Automated Robotic Gardener

Wallace Borges, Seana Falvey, Denver Lau, and Rafael Smith

Dept. of Electrical and Computer Engineering, University of Central Florida, Orlando, Florida, 32816-2450

***Abstract* — The paper describes a lower budget application of CNC design, internet applications, and machine learning for an automated robotic gardener for gardening enthusiasts/hobbyists. The mechanical design and electrical components are implemented for live streaming, drip irrigation, and motor movement. A web application provides collected sensor data, manual control, and chronological data tracking. A weed detection system is described and implemented. Finally, the results and effectiveness of these methods are discussed.**

***Index Terms* — Electric motors, microcontrollers, computer vision, machine learning, gardening.**

I. Introduction

The Automated Robotic Gardener, or Auto-Gardener for short, aims to reduce time used in gardening tasks, reduce water waste, lower pesticide use, and proliferate weed in gardens. It also helps eliminate problems associated with transportation and pollution. Auto-Gardener will perform regular maintenance of the garden which includes navigating around the garden area and performing gardening tasks. These include watering, detecting weeds, and monitoring plant vitals. It also provides an interface for users to monitor their garden remotely. Auto-Gardener uses a microcontroller to fully automate multiple gardening tasks. Once these tasks are finished, it periodically sends activity information to the server. It uses a drip irrigation system and weather information to optimize the watering schedule. The system traverses the entire garden, taking photos and sending it to an online server to process for weed detection.

The project is geared towards both gardening novices and enthusiasts- anyone that is interested in taking care of their personal garden without needing to routinely manage it. Hobby projects that currently exist are very simple and do not have the technology that the Auto-Gardener has such as machine learning and Web App. A possible application could be to use for small businesses or amateur cooks who need to grow small scale produce like herbs. It could even be used to track any animals traveling through the area from the live stream.

The main appealing aspect is to be in control of what you consume. Pesticide information associated with each produce is not readily available to the public. Even chemicals that prolong shelf life are added to fresh produce. Being able to harvest your food at a later time than high-capacity farms allow the food to gain more vitamins. Lastly, not buying food stored in plastic limits the accidental microplastics consumed. The Automated Robotic Gardener, through its ease of use and automated tasks, is a step towards solving these multiple issues in the future.

II. Overview of the System Components

The following section is an overview of the system components chosen for the robotic gardener and how the components are interfaced.

A. Microcontroller

The microprocessor used is the ATmega328P. It has 23 GPIO pins for the input/output purposes [1]. The microprocessor has an internal RC oscillator running at 8.0 MHz, but an external 16 MHz oscillator is connected for more accurate clock functions. An Arduino board and Arduino IDE is used to program and debug the microcontroller.

B. Raspberry Pi & Camera

The Raspberry Pi is near independent of the microcontroller and is used to control the camera and TCP/IP communications. This is a modification from the original design which had the ESP-32 to control the camera and handle WiFi communication. Issues arose with the ESP-32 as performance and power limitations were consistently impeding progress. The ESP-32 was also more difficult to program and debug, due to the lack of a development board and lack of officially supported libraries. The ESP-32's inability to livestream video at the required resolution and handle TCP/IP communications is what ultimately guaranteed the switch of hardware. The Raspberry Pi's superior specifications: A quad-core 1.5GHz CPU, 4GB RAM and upto 64GB SD card memory [2], means that it can easily handle both live streaming video and TCP/IP communications.

C. Motors and Motor Drivers

NEMA17 Motors and BIQU A4988 Motor Drivers controlled the gantry movement. For precise positioning, stepper motors are the best choice. As it corresponds to its name, stepper motors move in steps. They have a controller framework that assigns the situation through signal pulses sent to a driver, which deciphers them and sends corresponding voltage to the motor. Stepper motors

are moderately easy to control; however, they draw maximum current constantly. In full step resolution mode it took 200 microsteps for a NEMA17 motor to complete one full rotation. These chips also have the benefit of keeping the motors taught, preventing them from rolling backwards or slipping on the gantry track. While running the NEMA17 stepper motor consumes approximately 1.2 amps, and provides 3.17Kg*cm of torque.

