality. If this product were to go into large-scale production, more features should be implemented to improve customer satisfaction. These features might include movement tracking, plant-specific watering suggestions, plant-specific sunlight options, better data representation options, and more historical data access. These features would not be difficult to implement and would significantly improve the user experience.

References

[1] L. Boone, "Must Reads: They don't own homes. They don't have kids. Why millennials are plant addicts," *LA Times*, Jul. 24, 2018.

[2] A. Volkov and D. R. Ranatunga, "Plants as Environmental Biosensors," *Plant Signal. Behav.*, vol. 1, no. 3, Jun. 2006.

[3] G. Gage, "Transcript of 'Electrical experiments with plants that count and communicate,'" *TED*, Apr. 2017. https://www.ted.com/talks/greg_gage_electrical_experiments_with_plants_that_count_and_communicate/transcript (accessed Sep. 08, 2020).

[4] W. Cai and Q. Qi, "Study on Electrophysiological Signal Monitoring of Plant under Stress Based on Integrated Op-Amps and Patch Electrode," *Journal of Electrical and Computer Engineering*, Oct. 10, 2017. https://www.hindawi.com/journals/jece/2017/4182546/ (accessed Sep. 08, 2020).

[5] "The Plant SpikerBox," *Backyard Brains*. https://backyardbrains.com/products/PlantSpikerBox (accessed Sep. 09, 2020).

[6] H. Sareen, "Cyborg botany : augmented plants as sensors, displays and actuators," 2017.

[7] "Phytosensing and phytoactuating system," *Cybertronica Research*. http://cybertronica.co/?q=products/phytosensor (accessed Sep. 08, 2020).

[8] "myRIO-1900 User Guide and Specifications - National Instruments," p. 32.

[9] "MSP-EXP432P401R Texas Instruments | Development Boards, Kits, Programmers | DigiKey." https://www.digikey.com/product-detail/en/texas-instruments/MSP-EXP432P401R/296-39653-ND/5170609 (accessed Sep. 08, 2020).

[10] "ESP32-WROOM-32D Espressif Systems | RF/IF and RFID | DigiKey." https://www.digikey.com/product-detail/en/espressif-systems/ESP32-WROOM-32D/1904-1023-1-ND/9381732 (accessed Sep. 08, 2020).

[11] "Amazon.com: HiLetgo FT232RL FTDI Mini USB to TTL Serial Converter Adapter Module 3.3V 5.5V FT232R Breakout FT232RL USB to Serial Mini USB to TTL Adapter Board for Arduino: Computers & Accessories." https://www.amazon.com/HiLetgo-FT232RL-Converter-Adapter-

Breakout/dp/B00IJXZQ7C/ref=sr_1_43_sspa?dchild=1&keywords=ftdi+programmer&qid=1607560179&sr=8-43-spons&psc=1&spLa=ZW5jcnlwdGVkUXVhbGlmaWVyPUFDNFo5QTZROUFVUDQmZW5jcnlwdGVkSWQ9QTA3NDE4NzEyWk5CVkxQOUc2MEtWJmVuY3J5cHRlZEFkSWQ9QTAyMzYyODYxMFQ5WjFWWENXSFRZJndpZGdldE5hbWU9c3BfbXRmJmFjdGlvbj1jbGlja1JlZGlyZWN0JmRvTm90TG9nQ2xpY2s9dHJ1ZQ== (accessed Dec. 09, 2020).

[12] "Multisim," *National Instruments*. https://www.ni.com/en-us/shop/software/products/multisim.html (accessed Sep. 13, 2020).

[13] "Ultiboard," *National Instruments*. https://www.ni.com/en-us/shop/software/products/ultiboard.html (accessed Sep. 13, 2020).

[14] "CCSTUDIO_10.1.1.00004 | TI.com." https://www.ti.com/tool/download/CCSTUDIO (accessed Dec. 09, 2020).

[15] PlatformIO, "PlatformIO is a professional collaborative platform for embedded development," *PlatformIO*. https://platformio.org (accessed Dec. 09, 2020).

[16] "Pololu Chassis Kit for TI-RSLK MAX." https://www.pololu.com/product/3670 (accessed Dec. 09, 2020).

[17] "ESP32-WROOM-32D Wi-Fi Certification," *Espressif*, Apr. 25, 2018. https://www.espressif.com/sites/default/files/ESP32-WROOM-32D_Wi-Fi_Certificate.pdf.pdf.

[18] "ESP32-WROOM-32D FCC Wi-Fi BT4.0 Certificate," *Espressif*. https://www.espressif.com/sites/default/files/esp32-wroom-32d_fcc_wi-fi_bt4.0_certificate.pdf (accessed Sep. 15, 2020).

[19] "IPC-2221A.pdf." Accessed: Dec. 09, 2020. [Online]. Available: https://www.ipc.org/TOC/IPC-2221A.pdf.

[20] "IPC-A-600J.pdf." Accessed: Dec. 09, 2020. [Online]. Available: https://www.ipc.org/TOC/IPC-A-600J.pdf.

[21] "SMD Packages: Sizes Dimensions Details » Electronics Notes." https://www.electronics-notes.com/articles/electronic_components/surface-mount-technology-smd-smt/packages.php (accessed Dec. 09, 2020).

[22] "USB Standards: USB 1, USB 2, USB 3, USB 4 » Electronics Notes." https://www.electronics-notes.com/articles/connectivity/usb-universal-serial-bus/standards.php (accessed Dec. 09, 2020).

[23] "Introduction to SPI Interface | Analog Devices." https://www.analog.com/en/analog-dialogue/articles/introduction-to-spi-interface.html# (accessed Dec. 09, 2020).

[24] "UM10204 I2C-bus specification and user manual," vol. 2014, p. 64, 2014.

[25] "Fusion 360," *Autodesk*. https://www.autodesk.com/products/fusion-360/overview (accessed Sep. 13, 2020).

[26] "Cura," *Ultimaker*. https://ultimaker.com/software/ultimaker-cura (accessed Dec. 08, 2020).

[27] "VirtualBench," *National Instruments*. https://www.ni.com/en-us/shop/hardware/products/virtualbench-all-in-one-instrument.html (accessed Dec. 08, 2020).

[28] "Analog Discovery 2," *Digilent*. https://store.digilentinc.com/analog-discovery-2-100msps-usb-oscilloscope-logic-analyzer-and-variable-power-supply/ (accessed Dec. 08, 2020).

