University of Southern Queensland

Faculty of Health, Engineering and Sciences

Development Of an Adaptive Control Mechanism to Enable Automated Irrigation on Gantry Robots

A dissertation submitted by

Corey Ahlers

In fulfilment of the requirements of

ENG4111 and 4112 Research Project

towards the degree of

Bachelor of Engineering (Honours) (Electrical and Electronic)

Submitted October, 2022

ABSTRACT

Currently there is a need for automated Space Agriculture systems to detect signs of early plant stress, to ensure both food safety and security on board space missions. Astronauts are hindered by the need to maintain constant communication to accurately diagnose plant stresses and maintain nutritional and psychologically beneficial plant life; so, a need for launch ready automation software has become apparent. Without a system capable of detecting early plant stresses to ensure food safety and security, further resources are required to address these problems manually. This increase of resources includes the time needed to facilitate communication with Earth based specialists to diagnose specimens, resulting in a loss of valuable time in preventing plant stress.

From the examined literature, the area of automated Space Agriculture has had little practical application with most automated agricultural projects such as irrigation and monitoring, performed on Earth through gantry topologies. Gantry topologies refer to a structure consisting of an overhead bridge beam supported by a platform, which allows a manipulator to move along multiple axis. Therefore, there is a knowledge gap in formulating a system that is capable monitoring and adapting to the condition of plant specimens through an automated adaptive control algorithm to maintain existing plants in microgravity environments.

Through this knowledge gap, the project aimed to answer if greenhouse gantry topologies are capable of reliably interpreting and adapting automatically in real time, facilitating plant life in isolated conditions. This question was answered through the development of an adaptive control plugin that was used in conjunction with a FarmBot Genesis XL gantry topology. The FarmBot topology was able to capture a total of 30 images of individual plant specimens which were used by the adaptive control plugin. The adaptive control plugin was able to process these plant specimens and adapt upon the change in specimen area with recommendations of applied water to each specimen.

Results showed that although the adaptive plugin was able to process data, inconsistencies were found in the processing and storage of plant area, specimen recognition and generated adaptive water values. These inconsistences formed recommendations on the further operation and development of the FarmBot and adaptive control topologies.

Keywords: Space agriculture, Machine Vision, Robotics, Automation, Adaptive Control.

University of Southern Queensland

Faculty of Health, Engineering and Sciences

ENG4111 & ENG4112 Research Project

Limitations of Use

The Council of the University of Southern Queensland, its Faculty of Health, Engineering and Sciences, and the staff of the University of Southern Queensland, do not accept any responsibility for the truth, accuracy or completeness of material contained within or associated with this dissertation.

Persons using all or any part of this material do so at their own risk, and not at the risk of the Council of the University of Southern Queensland, its Faculty of Health, Engineering and Sciences or the staff of the University of Southern Queensland.

This dissertation reports an educational exercise and has no purpose or validity beyond this exercise. The sole purpose of the course pair entitles "Research Project" is to contribute to the overall education within the student's chosen degree program. This document, the associated hardware, software, drawings, and any other material set out in the associated appendices should not be used for any other purpose: if they are so used, it is entirely at the risk of the user.

Certification

I certify that the ideas, designs and experimental work, results, analyses and conclusions set out in this dissertation are entirely of my own effort, except where otherwise indicated and acknowledged.

I further certify that the work is original and has not been previously submitted for assessment in any other course or institution, except where specifically stated.

Corey Ahlers

Student Number: [redacted]

ACKNOWLEDGEMENTS

I would like to acknowledge Dr Jacob Humpal and Dr Cheryl McCarthy for their continued support through the creation of this dissertation and for providing the FarmBot Genesis XL that is used throughout this project.

Table Of Contents

List Of Figures

List Of Tables

Glossary Of Terms

1. Gantry: A gantry refers to a style of robot that consists of a central platform which supports a bridge-like structure housing a manipulator.
2. Topology: A topology refers to the way a series of interrelated system parts are arranged.
3. FarmBot: The FarmBot is an open-source gantry topology that utilises an online user API to control the robot and its onboard hardware.
4. API: An API stands for an Application Programming Interface, an application used to interface with further software through system commands.
5. HSV: HSV stands for Hue Saturation Colour, which is a way of categorising a colour based on its hue and saturation characteristics.
6. RGB: RGB stands for Red, Green, and Blue, categorising colours based on how much of these colours there are within a colour.
7. Manipulator: A robotic manipulator is a mechanical device situated on the end effector of a robotic topology. Manipulators are responsible for providing a means for a system to interact with its physical environment.
8. Cross Slide: A cross slide refers to a motorised plate, facilitating travel along the Y axis of a gantry topology.
9. Drive Trains: Drive trains refer to an elastic belt that is situated across an axis motor. Drive trains allow machine components such as the cross slide to move along an axis.
10. LED: An LED refers to a Light Emitting Diode, an electrical component that produces light through an applied voltage.
11. Adaptive: Adaptive is an adjective that describes an ability to adapt. In this project, the word adaptive in “adaptive control plugin” refers to the automated behaviour style of the system.

Chapter 1: Background

From the early 1950's space travel has proven to be humanity's final frontier. Developments in technology and rocket design prevalent during the great space race between America and the Soviet Union have demonstrated the ability for humanity to reach the Moon (History.com, 2020). However, as the required travel distance grows, difficulties arise as the overall journey's travel time influences the sustainability of both physical and psychological systems.

These difficulties include bone loss, space adaptation syndrome and the various psychological impacts of remaining confined to a small area in zero gravity for prolonged periods of time (Wheeler, 2018). However, one of the lesser-known challenges of long-term space travel is the cultivation of plants. Plants rely heavily on the presence of gravity to disperse and drain water from its roots; in a microgravity setting, water holds its position within the soil, leading to saturation (NASA, 2010). Further issues include the inefficient designation of crew labour and speciality, the efficient use of physical resources such as energy water and nutrients, assured food safety, and the need for high crop yields (Keeter, 2020; Ngo, 2003). In addition, the same stresses experienced by plant life on Earth such as pathogens and parasites pose a problem. These current problems of efficient resource use and food safety problems seen in Space Agriculture are capable of being solved through machine vison and automatic robotic systems; the need for development of these systems has become apparent to progress humanity's goal for long term space travel.

To achieve this goal, it is necessary to draw upon inspiration from automated agricultural projects here on Earth and apply the same principles into the Space Agriculture setting (Shafi, 2020). Primarily, advancements in traditional automated agriculture can be divided into the categories of detection, maintenance, and irrigation, each of which pose unique challenges. For example, traditional means of mass crop irrigation prove to be ineffective in small-scale space agriculture settings due to how water behaves in micro-gravity (Nagura, 2019). Traditional irrigation also promotes plant pathogens through direct leaf exposure to moisture (Dixon, 2015). Therefore, it has become necessary to develop a self-sufficient system capable of addressing these various hurdles. The proposed research aims to utilise a gantry robot topology to inject water directly into the soil to irrigate plant specimens, maintaining the plant specimens through automated adaptive water application and area monitoring.

1.2 Project Development Scope and Aim

The main goal of this project is to supply adaptive control software for plant irrigation in Space; with an aim to provide initial steps in alleviating existing problems, while creating a foundation to combat future problems. The initial steps of this project focus on the automated monitoring and adaptive application of water to plant specimens to answer the question: Are greenhouse gantry topologies capable of reliably interpreting and adapting automatically in real time, facilitating plant life in isolated conditions?