D. Soil Moisture Sensor

The sensor used for this project is a capacitive I2C sensor that also provides ambient light and temperature readings built by Catnip Electronics [3]. Through testing the soil moisture sensor reads values of: 200 for dry air, and 300 for pure water. This gives a range of 100 integer values as an indicator. The soil moisture sensor was chosen for its capacitive readings, its ruggedness, its insensitivity, and stability. Catnip Electronics sensor uses a I2C protocol, operating at 3.3V to 5V, and consuming very little current, 0.7 mA.

E. Ultrasonic Sensor

Ultrasonic Ranging Module HC-SR04, provided distance measurements which were used to track the amount of water in the water tank. The sensor is accurate from a distance of 2cm to 400cm. Like many of the components, the HC-SR04 needs a power supply of 5V DC. Its trigger input pulse width is 10µs and has a resolution of 0.3cm, a qualified choice for the design.

F. Watering

The Mountain Ark Mini Pump was a late addition to the project, superseding the original water solenoid design which provided only a trickle of water. This water pump fits perfectly as a replacement since it operates off the same voltages, 10.5 to 13.5V DC, as the solenoid. The only difference is that the pump consumes power at 4.8 Watts.

G. Power

The power system is across the board. There are two sources of power, for the Raspberry Pi, a 5V, 3A AC/DC power converter and for the ATmega and its components, a 12V, 2A AC/DC converter. This method was chosen for ease to focus on more important aspects and goals of the project. The motors and water pump were the only electronics that needed a 12V supply. Hence, a power converter from 12V to 5 V was needed for the rest. The LM2596 was the DC/DC step down converter that fulfilled the requirements.

III. Physical Design

The team built the ACRO510 System by Open Builds. The ACRO System provides a modular system for an accurate positioning system. It was selected for cost effectiveness, openly documented design, and flexibility to place different technologies. The versatility allowed the team to add different attachments.

The dimensions of the original ACRO510 System design is 40 inches long by 20 inches wide. Although, the actual work area of the motor-camera combo is only 12 inches by 30 inches, according to Open Builds . The drive system is by GT2 timing belts. Two belts are designed on either side of the x-axis movement, and one is designed along the y-axis.

With the team's addition of the Aluminum V-Slot Linear Rails dug into the ground, the frame sits at approximately 18 inches high. Sufficient height is key to taking exceptional pictures for the AI to make accurate decisions. The Raspberry Pi and its accompanying camera were rigged facing down, onto the acrylic plates of the y-axis. The 10 feet long water pipe, coming from a 5 gallon water bucket, was attached to the same acrylic plates. Care was taken to point the water flow away from electronics. With this, the motors can be moved directly above the plants to water them individually. The water pipe arrangement was an epiphany by the team late into the build process. A potential future concept would be to program the system to water based on each plants' personal needs.

IV. System Concept

The systems are cyclic in nature, alternating between autonomous functions and user commands. User commands receiving priority over returning to autonomous functions.

A. System Software Concept

The project is designed to be autonomous with the ability to accept user input. The autonomous functions are carried out with the locally installed electronics. These functions include: watering, soil moisture recording and image recording. Weed detection and weather reporting are offloaded to the user's personal computer on the desktop application.

B. System Hardware Concept

The three main processors are the ATmega328, which handles the majority of the functions, the Raspberry Pi 4 Model B, and a computer hosting as a server. Fig. 1 shows the block diagram representation of the hardware interface in the overall design. The arrows indicate the flow model of the inputs and outputs. The Raspberry Pi and ATmega

transmit data by serial communication. The Raspberry Pi and computer, which hosts the server and Web App, communicate via the Transmission Control Protocol. The ATmega328 within the PCB is the primary hub, controlling 10 of the components. The Raspberry Pi had a Wi-Fi chip and SD card that was utilized.