[29] “TI-RSLK Max Edition Curriculum.” https://university.ti.com/en/faculty/ti-robotics-system-learning-kit/ti-rslk-max-edition-curriculum (accessed Dec. 09, 2020).
[30] *espressif/esp-idf*. Espressif Systems, 2020.
[31] “Visual Studio Code - Code Editing. Redefined.” https://code.visualstudio.com/ (accessed Dec. 09, 2020).
[32] “Anaconda | Individual Edition,” *Anaconda*. https://www.anaconda.com/products/individual (accessed Dec. 09, 2020).
[33] “Project Jupyter.” https://www.jupyter.org (accessed Dec. 09, 2020).
[34] “Home - pip documentation v20.3.1.” https://pip.pypa.io/en/stable/ (accessed Dec. 09, 2020).
[35] “opencv/opencv,” *GitHub*. https://github.com/opencv/opencv (accessed Dec. 09, 2020).
[36] “Module: tf | TensorFlow Core v1.15.0,” *TensorFlow*. https://www.tensorflow.org/versions/r1.15/api_docs/python/tf (accessed Dec. 09, 2020).
[37] “NumPy.” https://numpy.org/ (accessed Dec. 09, 2020).
[38] “opencv/opencv_extra,” *GitHub*. https://github.com/opencv/opencv_extra (accessed Dec. 09, 2020).
[39] “JavaScript,” *MDN Web Docs*. https://developer.mozilla.org/en-US/docs/Web/JavaScript (accessed Dec. 10, 2020).
[40] “React Native.” https://reactnative.dev/ (accessed Dec. 10, 2020).
[41] “Expo,” *Expo*. https://expo.io/ (accessed Dec. 10, 2020).
[42] “Git.” https://git-scm.com/ (accessed Dec. 10, 2020).
[43] “GitHub Documentation.” https://docs.github.com/en (accessed Dec. 10, 2020).
[44] “Amazon EC2,” *Amazon Web Services, Inc.* https://aws.amazon.com/ec2/ (accessed Dec. 10, 2020).
[45] “Working with Amazon S3 Buckets - Amazon Simple Storage Service.” https://docs.aws.amazon.com/AmazonS3/latest/dev/UsingBucket.html (accessed Dec. 10, 2020).
[46] “NGINX Docs | NGINX Reverse Proxy,” *NGINX Documentation*. https://docs.nginx.com/nginx/admin-guide/web-server/reverse-proxy/ (accessed Dec. 10, 2020).
[47] “Let’s Encrypt,” *Let’s Encrypt*. https://letsencrypt.org/ (accessed Dec. 10, 2020).
[48] Node.js, “Node.js,” *Node.js*. https://nodejs.org/en/ (accessed Dec. 10, 2020).
[49] “MongoDB Atlas: Cloud Document Database,” *MongoDB*. https://www.mongodb.com/cloud/atlas/lp/try2 (accessed Dec. 10, 2020).
[50] “Postman | The Collaboration Platform for API Development,” *Postman*. https://www.postman.com/ (accessed Dec. 10, 2020).
[51] “Compass,” *MongoDB*. https://www.mongodb.com/products/compass (accessed Dec. 10, 2020).
[52] *axios/axios*. axios, 2020.
[53] “FreeRTOS - Market leading RTOS (Real Time Operating System) for embedded systems with Internet of Things extensions,” *FreeRTOS*. /index.html (accessed Dec. 10, 2020).
[54] *espressif/esptool*. Espressif Systems, 2020.
[55] “ArduCAM ESP32 UNO,” *Arducam*. https://www.arducam.com/knowledge-base/arducam-esp32-uno/ (accessed Dec. 10, 2020).

[56] S. Boonsawat, “Proper disposal treatment for printed circuit board waste: the developoment of pyrolysis based recycling technology,” *Int. J. Environ. Eng.*, vol. 1, no. 2, Jun. 2014, Accessed: Sep. 13, 2020. [Online]. Available: https://www.researchgate.net/publication/264117570_Proper_disposal_treatment_for_printed_circuit_board_waste_the_developoment_of_pyrolysis_based_recycling_technology.

[57] E. Chung, “Bioplastics: What they are and why they’re not a perfect solution | CBC News,” *CBC*, Mar. 05, 2020. https://www.cbc.ca/news/technology/bioplastics-backgrounder-1.5485009 (accessed Sep. 08, 2020).

[58] J. F. D. Groot, D. V. der Veen, and E. G. Hempenius, “Plant growth system,” US20160143228A1, May 26, 2016.

[59] A. K. Moon and K. KIM, “Plant observation device and method,” US8947525B2, Feb. 03, 2015.

[60] M. Blaylock, B. Ferguson, V. Chandler, and C. Poynton, “Plant biosensor systems,” US20050114923A1, May 26, 2005.

[61] “DRV8838.” Texas Instruments, Jun. 2016, Accessed: Dec. 08, 2020. [Online]. Available: https://www.ti.com/lit/ds/symlink/drv8838.pdf?ts=1607462835438&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FDRV8838.

[62] “VEML7700.” Vishay, Sep. 20, 2020, Accessed: Dec. 08, 2020. [Online]. Available: https://www.vishay.com/docs/84286/veml7700.pdf.

[63] J. Fromm and S. Lautner, “Electrical signals and their physiological significance in plants,” *Plant Cell Environ.*, vol. 30, no. 3, pp. 249–257, 2007, doi: https://doi.org/10.1111/j.1365-3040.2006.01614.x.

[64] A. G. Volkov, J. C. Foster, and V. S. Markin, “Signal transduction in Mimosa pudica: biologically closed electrical circuits,” *Plant Cell Environ.*, vol. 33, no. 5, pp. 816–827, 2010, doi: https://doi.org/10.1111/j.1365-3040.2009.02108.x.

[65] “AD623.” Analog Devices, 2020, Accessed: Dec. 04, 2020. [Online]. Available: https://www.analog.com/media/en/technical-documentation/data-sheets/AD623.pdf.

[66] “LMC6482.” Texas Instruments, Apr. 2020, Accessed: Dec. 04, 2020. [Online]. Available: https://www.ti.com/lit/ds/symlink/lmc6482.pdf?HQS=TI-null-null-digikeymode-df-pf-null-wwe&ts=1607125347775.

[67] “SEN-13637 SparkFun | Mouser,” *Mouser Electronics*. https://www.mouser.com/ProductDetail/474-SEN-13637 (accessed Dec. 09, 2020).

[68] “OV2640 Camera.” https://www.seeedstudio.com/OV2640-Fisheye-Camera-p-4048.html (accessed Dec. 09, 2020).

[69] “ESP HTTP Client - ESP32 - — ESP-IDF Programming Guide latest documentation.” https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/protocols/esp_http_client.html (accessed Dec. 10, 2020).

[70] Chris, “Image Background Removal using OpenCV (Part 1),” *Medium*, Jun. 21, 2020. https://medium.com/@chris.s.park/image-background-removal-using-opencv-part-1-da3695ac66b6 (accessed Dec. 09, 2020).

[71] S. Rasathurai, “Glare removal With Inpainting[OpenCV Python],” *Medium*, Apr. 06, 2020. https://medium.com/@rcvaram/glare-removal-with-inpainting-opencv-python-95355aa2aa52 (accessed Dec. 09, 2020).

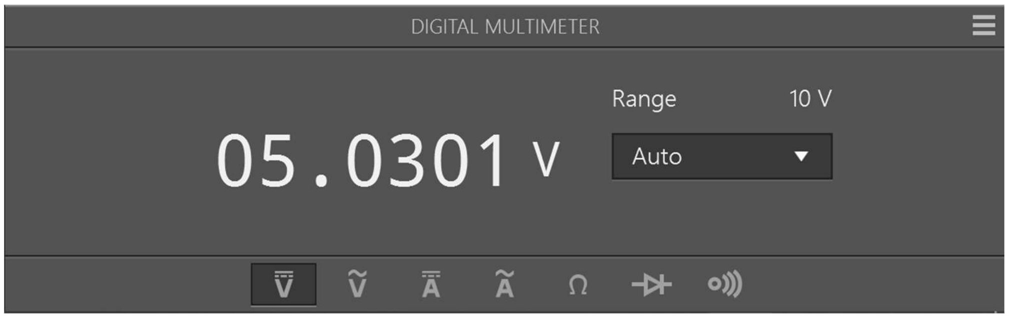
[72] sentdex, *Deep Learning with Python, TensorFlow, and Keras tutorial*. 2018.