Furthermore, this project aims to evaluate the viability of current commercial gantry topologies, investigating their physical capabilities to develop and implement a machine vision based adaptive control system. Evaluating the resulting topology based on factors such as accuracy and long duration reliability.

1.3 Overview of Research Objectives and Questions

A series of research objectives and sub-questions have been designed to streamline project development. These research objectives and sub-questions primarily focus on answering questions around administering and monitoring of plant specimens.

This project's overarching research question is investigated through the following sub-questions:

1. *What commercial gantry topologies are available for the project application?*
2. *Will the FarmBot's existing programs and technology suit the application?*
3. *Can the FarmBot API function in an isolated environment in conjunction with a third-party plugin?*
4. *Can the developed adaptive control plugin reliably measure plant area?*
5. *Can the selected gantry topology accurately and reliably apply water to plant specimens autonomously?*
6. *What future development is required to further facilitate true automatic plant analysis and maintenance?*

To answer these research questions, the following research objectives were created to form the basis of this project's methodologies. If time permits, further research objectives include the development of software systems to assist in the processing and refinement of data collection algorithms.

These objectives are listed as follows:

1. Conduct literature review on precision agriculture topologies, microgravity agriculture, current gantry agriculture applications and manipulators.
2. Identify core components of a remote automated microgravity greenhouse based on existing gantry topologies, including sensors, algorithms, manipulators, and server communications.
3. Identify a suitable topology for a microgravity greenhouse with gantry robot which can operate locally with reduced reliance on external communications.
4. Develop a localised, stable system software and hardware framework and integrate with an existing gantry robot.
5. Construct a functional prototype of the desired gantry topology and assess physical capabilities of the prototype for plant monitoring actions.
6. Test candidate methodologies for monitoring of plants over a minimum of one week to evaluate topology performance.
7. Gather and evaluate experimental data on topology components and report on functionality within physical, software and system level performance.

With further objectives if time is permitted:

8. Refine data collection and processing algorithms to increase validity and reliability of experimental data as required.
9. Further develop and implement graphical user interface to enhance functionality for future research of plant monitoring in a microgravity greenhouse with gantry robot.

1.4 Literature Review

The main objectives of automated greenhouse topologies are to implement an autonomous adaptive control system to generally manage, monitor, and maintain small-scale crops needs such as irrigation. Where the literature differs in comparison to the project is in the application of the project. Currently, there are a variety of simple gantry machines capable of operating here on Earth. It has been found that these gantry machines are capable of being used in conjunction with machine vision to further evaluate and broadcast basic plant conditioning (Takara, 2021). Furthermore, there are different forms of automated gantry robots that differ in functionality and robustness; for example, a single gantry system that can irrigate can also be capable of pruning plants (Zhao, 2021).

There are also specific Arduino and Raspberry Pi libraries and architectures used to make automation possible throughout the literature. These libraries and projects are open source and able to be implemented into the final project design providing a stable basis to build off, saving time on areas in development that would otherwise require additional resources to accomplish (Takara, 2021). The manipulator is also prevalent in the literature, with designs based on durability within the system itself. These designs take pre-existing styles of manipulators and change their material properties, allowing them to be used in other environments. In the context of multipurpose and interchangeable manipulators, the literature is sparse and only currently emerging (Tian, 2020). As it currently stands, multipurpose manipulators appear to be the best option due to their flexibility and efficiency in conducting tasks. By utilising a multipurpose manipulator robot topologies can switch tools mid process rather than breaking routine to change tools.

1.4.1 Current Gantry Robot Applications

In modern day robotics there are a variety of different applications that the gantry robot architecture has been used in; these applications are mostly in assembly operations such as construction cranes (Golovin, 2020), welding (Zych, 2021) and motherboard manufacturing. The gantry robot is a precise and accurate robotic topology; however, the use of these robotic topologies greatly decreases when applied to the agricultural industry. Current gantry applications in agriculture range in both precision and scale, where the two factors often directly correlate with each other. Larger scale indoor growing operations typically introduce an inaccurate quadrant spraying approach, aiming to reduce the amount of water lost in comparison to traditional irrigation methods by increasing its effective area. In smaller scale greenhouse operations, precision is of the utmost importance which compromises overall speed in favour of individual quality and accuracy (Wrest Park History Contributors, 2009).

The use of gantry robot topologies differs depending on both the application and manipulator. For example, for both small and large applications gantry robots allow operators to utilise a highly stable movable base to conduct machine vision monitoring of plants for pathogen activity and general specimen health (Sharath, 2021). This principle is also used for plant harvesting; however, this application is only seen in small-scale operations.

1.4.2 Current Manipulators

An important component of the gantry topology is its primary method of interaction with the physical environment. This component is a manipulator, which can take a variety of forms based on the environment the robot needs to interact with.

Presently the literature describes multiple possible manipulators that could be used for this project, which can be broken down into separate categories. Firstly, there are manipulators that can be used for a specific purpose and interchanged at the operators will (Baohua, 2020). These manipulators are currently seen in large and small-scale automated agriculture. However, these manipulator systems are often inefficient compared to multipurpose manipulators when the need for a separate manipulator is constant (Tian, 2020). For example, a dedicated plant watering manipulator that occasionally needs to perform monitoring or data collection tasks such as assessing soil moisture, will see a reduction in efficiency in comparison to a multipurpose manipulator capable of performing these tasks interchangeably.

In comparison, the literature strongly suggests that the use of a multipurpose manipulator is more efficient, and less resource intensive compared to interchangeable manipulators. Although, multipurpose manipulators can prove to be redundant in scenarios where the flexibility of on hand tools is not of a strong priority (Arian, 2017). This redundancy can prove to be ineffective in terms of cost and expansion of implementation. However, for pre-allocated tasks or systems that largely rely on efficient timing, multipurpose manipulators can prove invaluable.

1.4.3 Other Precision Agriculture

Throughout the literature there are many examples of different forms of precision agriculture robot topologies on Earth that differ from the proposed gantry robot topology design. One application is a form of non-intrusive cloud-based design of agricultural automation (Tan, 2016). Unlike the gantry robot topology, this form of robot is purely sensor and information based with no means of physically interacting with the environment, instead relying on peripheral systems and pre-existing hardware to act upon the issues flagged by the robot. Robots that operate through the IoT (Internet of Things) framework also fall into this category of precision agriculture (Maroli, 2021). These robots do not fit within the scope of the gantry design, as they are intended for larger scale agriculture. Due to the sensor driven nature of these systems, an implementation of this approach to precision agriculture would result in an overcomplication of the current application.

Within the literature there also variations of the traditional gantry topology that have been modified to suit a particular application. These forms of precision agriculture function similarly to the proposed gantry system; however, they differ greatly as their degrees of freedom increase (Zhao, 2016). An example of this concept is the optimisation process of a robotic arm for harvesting applications, testing topology effectiveness in relation to degrees of freedom as shown in Figure 1 (Bloch, 2018).