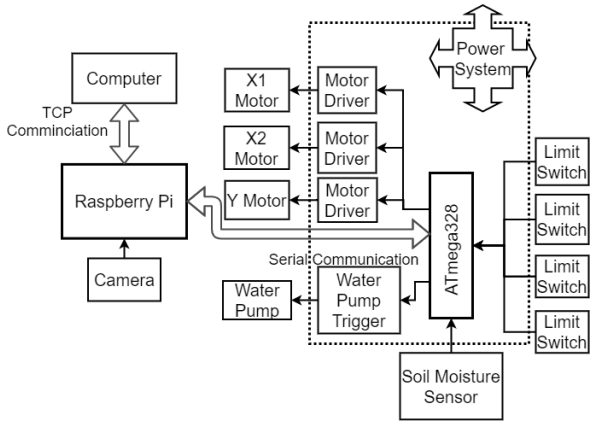


Fig. 1. Block diagram of the major system components. I/O flow model is shown.

V. Printed Circuit Board

With the exception of the Raspberry Pi and its camera, the printed circuit board for this project contains all electronics needed for this project. The major components contained on the PCB are the 12V to 5V power converter, motor drivers, ATmega328P microcontroller, LEDs to signify power and operation, and transistor for pump operation. While not every pin on the ATmega was used, every pin is designed to be accessible on the PCB. Consequently, extra soldering pads were added to the design.

Most of the design is uncomplicated and does not need to be discussed. The following are standard practice. There is a 16MHz external clock crystal oscillator (XCO) with a capacitor at each terminal. This is a cornerstone to programming the microcontroller for clock dependent electronics or programs. Two pull-up resistors are included for the soil moisture sensor since it utilizes the I2C serial communication protocol. The driver has an open drain, so it needs a means of pulling the signal from low to high. Each of the SDA and SCL signal lines has a 4.7kΩ resistor. The circuit of a TIP120 transistor, a 1N0001 diode, and 2.2k resistor shown in the top middle area of Fig. 2 controls the water pump. The transistor receives signals LOW or HIGH from the pin which the base is connected to. In this design shown in Fig. 2, the emitter had to be physically soldered to ground, as there was a small, yet significant error. 2 LEDs indicate power and starting program of the microcontroller. A future design would be to add additional LEDs for indicating power for the water pump, motors, and ultrasonic sensors.

The PCB has two layers, is 109 by 89 mm, and has a 1.6 mm thickness. The surface finish is HASL with lead, and the copper weight is 1 oz.

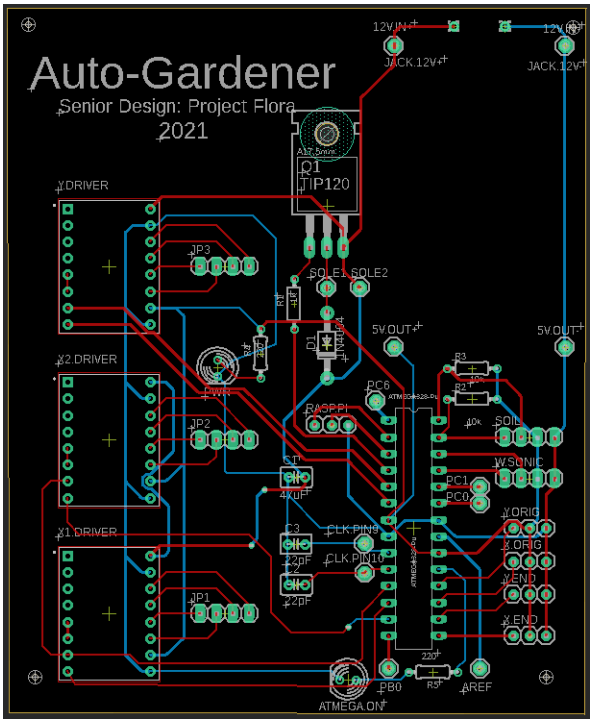


Fig. 2. Printed Circuit Board Design

Lastly, the PCB was implemented to have multiple male pin headers. This allows it to not commit to the PCB design while permitting for complex circuits.

VI. Software Detail

The software can be split into three parts, each dealing with a major processing component: the ATmega, the Raspberry Pi, and the desktop application. The ATmega portion handles: motor control, soil moisture, water level and communicates with the Raspberry Pi. The Raspberry Pi in turn handles the camera live stream and image capture as well as communication with the ATmega and the user. The user's desktop application processes images for weed detection, pulls weather information from a third party service and communicates with the Raspberry Pi. The software detail of the desktop application is discussed in a later section.