[73] "Notifications," *Expo Documentation*. https://docs.expo.io/versions/latest/sdk/notifications/ (accessed Dec. 10, 2020).
[74] V. S. Bisen, "Application of Computer Vision in Precision Agriculture & Farming," *Medium*, Oct. 15, 2020. https://medium.com/vsinghbisen/application-of-computer-vision-in-precision-agriculture-farming-79b0600d5a5d (accessed Dec. 09, 2020).

Appendix

Appendix A

Hardware Test Plan Results

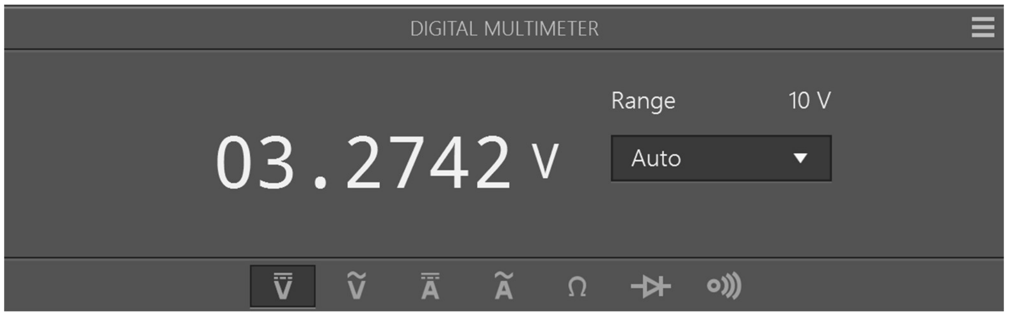

DIGITAL MULTIMETER

05.0301 V

Range 10 V

Auto

Figure 43: LaunchPad 5 V Verification

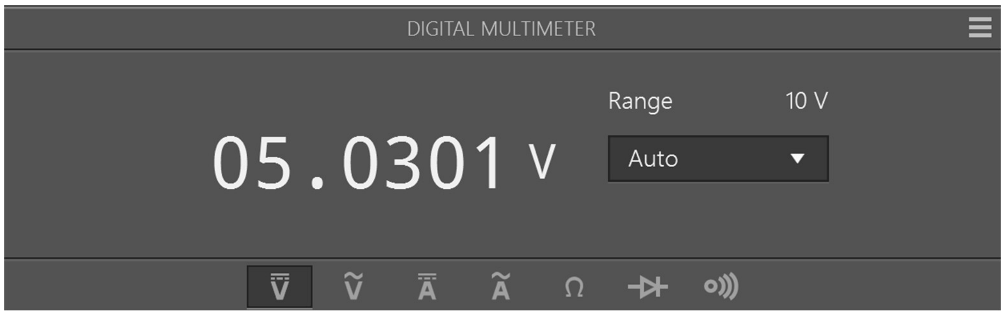

DIGITAL MULTIMETER

03.2742 V

Range 10 V

Auto

Figure 44: LaunchPad 3.3 V Verification

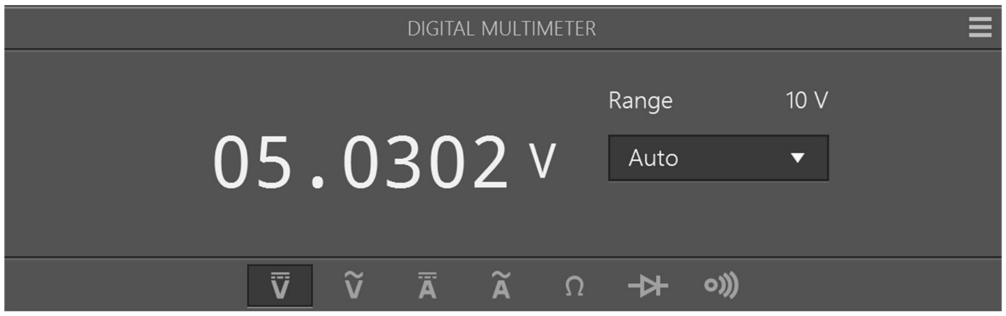

DIGITAL MULTIMETER

05.0301 V

Range 10 V

Auto

Figure 45: U1 Pin 7 Vcc Verification

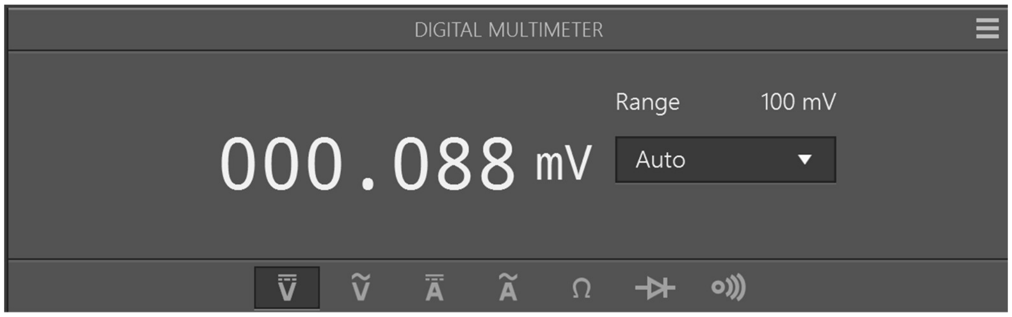

DIGITAL MULTIMETER

05.0302 V

Range 10 V

Auto

Figure 46: U7 Pin 7 Vcc Verification

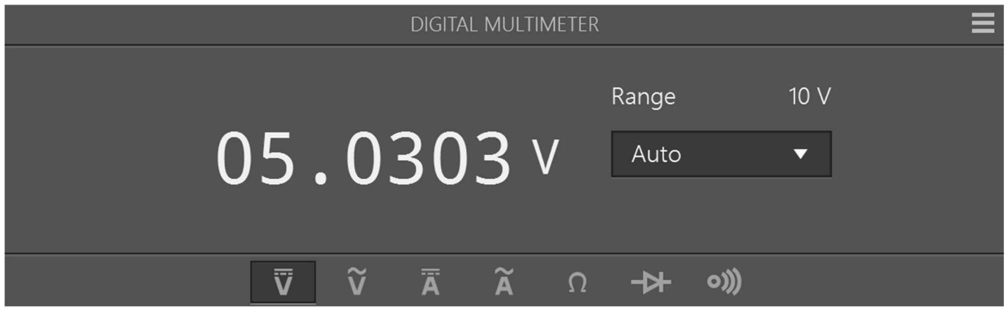

DIGITAL MULTIMETER

000.088 mV

Range 100 mV

Auto

Figure 47: U7 Pin 4 GND Verification

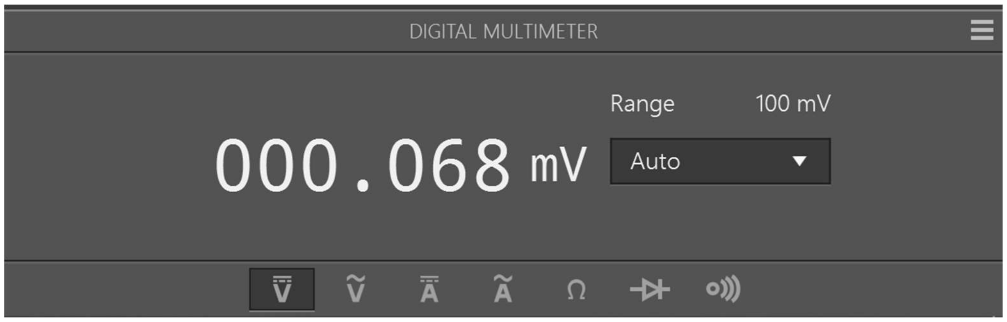

DIGITAL MULTIMETER

05.0303 V

Range 10 V

Auto

Figure 48: U2 Pin 8 Vcc Verification

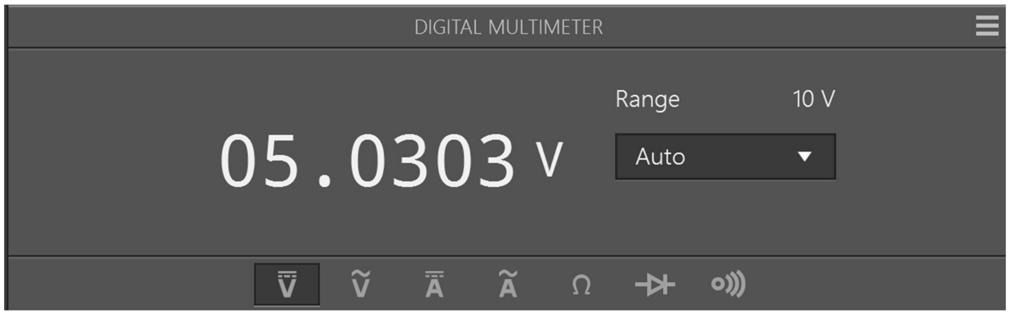

DIGITAL MULTIMETER

Range 100 mV

000.068 mV

Auto

Figure 49: U2 Pin 4 GND Verification

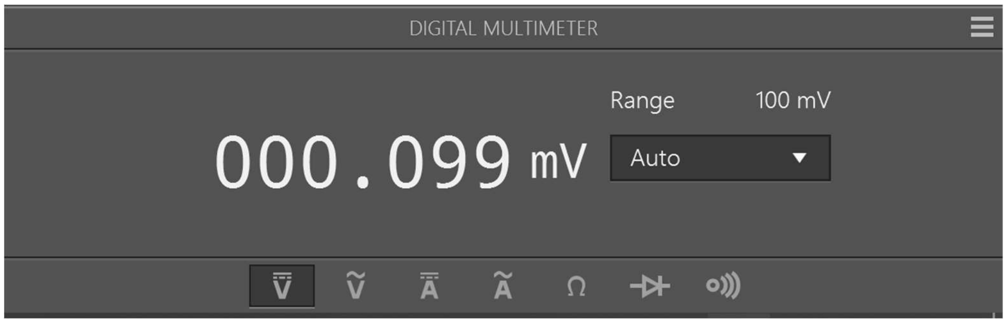

DIGITAL MULTIMETER

Range 10 V

05.0303 V

Auto

Figure 50: U3 Pin 8 Vcc Verification

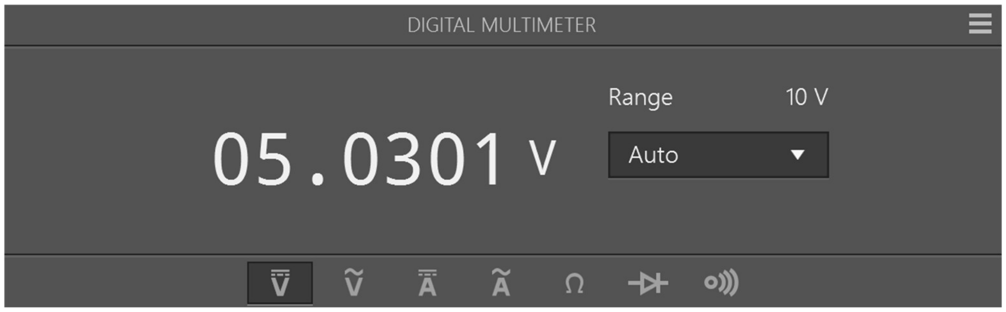

DIGITAL MULTIMETER

Range 100 mV

000.099 mV

Auto

Figure 51: U3 Pin 4 GND Verification

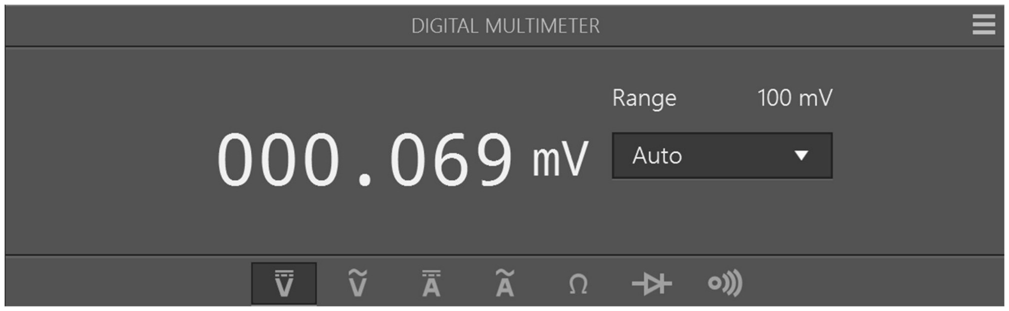

DIGITAL MULTIMETER

Range 10 V

05.0301 V

Auto

Figure 52: U6 Pin 8 Vcc Verification

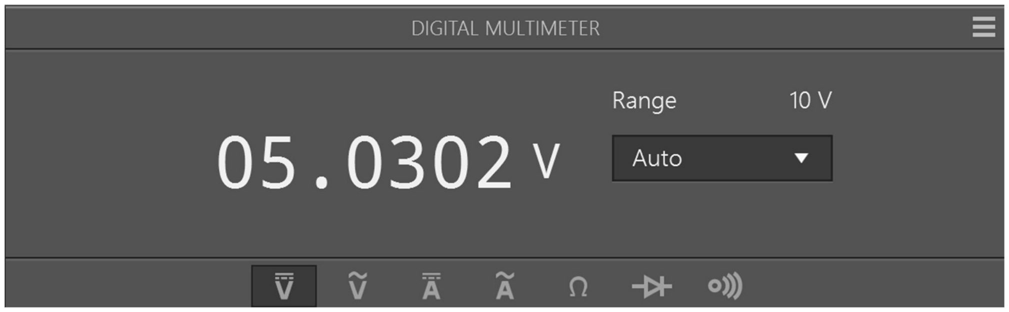

DIGITAL MULTIMETER

000.069 mV

Range 100 mV

Auto

Figure 53: U6 Pin 4 GND Verification

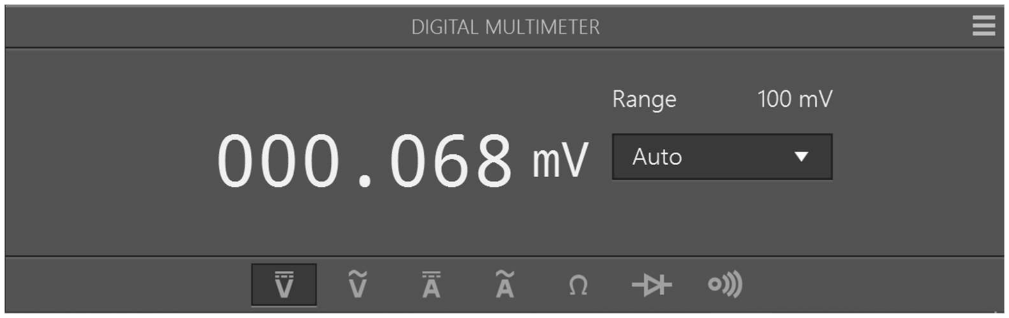

DIGITAL MULTIMETER

05.0302 V

Range 10 V

Auto

Figure 54: U10 Pin 8 Vcc Verification

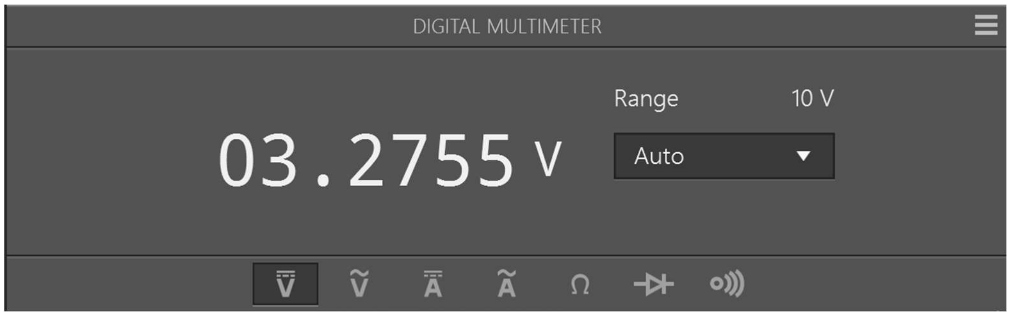

DIGITAL MULTIMETER

000.068 mV

Range 100 mV

Auto

Figure 55: U10 Pin 4 GND Verification

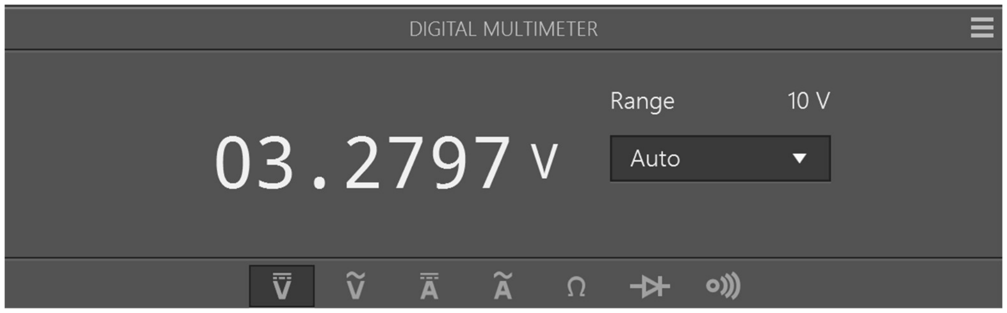

DIGITAL MULTIMETER

03.2755 V

Range 10 V

Auto

Figure 56: ESP32 Vcc Verification

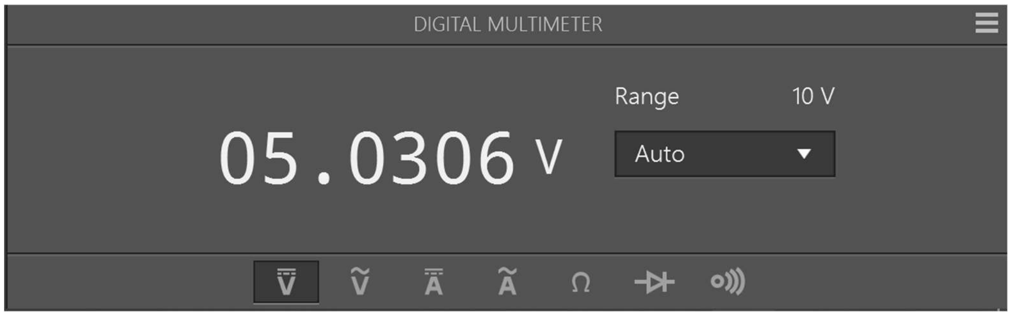

DIGITAL MULTIMETER

03.2797 V

Range 10 V

Auto

Figure 57: Camera Vcc Verification

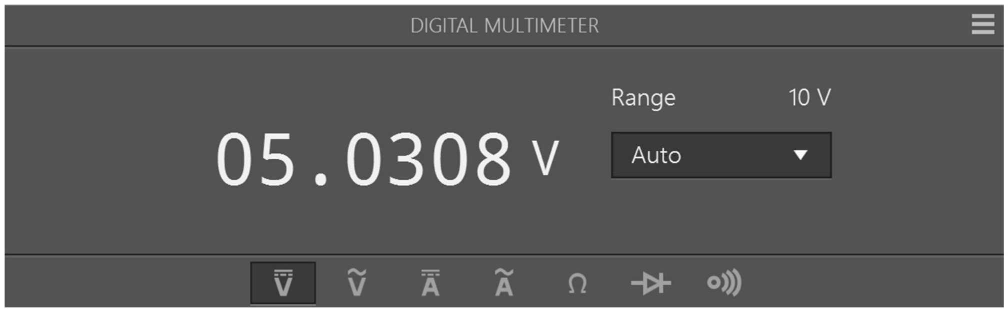

DIGITAL MULTIMETER

05.0306 V

Range 10 V

Auto

Figure 58: Cliff Sensor 1 Vcc Verification

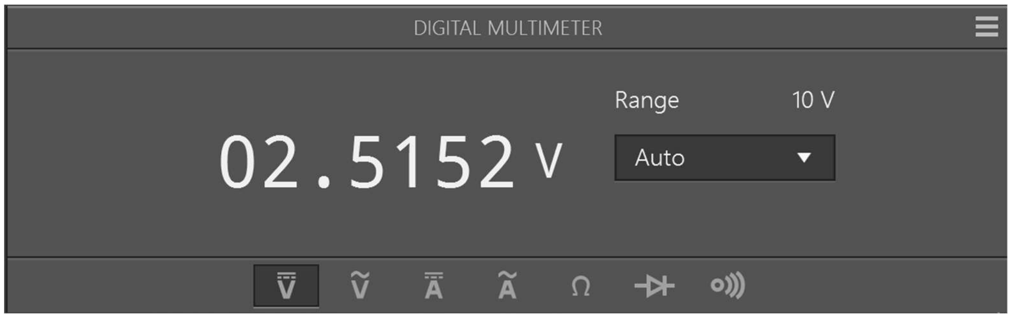

DIGITAL MULTIMETER

05.0308 V

Range 10 V

Auto

Figure 59: Cliff Sensor 2 Vcc Verification


DIGITAL MULTIMETER

02.5152 V

Range 10 V

Auto

Figure 60: Vmid Generator Verification

Appendix B
Robotany Firmware

Firmware Project

RobotanyControl.zip

Main Program

```
//-----------------------------------------------------------------------------------
// Includes
//-----------------------------------------------------------------------------------
#include <stdint.h>
#include "msp.h"