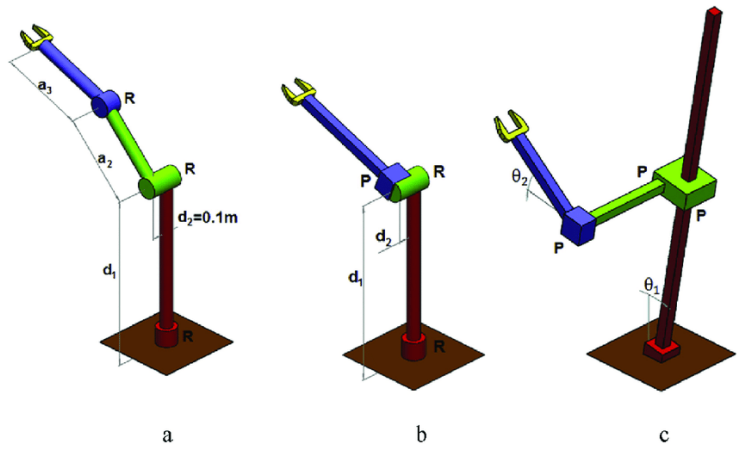

Figure 1: Virtual optimisation of robotic arms with varying degrees of freedom

By applying more degrees of freedom to the robot, it becomes more flexible in its capability to interact with plant specimens; however, its size and peripheral supporting structure grow respectively per additional degree of freedom. Although this design would suit the application based on what plant specimens were actively being maintained, the loss of modularity and the increased complexity of the overall system itself greatly hinder the system's ability to be easily implemented and maintained in long duration Space missions.

1.4.4 Effects of Microgravity in Precision Agriculture

There are a variety of studies and examples of prolonged observation of plant growth within varying magnitudes of microgravity; these studies demonstrate that there are many benefits experienced by plants during their growth cycle that occur through the introduction of microgravity. The first benefit microgravity has on plant growth is the significantly reduced amount of water and nutrient leeching required to achieve plant maturity (Maggi, 2010). This reduced rate of 90% moisture leeching leads to higher moisture retention within the soil of the plant, indicating that the required amount of water in irrigation systems will be 90% less than on Earth (Maggi, 2010).

The next beneficial side effect of plant growth in microgravity is their increased resulting maturity size. Through exposure to microgravity, plant specimens can grow taller due to the decreased gravitational resistance of transporting nutrients through the main stem of the plant. In addition, this reduction in gravitational resistance indicates that the distributed water can be utilised more effectively in microgravity environments (Shusaku, 2021). This demonstrated increase in growth rate in microgravity situations is corroborated in studies on mung beans (Shusaku, 2021) and corn, producing increased growth rates of up to 77% when compared to controls (Oluwefami, 2019).

Plants cultivated in microgravity also suffer detrimental health effects from the stress induced in this different environment. Through the introduction of a microgravity environment, plants can suffer from poor aeration of roots to chromosomal aberrations leading to challenges in plant growth and development (Wheeler, 2009).

1.4.5 Current Availability of Small-Scale Gantry Robots

Throughout the examined literature, there is a great divide in the availability of small-scale gantry robot topologies. Typically, available gantry topologies examined have a potential range of motion under 0.25 m^2 and are suited to desktop applications and high precision prototyping, rather than practical larger-scale automation (RS Components, 2022). In addition, most of the examined potential gantry architectures are governed by the company responsible for manufacturing the gantry topology, featuring the use of their own proprietary software and diagnostic tools. Beyond this 0.25 m^2 coverage, the availability of gantry topologies extends to factory and large-scale automation applications requiring custom ordering and fabrication (Sage Automation, 2022).

Due to limitations induced by this proprietary software, customisation and further development or integration of system API's and software are unachievable and drastically reduce the viability of using these gantry topologies in this project. The FarmBot, a consumer-available gantry topology, features open-source software and a potential out-of-box maximum area of 18 m^2. This large operating area, in addition to its user API, provided the most suitable platform for system customisation and implementation in the project.

1.5 Knowledge Gap

After evaluating the literature there are a variety of small hurdles that cumulate to form a profound knowledge gap. The main knowledge gap derives from the systems application itself. Gantry robots are currently capable of functioning in nominal conditions here on Earth and in Space. However, within the literature, there is a lack of examples of independent monitoring and control of plants in extreme isolated conditions, forming the basis of this projects research question and goals. Achievement of localised independency is the area of the current knowledge gap regarding the application of gantry systems in the Space Agriculture sector. Although there is evidence of gantry systems automatically watering crops; there is not a system capable of autonomously adapting the amount of water applied to a plant based on its current circumstance. This system would allow for further independence of the gantry robot system and further the ideal goal of a self-sufficient system capable of maintaining plant life for the 3-year journey to Mars.

There is also a considerable knowledge gap in the use of multipurpose manipulators in small-scale space agriculture setting. There is evidence for multipurpose manipulators being used in both automated and manually controlled agricultural settings, however these settings are generally large in scale, where the use of a multipurpose manipulator is in harvesting the crop. Therefore, a system that is capable monitoring and adapting to the condition of plant specimens through an automated precise irrigation manipulator forms the basis of the knowledge gap.

This research will establish whether a combination of localised independence and multipurpose manipulation can achieve the self-sustaining plant agriculture required for long duration space travel.

1.6 Dissertation Outline

With the knowledge gap and aim of the project identified, this dissertation addressed these aspects through a series of chapters leading to a final discussion and summary. These chapters have been organised to linearly outline the formation of the adaptive control plugin to its application to live data.

1. Chapter 2 - Topology Construction: This chapter describes the assembly of the gantry topology and its camera feed.
2. Chapter 3 – Adaptive Control Development: Chapter 3 builds upon chapter 2, describing the development of the adaptive control plugin and its algorithms.
3. Chapter 4 - Simulated Testing: After developing the adaptive control plugin, chapter 4 details the plugins initial application to a simulated test scenario.
4. Chapter 5 – Results: Chapter 5 combines chapters 2 and 3, gathering live data using the gantry topology and processing it using the adaptive control plugin.
5. Chapter 6 – Discussion: Chapter 6 discusses the results obtained within chapter 5, forming recommendations on the functionality of the adaptive control plugin.
6. Chapter 7 – Summary and Conclusion: Chapter 7 concludes the dissertation with a summary of the project, evaluating how chapters 2 through 6 have addressed the projects objectives.

Chapter 2: Topology Construction

Before describing the adaptive control component of the project; Chapter 2 recounts the construction of the FarmBot topology and the mobile bed utilised throughout Chapter 5 in Figures 21 and 22. Furthermore, Chapter 2 details the problems that occurred throughout construction process and initialisation, in addition to the steps taken to resolve these problems.

2.1 Overview of FarmBot Topology Components

The FarmBot system in its entirety can be reduced into two categories. The first category of the FarmBot system is the mobile bed that the gantry topology is built upon, which consists of two RACK-IT™ shelving units joined with four adjustable threaded rods. Whereas the second category of the FarmBot topology is the FarmBot components and electronics that are built upon the shelving units.

2.2 Construction of Mobile Bed and FarmBot Topology

Construction of the FarmBot gantry topology began with the assembly of each of the two RACK-IT™ shelving units. These shelving units were set to hold the main bed the FarmBot at the centre of the shelving units, providing an ergonomic position to make alterations to the shelving units.

Next, four equally spaced holes were drilled into the inner side of each of the shelving units to house four 500mm threaded rods. These threaded rods allow the shelving units to be incrementally adjusted to suit the desired 1.5 m Z axis of the FarmBot gantry topology.

After the shelving units were in the correct position, installation of the FarmBot tracks began. For installation to be possible, permanent alterations to the provided FarmBot tracks had to be made to satisfy the dimensions of the shelving units. To accomplish this installation, the length of the shelving units was recorded at 2.65 m and the difference between the provided 3 m tracks was calculated as 305 mm. This offset was measured, and the required cut was performed (Figure 2).