A. Microcontroller Program

The ATmega328p's program first checks for serial communication. The ATmega328p software logic flow is

shown in Fig. 3. This serial communication can come from the Raspberry Pi, ultimately from the user by the Web App, or from the check moisture function. If there are no new commands, the motors will automatically turn on for a Patrol Route.

The Patrol route has 10 coordinates that it moves to. The success of the project was dependent on the stepper motors moving correctly to the coordinates. In order to speed up traversal times of the gantry, the software will send a signal to move the gantry 90% of the distance needed, with the final 10% distance covered using small measured steps checking the limit switches to ensure the gantry does not run out of track.

Next the moisture is checked and the loop begins again. On the other hand, if the patrol route is paused by a user command, it will translate the command, execute, and send the current status. It waits for any additional commands before it chooses to continue onto the patrol route or pause the patrol route. The software restarts its loop. Both automation and optional manual control are achieved through this plan.

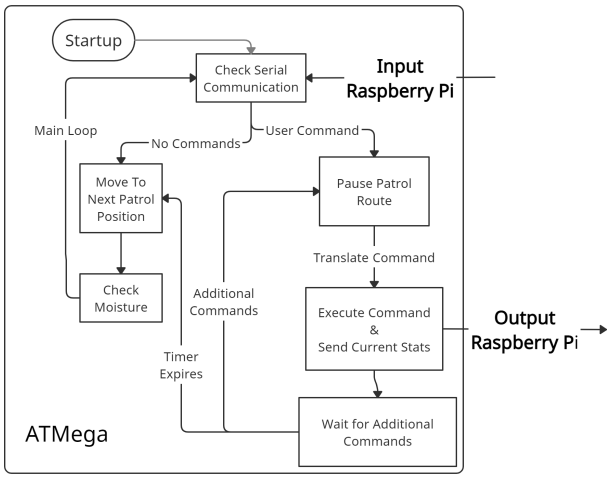


Fig. 3. ATmega Software System

B. Raspberry Pi Program

The Raspberry Pi considers the data from the ATmega and the Desktop App through multi-threading. Fig. 4. was created as a visual aid. It shows a flowchart of how the overall program works. Through the serial communication thread, it reads any input from the ATmega and checks for any commands in the queue. If it is busy, it will pause. Else, the command is sent out to the ATmega where it will perform the command. Automatically, the camera server will turn on at startup. The video is fed live through the server onto the App. For the TCP/IP thread, it will check for any new user commands from the App and will add it to the command queue. Information is updated and the loop restarts to check for a new user command.

As shown in both flowcharts, neither of the programs intends to halt to a complete stop. They are intended to keep checking for new information.

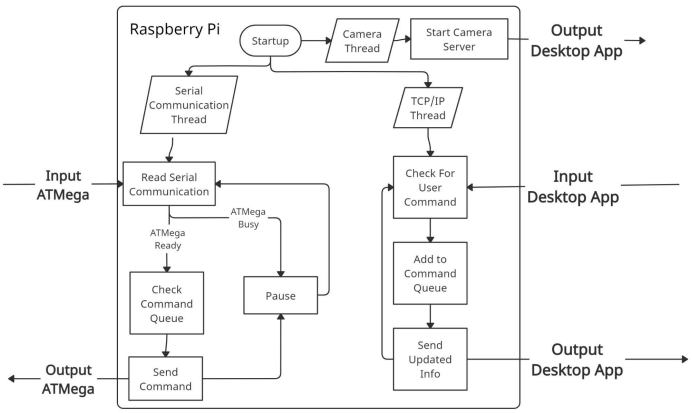


Fig. 4. Raspberry Pi Software System

C. Communications

All three of the major software components: the ATmega, the Raspberry Pi and the desktop application communicate information using the same format for ease of programming. For this project both the serial communication and TCP/IP use formatted strings for any data sent back or forth. The slowdown caused by using resource intensive functions like string compare or string tokenize was considered minimal and acceptable for having human readable messages. Through testing, the desktop application was able to communicate through the Raspberry Pi to the ATmega with an average time of 1.74 seconds, well beneath Auto-Gardener's engineering specification.