#include "Clock.h"
#include "../inc/PWM.h"
#include "../inc/LaunchPad.h"
#include "../inc/Motor.h"
#include "../inc/BumpInt.h"
#include "../inc/ADC14.h"
#include "../inc/TimerA1.h"
#include "../inc/IRDistance.h"
#include "../inc/SysTickInts.h"
#include "../inc/Tachometer.h"
#include "../inc/Reflectance.h"
#include "PortPins.h"

//-----------------------------------------------------------------------------------
// Macros
//-----------------------------------------------------------------------------------
#define TURN_SPEED              1500
#define DIFFERENCE_THRESHOLD    100
#define DIRECTION_THRESHOLD     50
#define NUM_TASKS               4
#define TIMERA1_PERIOD          50  // Units = 2 micro seconds
#define NUM_SAMPLES             256
#define DISTANCE_LIMIT          170
#define DRIVE_THRESHOLD         1000 // 1s
#define MAX_ADC                 16383
#define MAX_VOLTAGE             3.3
#define PLANT_THRESHOLD         4964 // ~(Threshold Voltage / Max Voltage)*(2^14 - 1)
                                     // Threshold voltage = 1V, Max voltage = 3.3V
#define QTR_EMITTERS_OFF        0
#define QTR_EMITTERS_ON         1
#define QTR_EMITTERS_ON_AND_OFF 2
#define QTR_MAX_SENSORS         16

//-----------------------------------------------------------------------------------
// Global Variables
//-----------------------------------------------------------------------------------
uint8_t gCollisionFlag, gADCflag; //mailbox
uint32_t gCounter; //For how long to drive forward
```

```
uint32_t gSignalArr[2]; //Hold previous signal and current signal
uint8_t  gLightingFlag = 0; //Flag for whether the plant signal is over threshold
volatile uint16_t gLS1Data;
volatile uint16_t gLS2Data;

//------------------------------------------------------------------------------------
// Read light data task
//------------------------------------------------------------------------------------
void ReadLightData(void){
    gLS1Data = VEML7700_Read_ALS1();
    gLS2Data = VEML7700_Read_ALS3();
}

//------------------------------------------------------------------------------------
// Check CH18 ADC Reading
//------------------------------------------------------------------------------------
void CheckCH18ADC(uint32_t raw18){
    if (raw18 >= PLANT_THRESHOLD) {
        gLightingFlag = 1;
    }
}

//------------------------------------------------------------------------------------
// Controller
//------------------------------------------------------------------------------------
void Controller(void){
    int32_t  LeftDist, RightDist;
    uint32_t raw14, raw16, raw18, raw19;

    P1->OUT ^= 0x01; // Profile
    P1->OUT ^= 0x01; // Profile

    LeftDist  = gLS1Data;
    RightDist = gLS2Data;
    ADC_In18_19_14_16(&raw18,&raw19,&raw14,&raw16);
    CheckCH18ADC(raw18);

    if (gLightingFlag) {
        if ((LeftDist - RightDist) >= DIFFERENCE_THRESHOLD) {
            Motor_Left(TURN_SPEED, TURN_SPEED); // Turn left
        }
        else if ((LeftDist - RightDist) <= -DIFFERENCE_THRESHOLD) {