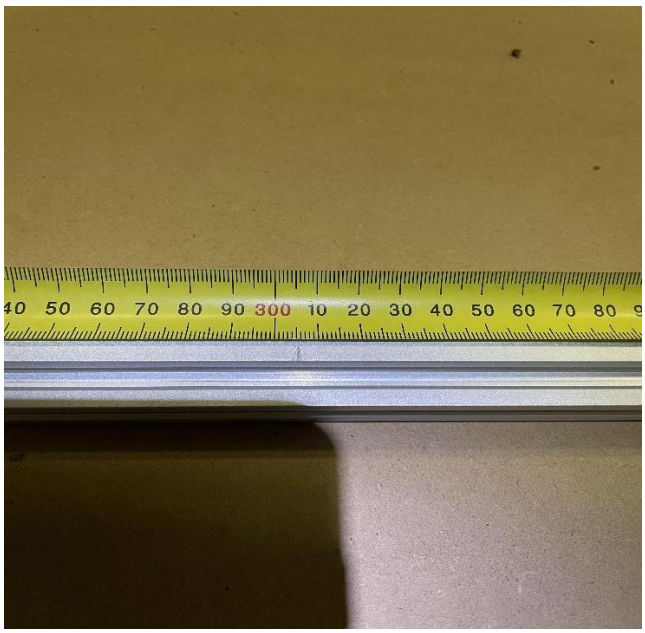

Figure 2: Measurement of the required cut

2.2.1 Attachment of the Cross Slide

After construction of the mobile bed and the FarmBot gantry tracks, the topology Z component was installed. The construction of the Z component began with construction of the FarmBot wheel plates according to the provided FarmBot documentation.

However, when installing each of the two-wheel plates, it was discovered that there was no allocated room for the wheel plates to reside in. This limitation meant that the FarmBot gantry tracks had to be raised to accommodate for each of the wheel plates, which was accomplished by incrementally raising the track level to suit the wheel plates shown (Figure 3).

Figure 3: Incremental raising of the gantry tracks.

After raising and determining that the FarmBot wheel plates could now travel along each of the gantry tracks, the shelving units were locked into place. This allowed the horizontal Y bar to be installed across each of the supports.

Next, the Cross Slide plate responsible for housing both the Z and Y motors was assembled following the provided FarmBot documentation. However, much like the installation process of the FarmBot gantry tracks, it was discovered that there was not enough room within the FarmBot working environment to accommodate for the height of the Z axis.

This limitation resulted in a deconstruction of the FarmBot gantry topology and tracks, as the shelving units had to be readjusted to accommodate for the lack in height. This re-adjustment allowed the FarmBot to successfully house the Z axis component.

2.2.2 Attachment of Drive Trains

After the cross slide and Z axis bar were successfully installed, the final stage in completing the mechanical design of the FarmBot topology was to install the drive trains. This was done by threading the provided G2 timing cables through the beams situated on the wheel plates and along the tracks of the topology. Each end of the G2 timing cable was then locked to either end of the FarmBot tracks, ensuring the cable was tight. This process was then repeated on the Y axis bar through the cross slide which is shown in Figure 4.

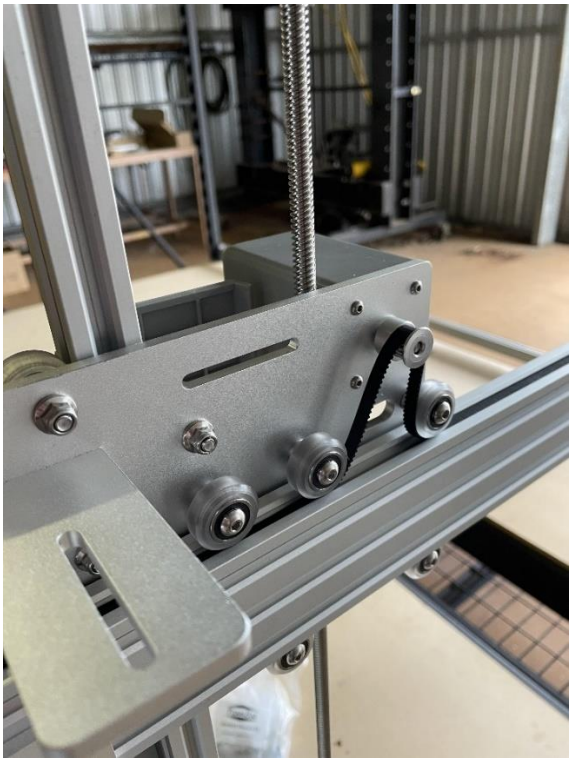

Figure 4: Installation of the Y axis Drive Train.

2.3 Implementation of System Electronics and Plumbing

The final stage of assembling the main FarmBot topology was the assembly of the systems cable carriers and connection of the system electronics. This began with altering the length of the Y axis cable carrier to suit the smaller size of the gantry topology. Originally three metres in length, the Y axis cable carrier and electronics were reduced to one and a half metres for the construction of the FarmBot. However, due to how the cable trays were constructed, they were able to be simply unclipped and repositioned to meet the new size requirements of the FarmBot.

After adjusting the Y axis cable carriers and installing the X and Z axis cable carriers onto each of the corresponding rails, rubber tube was positioned along each cable carrier forming an uninterrupted pipeline to the gantry's water supply.

2.3.1 The Z axis Electronics

The first electronics to be installed into the system were the electronics situated along the X axis. These consisted of the data cable for the universal tool mount, the camera, the vacuum pump, the Z axis encoder, and motor cables. Each of these cables were then threaded through the Z axis cable carrier with the Z axis motor and encoder cables installed onto the Z axis motor unit.

2.3.2 The Y axis Electronics

Next, the Y axis electronics were positioned within the Y axis carrier cable to be connected to the Z axis counterpart. These electronics consisted of extensions for the Z axis motor and motor encoder, an extension of the universal multitool cable, an extension of the vacuum pump cable and the Y axis encoder and motor cables.

Like the process used in installing the Z axis electronics, each of these cables were threaded through the cable carrier with the Y axis encoder and motor carriers installed into the Y axis motor situated on the system cross slide (Figure 5).

Figure 5: Completed installation of the Y axis cable carrier and cables.

2.3.3 The X axis Electronics

The X axis electronics unlike the Y and Z axis electronics are not situated within the X axis cable carrier situated along the left side of the FarmBot topology. This cable carrier is reserved as the primary track for the systems power supply and water source to connect to the system electronics cabinet (Figure 6).

Figure 6: The completed installation of the X axis cable carrier and cables.

Instead, the X axis electronics consist of two pairs of motor power and encoder cables threaded through the cable carrier supports of the Y axis bar. This allows the system to maintain room and mitigate flow issues within the Y axis cable carrier. Finally, these cables were threaded into the electronics cabinet resulting in the arrangement in Figure 7.

Figure 7: The completed FarmBot Electronics Box.

2.4 Initialisation of The Physical Topology

After completing the physical construction of the FarmBot, the system was initialised. It was through the initialisation process that multiple issues with the gantry topology were identified. To summarise these problems and the solutions derived to overcome them, the following table summarises these problems:

Table 1: An overview of each of the identified problems within the FarmBot topology and their resolutions.