VII. Computer Vision

The goal for the computer vision was to classify and locate a majority of the weeds in the garden. This was achieved by training an object detector on three classes, Garden Sage (Salvia officinalis), Red Lettuce (Lactuca sativa), and random weeds in the surrounding area. Certain species of plants are selected so that the detector can be trained on a variety of weed species while reducing the chances that it detects desired plants in the garden as weeds.

In the selection of object detector systems, accuracy and computation cost were the most relevant factors. Between anchor based one-stage detectors, RetinaNet, YOLO, and SSD were considered because of comparable computation speeds, open source architecture, and availability of pretrained weights [5]. YOLOv4 has a higher accuracy compared to these other real time detectors trained on the

same dataset, MS COCO-2017 with a 62.8% mean average precision on the architecture the team implemented [5].

YOLOv4(You Only Look Once) is composed of the major components, the backbone, neck, and head [5]. The image is first fed into the backbone which consists of a convolutional neural network (CNN) CSPDarknet53, a neck with SPP and PAN, and head YOLOv3 [5]. An input image is fed to the backbone, which then extracts multiple features of the image in feature maps as it passes through the network. The neck connects and mixes feature maps from backbone output and passes them to the head for prediction [6]. The head draws the bounding boxes and labels as the final output, displaying the percent confidence and classification for each.

For the dataset selection, a synthetic dataset was created due to lack of availability of large annotated weed datasets for the detector to train on. 40 images of empty soil and ground in the surrounding areas were taken and resized to 2016x1512 resolution from phone cameras at 16 inches in height. 10 pictures of garden sage, 10 pictures of red lettuce, and 99 images of weeds were taken using the same methods but the resolution was maintained below 700 pixels. Each of the plant images were manually cropped out using GIMP in order to add transparency for image composition. Using python software, a random portion for the background was cropped out and a combination of 1-3 weeds, 0-2 of garden sage/red lettuce was composed onto the image. Then additional image augmentation was performed randomly including rotation, flipping, brightness, and rescaling to the image to increase variability of the dataset. For each plant foreground added, a RGB mask was generated onto another image. Using opencv, each different colored RGB mask was found by detecting the largest contour of that color. Then it drew a bounding box around it, recording the classification, width, height, center x, center y and coordinates. These were written to an annotation file for each image. 3000 synthetic images were generated at 992x992px for training.

The detector was trained on a free GPU- Tesla T4 for 6000 epochs. For each epoch, the entire training dataset of 2400 images were resized to 416x416 to reduce computation time and ran through the entire network. This took approximately 10 hours for each version. Seven different versions of the detector were tested. The initial versions were used to test effectiveness at detecting weeds and improving accuracy. Further editions contained more cropped images of weeds. Adjustments were made such as increasing the number of weeds per image; variations in generated image size; applying green color segmentation to reduce noise; adding smaller weeds; expanding brightness ranges, increasing scaling ranges; and finally, adding more classes to detect in order to distinguish between desired plants and weeds. Due to computation constraints, the effects of each of these were not clear immediately. Adjustments such as increasing weeds to five per image and detecting additional classes decreased the overall accuracy of the detector. A constant indicator of the dataset not being diverse enough was overfitting. The detector accuracy would not increase with more training time either due to memorizing features of the dataset.

TABLE I
Detector Training Average Precision

Class	Average Precision	True Positive	False Positive
Garden Sage	95.27 %	138	12
Red Lettuce	96.41 %	127	6
Weed	93.61%	557	42

In the synthetic testing dataset containing 300 images, the best detector reached a 95.10 % mean average precision across the 3 classes as shown in Table I. The detector had 95.27%, 96.41%, and 93.61% average precision for Garden Sage, Red Lettuce, and weeds respectively. These statistics were calculated assuming a weed is correctly detected on 50% IoU, meaning the bounding box the detector predicted overlaps with at least 50% the detected plant's ground truth bounding box.

On real images, the detector was able to classify unseen weeds in it's dataset assuming similar height and weather conditions to the dataset. Its accuracy decreased significantly detecting plants in the garden bed. A video was taken of the entire gardening area to give the detector multiple angles of each plant. Out of 14 plants in the area, the detector was able to correctly classify 9 at a certain angle or 64.29% of the garden plants. The detector still had trouble identifying small weeds and confused garden sage plants for weeds at higher heights. This challenge could be overcome by having the object detection model customly configured to detect smaller objects or train it on a separate class of small weed images.