            Motor_Right(TURN_SPEED, TURN_SPEED); // Turn right
        }
        else if (abs(LeftDist - RightDist) <= DIFFERENCE_THRESHOLD) {
           Motor_Stop();
           if (gSignalArr[0] == 0) { // First run
               gSignalArr[0] = (RightDist + LeftDist) / 2; // Average right and left
                                                           // signals
               gSignalArr[1] = (RightDist + LeftDist) / 2; // Average right and left
                                                           // signals
               Motor_Forward(TURN_SPEED, TURN_SPEED);
           }
           else if (gCounter < DRIVE_THRESHOLD) {
               Motor_Forward(TURN_SPEED, TURN_SPEED);
```

```
                gCounter += 1;
            }
            else{
                gSignalArr[0] = gSignalArr[1]; // Reassign current to previous
                gSignalArr[1] = (RightDist + LeftDist) / 2; // Average right and left
                                                             // signals
                if (abs(gSignalArr[1] - gSignalArr[0]) <= DIRECTION_THRESHOLD) {
                     // Optimal position
                    Motor_Stop();
                    gCounter = 0;
                    gLightingFlag = 0;
                }
                else if ((gSignalArr[1] - gSignalArr[0]) < -DIRECTION_THRESHOLD) {
                     // Backwards
                    Motor_Left(TURN_SPEED, TURN_SPEED);
                    Clock_Delay1ms(1800);
                    Motor_Stop();
                    gCounter = 0;
                }
            }
        }
    }

   else{
     Motor_Stop();
   }
    gADCflag = 1; // Semaphore
    P1->OUT ^= 0x01; // Profile
}

//------------------------------------------------------------------------------------
// Handler for encountering ledges
//------------------------------------------------------------------------------------
void HandleCliff(void){
    uint32_t Left, Right;
    int32_t  LeftDist, RightDist;
    uint32_t raw14,raw16,raw18,raw19;

    P1->OUT ^= 0x01;  // Profile
    P1->OUT ^= 0x01;  // Profile
    ADC_In18_19_14_16(&raw18,&raw19,&raw14,&raw16);

    Right  = LPF_Calc3(raw16); // Right  = channel 16, P9.1
    Left = LPF_Calc2(raw14);   // Left = channel 14, P6.1

    LeftDist    = LeftConvert(Left);
    RightDist   = RightConvert(Right);

    if(LeftDist >= DISTANCE_LIMIT || RightDist >= DISTANCE_LIMIT) {
        Motor_Backward(TURN_SPEED, TURN_SPEED);
        Clock_Delay1ms(20);
        gLightingFlag = 0;
    }

    gADCflag = 1;             // Semaphore
    P1->OUT ^= 0x01;         // Profile
}

//------------------------------------------------------------------------------
// Collision Handler
//------------------------------------------------------------------------------
void HandleCollision(void){
   uint8_t CollisionData;
   uint8_t BumpInputValue = 0;
   uint8_t BumpReturnValue = 0;

   BumpInputValue = ~(BUMP_PORT->IN);
   BumpReturnValue = ((BumpInputValue & (BUMP5_BIT|BUMP4_BIT|BUMP3_BIT))>>2);
   BumpReturnValue = BumpReturnValue |((BumpInputValue & (BUMP2_BIT|BUMP1_BIT))>>1);