Problem Encountered	Resolution
Camera Failure	Upon initialisation, the provided endoscopic camera failed to establish connection to the FarmBot's on board Raspberry Pi. This prompted the use of a traditional webcam to be implemented.
Z-Axis Motor	During the initialisation process, the Z-axis motor failed to reliably drive the Z-axis cross slide. This failure caused the Z-axis motor to consistently stall, remaining at a fixed position. This was resolved by disconnecting the Z-axis motor to fix the Z axis at a constant height for monitoring.
Connection to FarmBot	When working to establish a connection to the FarmBot API, there were multiple issues with maintaining a reliable internet connection through an iPhone hotspot. It was discovered that iPhone hotspots will sporadically pause the connection to save internet data. This issue was resolved through the application of an internet modem and pre-paid data plan.
Bilge Pump	After establishing a reliable connection to the FarmBot, it was found that the intended Bilge Pump was unable to remain in a fixed position due to the length of the provided cabling. This was resolved by placing tanks at each section of the physical topology, moving the bilge pump to the appropriate tank when required.
Inverted Y Axis	While performing initial tests of the FarmBot's functionality, it was found that the Y-axis motor was inverted. This issue was resolved by inverting the polarity of the motor using the FarmBot's settings.

Chapter 3: Adaptive Control Development

The adaptive control plugin was developed in parallel with the FarmBot topology and mobile bed construction. Chapter 3 details the development of the adaptive control plugin, including the design choices made and functionality.

3.1 Libraries and Functions

3.1.1 Determination of Program Language

When beginning the creation of the adaptive control element in this project, the first stage of development began with the identification and implementation of necessary libraries that would make image processing and calculation possible. In addition, it was also important to decide on the primary language that the program was to operate in.

Analysing potential programming languages that the plugin could operate within, there were two languages that the system could operate in that could utilise machine vision libraries, these were C++ and Python. Although C++ is capable of compiling and parsing data faster than Python, the decision of what language should be used was decided upon the context of where the plugin is situated within the FarmBot API and backend servers.

The adaptive control plugin is situated within the front end of the FarmBot server, which allows the plugin to bypass most of the FarmBot's data management responsibilities. This meant that after implementation of the plugin, its main responsibilities were to manage and parse its own calculated adaptive data between the FarmBot API. Given the context of the programs operation, Python was the language employed by the plugin due to its automatic data management, simplistic language formatting and ease of implementation into embedded systems.

3.1.2 Determination of Machine Vision Libraries

After determining the programming language the plugin would operate in, a machine vision toolbox had to be decided on for the identification and measurement of plant area. Ideally, the machine vision toolbox had to be open source for ease of access and be capable of development within the Visual Studio Code development environment.

Through analysis of possible machine vision toolboxes supported by the Python language, the decision of a machine vision toolbox was between two equally applicable toolboxes. The first of these was Open CV, an open-source computer vision toolbox capable of utilising the Python language with over 2500 optimized algorithms in vision detection and machine learning (OpenCV, 2022). The second toolbox considered for the project application was PyTorch CV, an open-source branch of PyTorch specifically designed for computer vision applications in image classification, segmentation, and detection (Pytorchcv, 2022).

Although both toolboxes are capable of implementation into the plugin design, Open CV was utilised for the project due to its large community support and range of development resources that made implementation of the toolbox into the adaptive control design simple.

3.2 Machine Vision Implementation

After establishing and initialising the Open CV toolbox in Visual Studio Code, images received by the FarmBot's onboard camera could now be analysed. Analysis of images received by the FarmBot begins by reading each image using the "imread" function, which formats the colour data and size into an array readable by the program.

These images are then converted from their traditional Red Green Blue classification (RGB) into a Hue Saturation Value (HSV) colour format. Hue Saturation Value colour encoding allows the system to identify colours in varying levels of light and saturation as opposed to a combination of red, green, and blue. By using HSV formatting as opposed to RGB formatting, images received by the FarmBot are capable of analysing colour in varying conditions of light and potential glare experienced by the camera due to the FarmBot's onboard LED lighting.

The next stage in plant detection within the received FarmBot images was to create a high and low green threshold. This threshold allows the plugin to evaluate each HSV colour value in the received images and create a binary image based on which individual pixels fell within the threshold range. Later in the development of the adaptive control plugin, these green thresholds had to be changed to allow detection for paler plant specimens rather than brighter green specimens.

By creating a black and white binary image, the plugin could evaluate the contours of the identified plant through edge detection of the black and white image. Through identification of the plant's contours, calculation of its area is much faster due to the mitigation of noise present in the received full colour images of the FarmBot. An example of the completed product of the programs first stage, used in initial development is shown in Figure 8.

Figure 8: Calculated Contour Overlayed onto Received Example Image

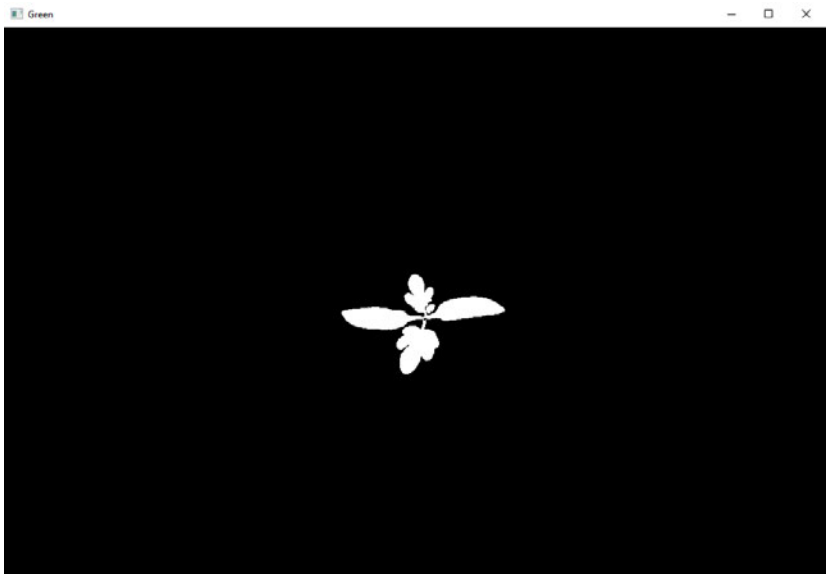

Figure 9: Binary black and white Image used for contour creation and area calculation

3.3 Calculation of Area

After completion of the first stage of the program, the second stage used the previously created contours to determine the area within them. This is done by initialising a new function containing a FOR loop equal to the size of the number of contours present within the image.

For each iteration completed by the FOR loop, the area of a contour is evaluated using OpenCV's "cv.contourArea" function, which returns the contour area in pixels and stores the result into a pre allocated "Areas" array. For every iteration the determined area is appended to the array, with the final iteration of the loop returning the final "Areas" array.

3.4 Specimen Identification and Information Formatting

After the plant specimen has been identified with its area calculated, the next section of the plugin is responsible for the identification of what plant specimen is currently being examined. To identify which plant specimen was under examination there were a variety of methods that could be employed, with each method equally viable.

For this application, there were two possible methods that the plugin could use to identify a plant specimen. The first method utilised the application of numerical digits to identify each plant specimen, due to its easy replication and unique shape for feature detection. However, for this method to be implemented successfully, a machine learning process would have to be employed to iteratively train the plugin to accurately recognise the digit present.

The second method instead utilises a dice pattern to interpret what specimen is under examination, using the same feature detection and contour recognition methods used in calculating the area. These dice patterns feature black adequately spaced circles on a white background as shown in Figure 10.

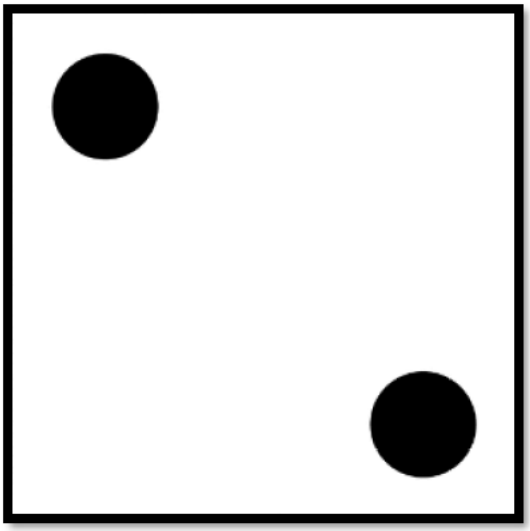

Figure 10: An example of a specimen label featuring a dice pattern representing "2".