Future improvements to increase detection rate and applicability of the plant detector would be to add plants in rain, fog, snow or other weather conditions. A specific weed could have been identified instead of many different

species. The number of different types of plants could have been increased in the garden at the cost of more training time and data collection. Another possible feature would be using a camera with zoom on possible smaller weeds or moving on the z-axis to get an optimal height range for detection.

VIII. Web App Architecture

The motivation for the web app was to create an interface where the user has virtual access to their garden, real-time representation of how the garden was doing, and on-screen controls for the device. Since the app would be the direct link of the user to the device, the design goals should be centered around the user's experience and what they would want as features. These features include: a user-friendly and seamless design, displaying plant vitals and sensor data in a concise way, on-screen motor and water solenoid controls, livestream and live capture capabilities, an interface for processing AI predictions on captured images, tracking of activities with log, and an option for direct communication with server. With these features available to the user, the app would serve as a virtual portal to their garden while also delivering the user peace-of-mind knowing that, while the garden is being taken care of autonomously, they still have access to controls with on-demand commands if they wanted to interact with their garden.

To build the app, Kivy was chosen as the framework. Kivy is a Python-based framework, which gives it the ability to use Python's various extensive and powerful libraries. Kivy also provides cross-platform functionality by delegating graphics rendering to the cross-language, platform-independent OpenGL API. This API is great for displaying and handling multimedia, which was an important feature since live streaming needs to be implemented, as well as an interface where the user could access the images captured. Another useful feature of Kivy is their use of kv language. Through kv language, the programmer is able to create a widget tree in a declarative way, while also binding widget properties to each other or to callbacks in a natural manner [7]. This provides kv language with fast prototyping and complex user interface (UI) build capabilities. Kivy also separates design logic and design layout by using both .py and .kv files, respectively. With the .py file handling the logic behind the algorithms, the .kv file handles all the layout design.

The root layout of the app is a TabbedPanel, which is a Kivy class that manages the screen by implementing tabs, which can access different pages when clicking on them. The tabs each have a different purpose. The first tab is the Main Menu, providing a real-time "Garden Snapshot" by showing current weather data, such as temperature and humidity, as well as measurement readings for the soil moisture and water tank levels. These can be refreshed by pressing the "Refresh Vitals" button, which fetches information from both an online weather API and from the device's own sensors. There is also a portion at the bottom that gives friendly tips on gardening and how to better use the device.

The next tab is the Camera tab, which focuses on live streaming of the camera and motor movement. Since the camera is mounted on the mobile gantry unit that is moved by motors, controlling the camera also means being able to control the motors. Thus, motor controls are provided in the left-hand side of the Camera tab. The controls allow for three methods of requesting motor movement: 1-dimensional arrow keys, buttons to jump to hard-coded quadrants, and text boxes to enter exact coordinates. The rest of the screen is filled with live camera feed. The feed is accessed through the internet and rendered on the screen. The live feed is helpful when moving the camera around since the requested movements are reflected in the camera's output. Under the live feed window is a "Capture Picture" button that saves the current frame shown at that instant and automatically starts processing the image using the AI weed detection functionality. With the current processing power, the detector can run computation on any of the 1440x1080p images from the live stream in under 10 seconds.

The next tab is naturally the Weeds tab, which provides an interface for calling and reviewing the predictions of the weed detection. The most recently processed image, called a "prediction", is displayed with the bounding boxes and labels drawn by the AI algorithm. On the right of the screen are stats derived from the image processing, showing the number of weeds, garden sage, and red lettuce found in the most recent prediction. Under the stats section is a file-choosing widget that lets the user select a previous prediction to display on the screen or select an image from the operating system to run the weed detection algorithm on.

The next tab is the Config tab, which has more user controls and an activity log. The activity log displays all the commands requested by the user, such as motor controls, refreshing vitals, saving an image, etc., as well as notifications from the server. It provides a timestamp with each log so they can be tracked and reviewed. Along the right-hand side of the screen are commands to clear or export the activity log to a text file. Under that is a text box that allows the user to enter the amount of seconds to

manually open the water solenoid for. Since the system is able to detect water levels autonomously, this provides an option for the user to manually influence the watering of their garden. Under that are controls dealing with the server, providing connect and disconnect buttons, a button that manually retrieves current position and sensor values, and a text box that allows the user to send specific messages to the server, such as “PAUSE:” or “RESET:”.