   BumpReturnValue = BumpReturnValue |(BumpInputValue  &  BUMP0_BIT);

   CollisionData = BumpReturnValue;

   if((CollisionData != 0x00) && (CollisionData != 0x0C)){
      Motor_Backward(TURN_SPEED, TURN_SPEED);
      Clock_Delay1ms(10);
      Motor_Stop();
      gLightingFlag = 0;
   }
   gCollisionFlag = 1;
}

//------------------------------------------------------------------------------
// Task Type Structure
//------------------------------------------------------------------------------
typedef struct {
    void      (*Task)(void);      //Pointer to function task
    uint32_t  TaskCycleCounter;   //Task is executed when (TaskCycleCunter ==
TaskExecutionCycle)
    uint32_t  TaskExecutionCycle; //TaskExecutionCycle = Scheduler Frequency/Task
Frequency
} TaskType;                       //Scheduler Frequency = 1/TIMERA1_PERIOD

//------------------------------------------------------------------------------
// Tasks
//------------------------------------------------------------------------------
TaskType Tasks[NUM_TASKS] = {
         {&Controller,       0, 5   }, //execution frequency = 2kHz
         {&HandleCliff,      0, 5   }, //execution frequency = 2kHz
         {&HandleCollision,  0, 1   }, //execution frequency = 10kHz
         {&ReadLightData,    0, 1   }  //execution frequency = 10kHz
};

//------------------------------------------------------------------------------
// Task Scheduler
//------------------------------------------------------------------------------
void TaskSchedulerISR(void){
    unsigned int i;
    for (i=0; i<NUM_TASKS; i++) {
        Tasks[i].TaskCycleCounter++;
        if (Tasks[i].TaskCycleCounter == Tasks[i].TaskExecutionCycle){
            Tasks[i].TaskCycleCounter = 0;
            (*Tasks[i].Task)();
        }
    }
}

//-------------------------------------------------------------------------------------
// Initialize Bump Sensors
//-------------------------------------------------------------------------------------
void BumpInit(void){
    BUMP_PORT->SEL0 &= ~BUMP_BITS; // Configure port pins for bump sensors as GPIO.
    BUMP_PORT->SEL1 &= ~BUMP_BITS;

    SET_BUMP_SWITCHES_AS_INPUTS;
    ENABLE_PULL_RESISTORS;
    PULL_UP_RESISTORS;
}

//-------------------------------------------------------------------------------------
// Main
//-------------------------------------------------------------------------------------
void main(void){
  uint32_t raw14, raw16, raw18, raw19;

  uint32_t size = NUM_SAMPLES;
  gADCflag = 0;
  gCollisionFlag = 0;

  DisableInterrupts();
  Clock_Init48MHz();

  Tachometer_Init();
  Motor_Init();
  BumpInit();
  LaunchPad_Init();

  ADC0_InitSWTriggerCh17_14_16(); // Initialize channels 17,14,16
  ADC_In18_19_14_16(&raw18,&raw19,&raw14,&raw16);
  LPF_Init2(raw14, size);   // P6.1/channel 14
  LPF_Init3(raw16, size);   // P9.1/channel 16

  VEML7700_Init();
  TimerA1_Init(&TaskSchedulerISR, TIMERA1_PERIOD); // TimerA1 tasks
  EnableInterrupts();     // 10000 Hz sampling

  while(1){

  }
}
```