This allowed the system to still use an easily replicable labelling format while avoiding the need for additional machine learning requirements. Through identification of each plant specimen, the calculated area of any specimen could be attributed to a corresponding number.

3.5 Adaptive Control Implementation

3.5.1 Generation of Initial Water Values

After each plant specimen could be recognised and organised based upon its assigned dice pattern, the initialisation of the adaptive control and main functionality of the plugin was possible. Initialisation of the program begins through the generation of 5 randomly generated water percentage values within the range of 25% to 75%, with the lowest and highest values within the array altered to guarantee each initialisation contains a 25% and a 75% specimen. The generated values are then stored within an array, which is combined with a separate array containing the specimen numbers and corresponding areas.

The iterative design of the adaptive control plugin can use these generated initial values as a basis for its iterative design. This design refers to the programs ability to adapt to new information each iteration; where an iteration refers to how many times the camera has passed over a specimen.

3.5.2 Second Iteration and Adaptive Control

The adaptive control of the plugin begins after the FarmBot API has received the initial water values and administered them. Using the FarmBot API, each plant specimen is analysed following the methods used for classification.

These new values are stored in an array and compared with the previously submitted initial values, allowing the change in plant area to be analysed by finding the difference between each iteration. The resultant difference of a plant specimen's area replaces its previous area value in the current iterations array, preparing it for analysis.

Each plant specimen has its water value characteristics evaluated in two ways. The first characteristic is where its applied water percentage lies relative to the full 25% to 75% range. This is where the

adaptive control algorithm makes its first assumption, that following a normal probability distribution, the optimal water value for the plant specimen is most likely within the 45% to 65% range.

Following this logic, the algorithm then performs a series of checks to categorise an appropriate threshold to further control water application. The algorithm begins by assessing if the currently examined plant specimen's water level is situated above or below 50%. This is then compared with the current plant specimens' difference in area to determine if the current specimen should have its applied water increased or decreased.

Next, the current plant specimen has its potential water application reduced to a window of ±10% within the limits of the 25% to 75% range. The algorithm then takes all the acquired information and produces one of four decisions in Figure 11.

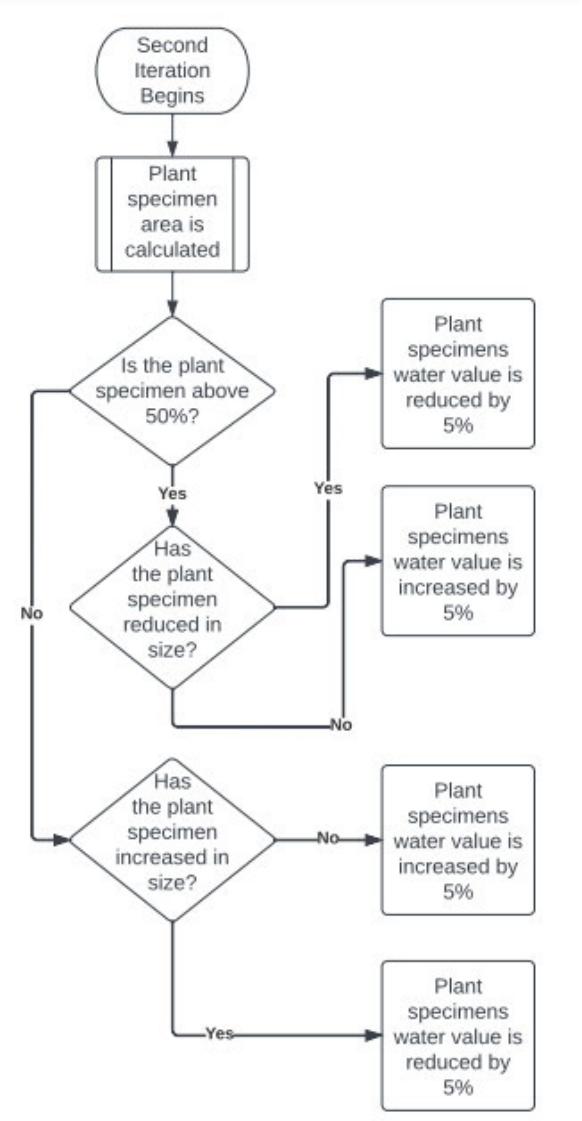

Figure 11: All potential decisions within the initial stages of the adaptive control plugin.

After every plant specimen has been evaluated, each applied water percentage is altered to reflect the decision produced by the algorithm and submitted to the FarmBot API.

3.5.3 Third Iteration and Continued Control

When the plugin reaches its third iteration, the adaptive control process is altered to reflect the smaller scope of each plant specimens applied water percentage. The third and subsequent iterations of the adaptive control algorithm begin in the same way as the previous iterations before it, analysing the growth and reduction of each plant specimen based on its applied water percentage.

However, when forming a decision on updating the plant specimens applied water percentage the algorithm now operates on a new set of decisions (Figure 12):

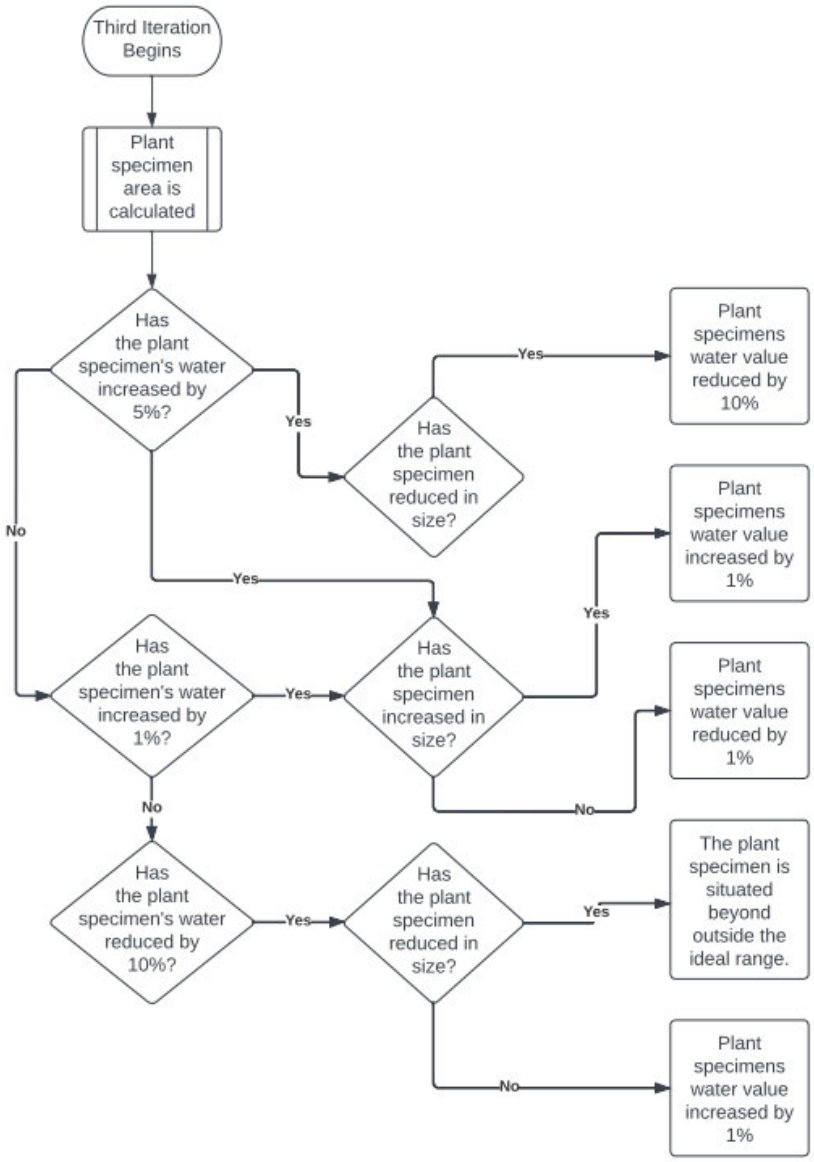

Figure 12: Updated potential decisions as the adaptive control moves into continued control.