The last tab is the About Us tab, which gives information about the group members and provides a link to the team’s GitHub to access source codes and more information about this project.

To test the performance of the app, each specific feature was selected and determined if they achieved the end user goals specified earlier. All three methods of requesting a motor command were tested individually, with the resulting outcomes compared to the expected outcomes. The arrow buttons activated the desired motors, moving the camera unit in the 1-dimensional direction requested by 400 units each press. The quadrant buttons requested movement to specific, hard-coded coordinates, so testing them involved measuring where the camera unit ended up and comparing that to the expected area. The text box for the exact coordinates were tested in a similar fashion as the quadrant buttons - entering specific coordinates and comparing where the camera was expected to go.

Testing the camera and image processing involved displaying the live feed and processed images on their respective tab displays. The live feed was tested by moving objects in and out of the frame of the camera and seeing if the same things would be displayed on the app. Afterwards, the clarity of the live stream was tested by controlling the motors to move the camera unit and testing the clarity of the livestream generated on the app. The capturing and AI integration were tested by taking a picture in a controlled area, where the amount and classification of the flora in the frame was known. The output of that prediction was compared to what was expected. The stats provided by the AI algorithm were also tested and verified that they matched the findings in the prediction image. Note, this test only verified that the displayed stats reflected what was found by the AI algorithm, not the actual accuracy of the weed detection; that testing was done separately.

The representation and collection of data was also tested. Interaction with the weather API was tested by requesting current weather data and comparing it to other verified sources of weather, such as the weather app on an iPhone and the weather data found when searching online. The collecting of values measured by the sensors were also tested by requesting data from the server through the “Refresh Vitals” and “Update data from Device” buttons, as well as through the textbox sending messages to it. All the tests proved to be successful. Ultimately, fulfilling the end user goals is the measure of a successful application.

IX. Hardware Testing

Individual tests were conducted on every electronic used in the project. Due to potentially lengthy shipping times, the parts received had to be tested to ensure that they were not defective in any way. All testing of the PCB design was built and confirmed functional on the breadboard before manufacturing.

For the ATmega microcontrollers, a known working Arduino development board was used to burn the bootloader of the chip, and then upload the “PinTester” program. The microcontroller being tested is then placed onto a breadboard with an LED hooked up to every GPI/O pin, which the program will set high, then low in successive order. This ensures that the ATmegas are not only able to be programmed but all their GPI/O pins are in working order.

The testing performed on the ATmega microcontrollers was also used for the ESP32 - CAM and Raspberry Pi modules. This is where the first issue with the ESP32 - CAM appeared as the onboard memory was extremely difficult to flash with the testing program. Since these modules would be handling live-streaming video and handling TCP/IP communications, separate testing programs were used to validate their performance. This second test of live-streaming video caused the team to abandon the ESP32 - CAM altogether as it was unable to stream 30 frames per second even at a reduced resolution. Later testing also revealed that the ESP32-CAM consumes almost 2 amps versus the Raspberry Pi being far more capable and only using 3.5 amps.

The first test conducted on the Nema17 stepper motors was to hook up the wires for the two coils to an LED and spin the motor by hand. This would cause the LED to light up if properly connected to the ends of a single coil. This not only ensured that the correct wires went to each coil winding, but that there was no current leakage between them.

After successfully testing the Nema17 motors, the A4988 motor driver chips were wired to them. Using the Arduino development board to send the signals needed for the Nema17 to do a full clockwise rotation, then a full counter-clockwise rotation. The motor drivers and motors

were then shuffled around to ensure that all drivers work with all motors.