Appendix C

Computer Vision and Growth Prediction Package

ComputerVision.zip

Server Package

PhotoSynthesis-Server.zip

ESP32 Package

Robotany-ESP-IDF.zip

Mobile Application Package

RobotanyReactApp.zip

Appendix D

Bill of Materials

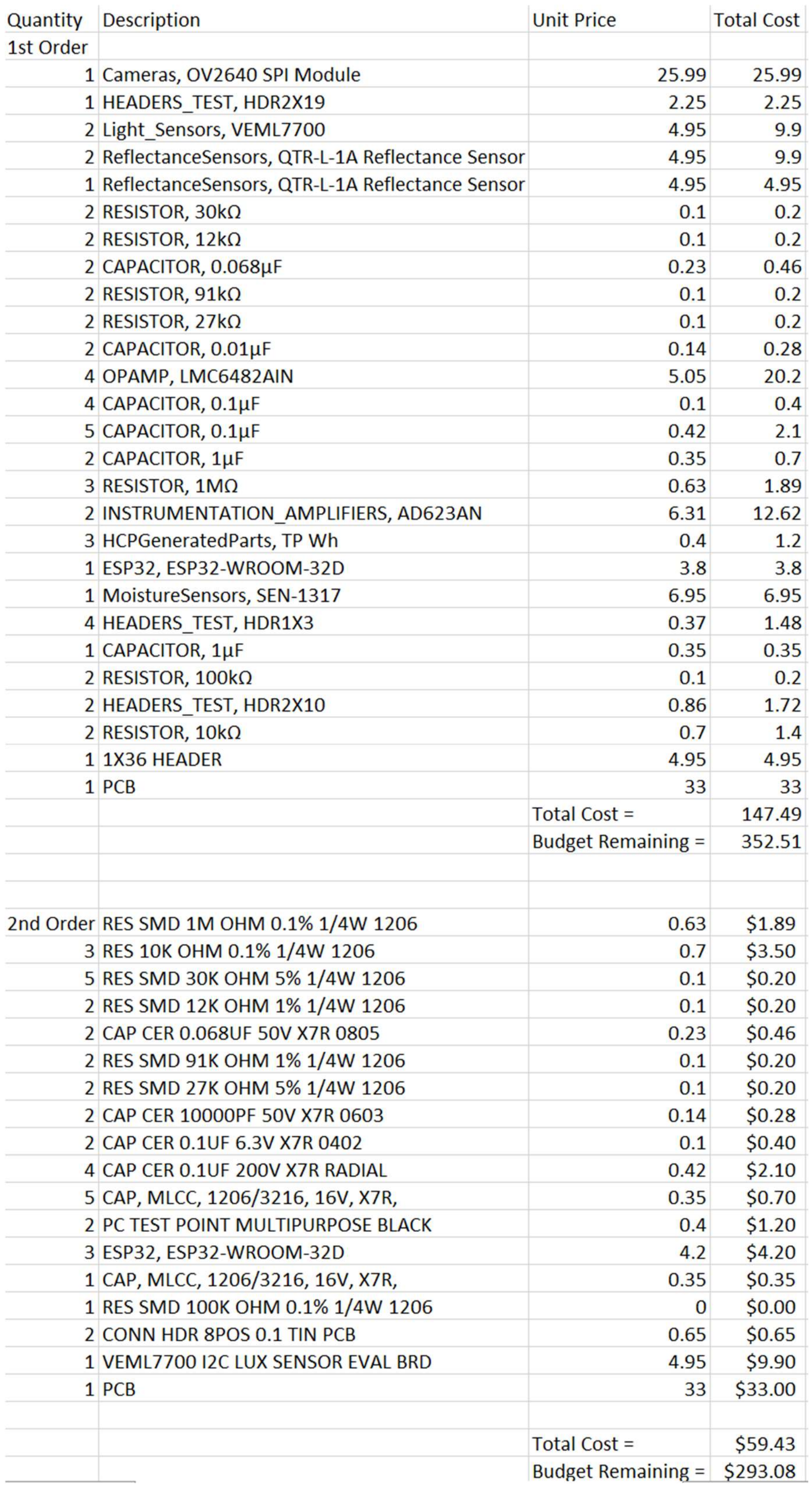
Table 4: Digi-Key Bill of Materials

Manufacturer Part Number	Manufacturer	Digi-Key Part Number	Customer Reference	Reference Designator	Packaging	Part Status	Quantity	Unit Price	Extended F	Quantity Available
4162	Adafruit Industries LLC	1528-2891-ND	Our Fourth Rodeo	LS1, LS2	Bulk	Active	2	4.95	$9.90	186
RT1206BRD071ML	Yageo	YAG2022CT-ND	Our Fourth Rodeo	R1, R8, R10	Cut Tape (CT)	Active	3	0.63	$1.89	67635
RN73H2BTTD1002B25	KOA Speer Electronics, Inc.	2019-RN73H2BTTD1002B25CT-ND	Our Fourth Rodeo	R9, R15, R16, R17, R18	Cut Tape (CT)	Active	5	0.7	$3.50	3686
RC1206JR-0730KL	Yageo	311-30KERCT-ND	Our Fourth Rodeo	R3, R11	Cut Tape (CT)	Active	2	0.1	$0.20	313102
RC1206FR-0712KL	Yageo	311-12.0KFRCT-ND	Our Fourth Rodeo	R4, R12	Cut Tape (CT)	Active	2	0.1	$0.20	299603
VJ0805Y683KXACW1BC	Vishay Vitramon	720-1842-1-ND	Our Fourth Rodeo	C4, C13	Cut Tape (CT)	Active	2	0.23	$0.46	5064
RC1206FR-0791KL	Yageo	311-91.0KFRCT-ND	Our Fourth Rodeo	R6, R13	Cut Tape (CT)	Active	2	0.1	$0.20	105771
RC1206JR-0727KL	Yageo	311-27KERCT-ND	Our Fourth Rodeo	R7, R14	Cut Tape (CT)	Active	2	0.1	$0.20	360582
UMK107B7103KAHT	Taiyo Yuden	587-5957-1-ND	Our Fourth Rodeo	C6, C15	Cut Tape (CT)	Active	2	0.14	$0.28	5364
0402B104K6R3CT	Walsin Technology Corporation	1292-1177-1-ND	Our Fourth Rodeo	C3, C5, C14, C16	Cut Tape (CT)	Active	4	0.1	$0.40	2041830
C330C104K2R5TA7301	KEMET	399-14009-1-ND	Our Fourth Rodeo	C_BP1, C_BP2, C_BP3, C_BP4, C_BP8	Cut Tape (CT)	Active	5	0.42	$2.10	20286
EMF316B7105KLHT	Taiyo Yuden	587-6519-1-ND	Our Fourth Rodeo	C1, C11	Cut Tape (CT)	Active	2	0.35	$0.70	5807
5011	Keystone Electronics	36-5011-ND	Our Fourth Rodeo	U8, U13, U21	Bulk	Active	3	0.4	$1.20	465114
ESP32-WROOM-32D (8MB)	Espressif Systems	1904-1024-1-ND	Our Fourth Rodeo	U4	Cut Tape (CT)	Active	1	4.2	$4.20	2437
EMF316B7105KLHT	Taiyo Yuden	587-6519-1-ND	Our Fourth Rodeo	C2	Cut Tape (CT)	Active	1	0.35	$0.35	5807