Following the new set of potential decisions, the algorithm continues to evaluate each plant specimen, adjusting its applied water percentage accordingly. The adaptive control of each plant specimen is considered as complete once the algorithm has moved between increasing and decreasing the plant specimen's water by 1% two times. This is because the program cannot reduce its resolution any further, achieving its most precise result.

3.6 HTML Wrapping and data parsing

After the adaptive control algorithms and image functions had been created, the final stage in completing the adaptive control plugin was to build the framework needed to parse data from the FarmBot API to the plugin. To achieve this, additional libraries were applied to the plugin to allow the plugin to be hosted locally through a HTML address.

3.6.1 PyJWT

The first stage in utilising incoming information from the FarmBot API was to be able to decrypt the received token to establish a secure connection between both programs. To facilitate the decryption process PyJWT was utilised within the plugin. PyJWT is an open-source Python library specifically designed for the purpose of receiving and decrypting Javascript tokens based on the RFC 7519 web token standard.

3.6.2 Flask HTML Hosting

With the system able to decrypt incoming tokens and receive the JavaScript information sent from the FarmBot API, the final stage in facilitating a connection between the two programs was to wrap the plugin as a HTML program with a corresponding local address. To achieve this stage Flask was utilised to generate the local HTML address and host the adaptive control plugin. Flask is an open-source library supported by the Python language, capable of HTML web design and web hosting.

Chapter 4: Simulated Testing

Chapter 4 of this report takes the completed adaptive control plugin and applies it to a series of 6 different butterleaf plant specimens. Each of these plant specimens demonstrate the characteristics of a unique growth scenario, testing the functionality of the adaptive control plugin in a range of scenarios.

This section also discusses the inconsistencies found through simulated testing; further recommending methods of resolving these found inconsistencies.

4.1 Simulated Testing Methodology

Through the completion of the adaptive control plugin, the first phase of testing could begin. This phase of testing consisted of taking pre-fabricated pictures and inputting them into the adaptive control plugin to visulise how the program would respond to live data. These images were obtained through a video timelapse of a buttertleaf lettuce specimen's growth over 21 days. Using these images, the dice pattern used for specimen identification was placed in the top right corner of the image to allow the program to label the image shown in Figure 13.

Figure 13: An example of a speciemen used in the adaptive plugin.

Consisting of 5 specimens, the adaptive plugin iterated through each specimens growth 6 times. This resulted in a total of 36 images that were evaluated by the adaptive control plugin, with each iteration saving the current state of the specimens growth and water application based on its performance. To achieve this, the adaptive control plugin searches within the directory *"Official_Test"* in Figure 14, and cycles through the contents of each folder within Figure 15.

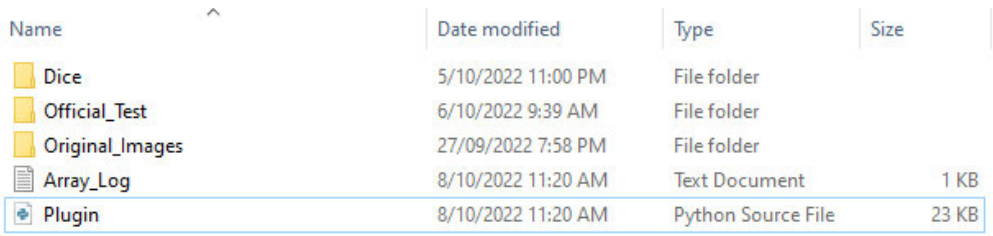

Figure 14: The folder the plugin draws from.

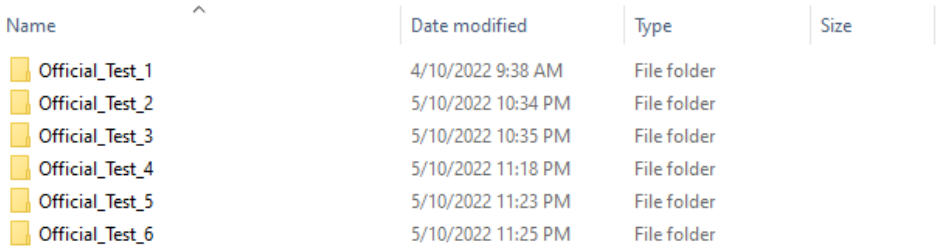

Figure 15: The contents of the "Official_Test" folder.

These folders represent the simulated stages of growth that each specimen will undergo, each holding 5 images for the simulated 5 plant specimens. To test how the adaptive control plugin would react, each plant specimens growth pattern was organised in a way that would simulate a possible plant reaction based on the initially prescribed water amount. An overview of each plants characteristic is described below (Table 2):

Table 2: Overview of each specimens designed characteristics.

Plant Specimen	Simulated Characteristic
1	Plant specimen 1 was curated to represent the best case scenario of ideal growth. Within each iteration, this plant specimen does not experience any reduction in size, continuing to grow until it reaches the systems complete state.
2	Plant specimen 2 was curated to represent overwatering. In this scenario, plant specimen 2 grows consistently until the third iteration, where it decreases in size until the final iteration. This specimen was designed to test the adaptive controls ability to handle drastic changes in growth past the third iteration.
3	Plant specimen 3 was curated to represent a fluctuation in growth toward the later stages of simulation. Plant specimen 3 shows continual growth until the third iteration, where it will fluctuate in size until the final iteration. This specimen was designed to test the adaptive controls ability to regulate fluctuation past the third iteration.
4	Plant specimen 4 was curated to test the adaptive controls ability to calculate larger plant sizes as well as identify specimen numbers in fluctuating backgrounds.

	In this scenario plant specimen 4 shows rapid growth in differing backgrounds until the final iteration.
5	Plant specimen 5 was curated to test the adptive controls ability to recognise smaller changes in size. Within the program, plant area is rounded to the nearest 0.5, 0 or 1. This allowed the plugin to demonstrate how it handles changes in size, which can lead to false completion conditions.

Once the program had completed its analysis over the 5 iterations, the results were manually entered into an Exel™ spreadsheet and the results were obtained.