Similarly the ultrasonic range-finders ordered were wired up to the Arduino development board. Using a testing program that would trigger the sensor and convert the signal to distance in centimeters, an object was placed at a known distance away to measure how accurate the sensors were. The sensors were then used to measure the amount of water, in gallons, in a bucket. Using the formula (1) and (2), the volume of water in the bucket was estimated.

$$Volume = \pi * Radius^2 * (Height - SensorDistance) * 231 \frac{inch^3}{gallon} \quad (1)$$

$$SensorDistance = EchoTime * 0.0135 \frac{inch}{second} * 0.5 \quad (2)$$

The I2C capacitive soil moisture sensor was also tested using the Arduino development board. Using the sample code provided by the manufacturer the team performed four tests: moisture in air, moisture in water, moisture in dry soil, moisture damp soil. This provided us the readout ranges to expect in the final demo testing.

The Mountain Ark mini pump and solenoid water valve were tested in the same manner due to time constraints, as well as the requirement they both operated the same due to the PCB being assembled and complete. Both devices were simply wired to a 12 volt power supply and observed if they turned on. In the case of the valve; the opening and closing caused by power on, power off was audible. Both devices were tested using a 5 gallon bucket to check water flow, this is when the solenoid was deemed insufficient for the project, rejected, and replaced by the mini pump.

X. Conclusion

One of the biggest lessons learned from this project: don't underestimate the importance of planning out small details. For example: more labels etched into the PCB and more space between wire connections would have made it easier to assemble the board which had to be done numerous times. Debugging was also slowed down during the project as the microcontroller had to be removed from the PCB in order to upload new programming. The lack of planning for small details also affected the software, where the addition of features throughout the project became increasingly difficult.

Overall the Auto-Gardener team was able to accomplish all the specifications set at the beginning of the project. Producing a prototype auto-gardening system that is able to water plants and identify weeds within the garden. With the greatest successes being: controlling the movement of the gantry through the desktop application, and the weed detection AI working on images obtained through the camera live stream.

References

[1] Microchip Technology Inc., "ATmega48A/PA/88A/PA/168A/PA/328/P," Microchip Technology Inc., 2020. [Online]. Available: https://ww1.microchip.com/downloads/en/DeviceDoc/ATmega48A-PA-88A-PA-168A-PA-328-P-DS-DS40002061B.pdf. [Accessed 28 11 2020].

[2] "Raspberry Pi 4 4GB," CanaKit, [Online]. Available: https://www.canakit.com/raspberry-pi-4-4gb.html?cid=usd&src=raspberrypi.

[3] Tindie inc., "I2C Soil moisture sensor", 2020. [Online]. Available: https://www.tindie.com/products/miceuz/i2c-soil-moisture-sensor/ [Accessed 2020].

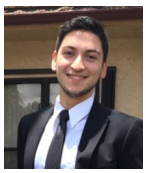
[4] "Motor Buyers Guide," Jamesco, [Online]. Available: https://www.jameco.com/Jameco/workshop/ProductNews/motor-buyers-guide.html. [Accessed 7 November 2020].

[5] C.-Y. W. H.-Y. M. L. Alexey Bochkovskiy, "YOLOv4: Optimal Speed and Accuracy of Object Detection," 23 April 2020.

[6] J. Solawetz, "Breaking Down YOLOv4," Roboflow, 4 June 2020. [Online]. Available: https://blog.roboflow.com/a-thorough-breakdown-of-yolov4/.

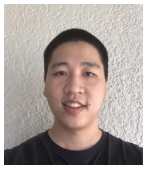
[7] "Programming Guide » Kv language", Kivy. [Online]. Available: https://github.com/YingLiu4203/LearningKivy/blob/master/Ch07%20Graphics/Graphics.md [Accessed 21 April 2021].

Biography

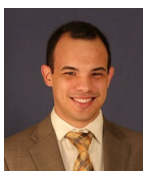


Wallace Borges is graduating with a Computer Engineering Degree from UCF. He enjoys doing front end UI design, as well as varied Python implementations. After graduating, he will start his position at Lockheed Martin as a Guidance, Navigating, and Control Engineer Associate.

Seana Falvey is graduating with an Electrical Engineering degree from UCF. Her interests lie in front end UI design, gardening and hopes to pursue a career in Power and Renewable energy.



Denver Lau is graduating with a Computer Engineering Degree from UCF. He is currently interested in computer vision, applications development, and cyber security.



Rafael Smith is graduating with a Computer Engineering Degree from UCF, spending most of his academic career exploring backend API and low-level hardware programming.