RT1206BRB07100KL	Yageo	YAG5016TR-ND	Our Fourth Rodeo	R2, R5	Tape & Reel (TR)	Active	2	0	$0.00	5000
PPTC081LFBN-RC	Sullins Connector Solutions	S7006-ND	Our Fourth Rodeo	Camera	Tray	Active	1	0.65	$0.65	18727

Table 5: Additional Parts Bill of Materials

Quantity	Description	Cost/Unit	Total Cost
1	Cameras, OV2640 SPI Module	25.99	25.99
1	Moisture Sensors, SEN-1317	6.95	6.95
2	HEADERS_TEST, HDR2X10	0.86	1.72

Table 6: Total Order Costs and Budget Remaining



Quantity	Description	Unit Price	Total Cost
1st Order			
1	Cameras, OV2640 SPI Module	25.99	25.99
1	HEADERS_TEST, HDR2X19	2.25	2.25
2	Light_Sensors, VEML7700	4.95	9.9
2	ReflectanceSensors, QTR-L-1A Reflectance Sensor	4.95	9.9
1	ReflectanceSensors, QTR-L-1A Reflectance Sensor	4.95	4.95
2	RESISTOR, 30kΩ	0.1	0.2
2	RESISTOR, 12kΩ	0.1	0.2
2	CAPACITOR, 0.068µF	0.23	0.46
2	RESISTOR, 91kΩ	0.1	0.2
2	RESISTOR, 27kΩ	0.1	0.2
2	CAPACITOR, 0.01µF	0.14	0.28
4	OPAMP, LMC6482AIN	5.05	20.2
4	CAPACITOR, 0.1µF	0.1	0.4
5	CAPACITOR, 0.1µF	0.42	2.1
2	CAPACITOR, 1µF	0.35	0.7
3	RESISTOR, 1MΩ	0.63	1.89
2	INSTRUMENTATION_AMPLIFIERS, AD623AN	6.31	12.62
3	HCPGeneratedParts, TP Wh	0.4	1.2
1	ESP32, ESP32-WROOM-32D	3.8	3.8
1	MoistureSensors, SEN-1317	6.95	6.95
4	HEADERS_TEST, HDR1X3	0.37	1.48
1	CAPACITOR, 1µF	0.35	0.35
2	RESISTOR, 100kΩ	0.1	0.2
2	HEADERS_TEST, HDR2X10	0.86	1.72
2	RESISTOR, 10kΩ	0.7	1.4
1	1X36 HEADER	4.95	4.95
1	PCB	33	33
		Total Cost =	147.49
		Budget Remaining =	352.51
2nd Order	RES SMD 1M OHM 0.1% 1/4W 1206	0.63	$1.89
3	RES 10K OHM 0.1% 1/4W 1206	0.7	$3.50
5	RES SMD 30K OHM 5% 1/4W 1206	0.1	$0.20
2	RES SMD 12K OHM 1% 1/4W 1206	0.1	$0.20
2	CAP CER 0.068UF 50V X7R 0805	0.23	$0.46
2	RES SMD 91K OHM 1% 1/4W 1206	0.1	$0.20
2	RES SMD 27K OHM 5% 1/4W 1206	0.1	$0.20
2	CAP CER 10000PF 50V X7R 0603	0.14	$0.28
2	CAP CER 0.1UF 6.3V X7R 0402	0.1	$0.40
4	CAP CER 0.1UF 200V X7R RADIAL	0.42	$2.10
5	CAP, MLCC, 1206/3216, 16V, X7R,	0.35	$0.70
2	PC TEST POINT MULTIPURPOSE BLACK	0.4	$1.20
3	ESP32, ESP32-WROOM-32D	4.2	$4.20
1	CAP, MLCC, 1206/3216, 16V, X7R,	0.35	$0.35
1	RES SMD 100K OHM 0.1% 1/4W 1206	0	$0.00
2	CONN HDR 8POS 0.1 TIN PCB	0.65	$0.65
1	VEML7700 I2C LUX SENSOR EVAL BRD	4.95	$9.90
1	PCB	33	$33.00
		Total Cost =	$59.43
		Budget Remaining =	$293.08