4.2 Simulated Testing Results

After collating the image datasets for each of the plant specimens and processing them with the algorithm, the received data was processed in Excel™ to obtain the following results (Figure 16):

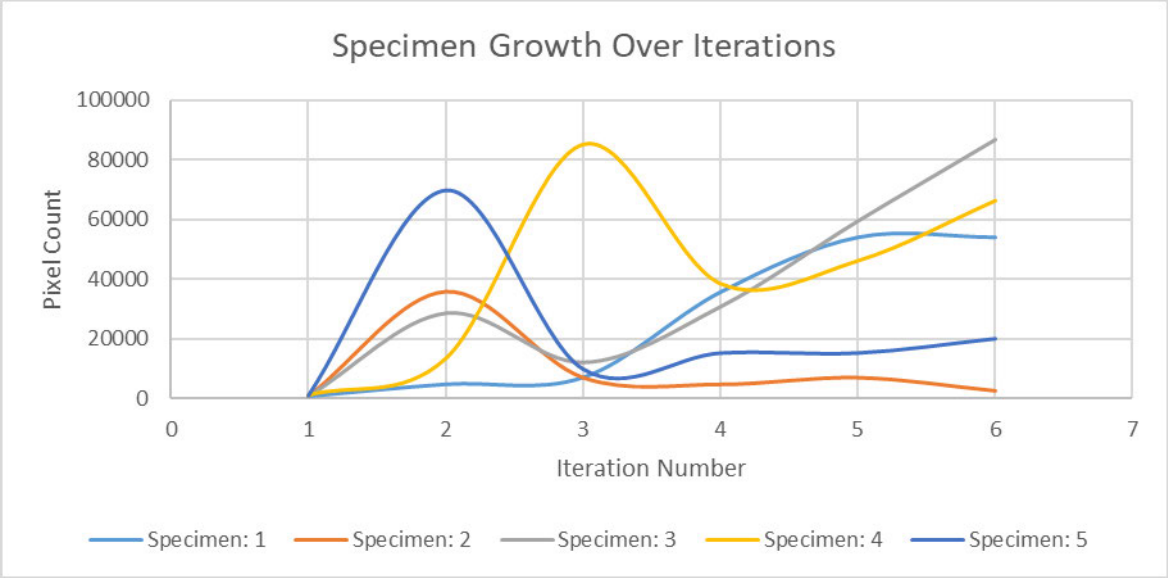

Figure 16: Obtained growth over specimen iterations.

When initially implementing the dataset into the adaptive control plugin, the system was able to accurately perceive each specimen's growth patterns, store them, and output them accordingly. As shown in the graph above, each specimen correlates with its designed purpose, demonstrating a broad range of potential growth cycles for each specimen.

Next, the applied water amount for each specimen over each iteration was obtained through the output of the adaptive control plugin and inputted into excel to receive the following output (Figure 17):

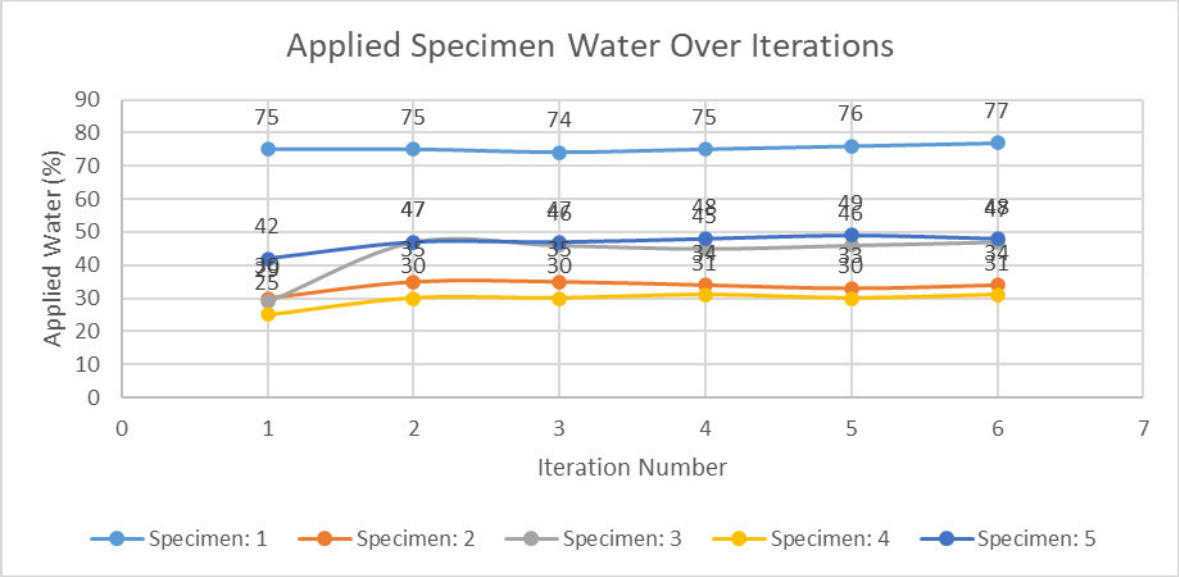

Figure 17: Applied water over specimen iterations.

After analysing the data obtained from the program, there are two discrepancies that can be seen in the output of the program. Firstly, for specimen 1, the applied water surpassed the hard limit of 75% from the initial dataset, achieving a final applied water level of 77%. Due to this abnormality, upon completion of the simulation the pre-determined completion state planned for specimen 1 was unable to be reached.

In addition, specimen 3 appears to spike by a large amount between the first and second iterations, achieving an increase of 18% in applied water level. This abnormality greatly surpasses the initial expected applied water level incrementation of 5% doubling it to reach the same applied water levels as specimen 5, which can be seen in comparison to other specimens (Table 3).

Table 3: Final simulated results table, showing discrepancies in the change in water.

Simulated Thesis Inputs And Outputs					
Iteration No.	Specimen No.	Observed Area	Percieved Area Change	Prescribed Water (%)	Change in Water
1	1	802	0	75	0
1	2	904.5	0	30	0
1	3	755	0	29	0
1	4	1056 5	0	25	0
1	5	843	0	42	0
Iteration No.	Specimen No.	Observed Area	Percieved Area Change	Prescribed Water (%)	Change in Water
2	1	4859 5	4057 5	75	0
2	2	35773.5	34869	35	5
2	3	28595	27840	47	18
2	4	13339	12282.5	30	5
2	5	69665	68822	47	5
Iteration No.	Specimen No.	Observed Area	Percieved Area Change	Prescribed Water (%)	Change in Water
3	1	7128	2268 5	74	-1
3	2	7128	-28645.5	35	0
3	3	12106.5	-16488.5	46	-1
3	4	85052.5	71713.5	30	0
3	5	9718	-59947	47	0
Iteration No.	Specimen No.	Observed Area	Percieved Area Change	Prescribed Water (%)	Change in Water
4	1	35773.5	28645.5	75	1
4	2	4859 5	-2268.5	34	-1
4	3	30793.5	18687	45	-1
4	4	38374.5	-46678	31	1
4	5	15123	5405	48	1
Iteration No.	Specimen No.	Observed Area	Percieved Area Change	Prescribed Water (%)	Change in Water
5	1	54115.5	18342	76	1
5	2	7128	2268 5	33	-1
5	3	59486.5	28693	46	1
5	4	46075.5	7701	30	-1
5	5	15123	0	49	1
Iteration No.	Specimen No.	Observed Area	Percieved Area Change	Prescribed Water (%)	Change in Water
6	1	54115.5	0	77	1
6	2	2721 5	-4406.5	34	1
6	3	86742	27255.5	47	1
6	4	66208	20132.5	31	1
6	5	19933.5	4810 5	48	-1

4.3 Observed Simulated Testing Inconsistencies

After simulating the adaptive control plugin using the 5 different plant specimen scenarios, there are inconsistencies in the program that become apparent. These inconsistencies stem from how the adaptive control plugin identifies and stores specimen data within the program. Analysis is completed based on the current iteration within the program. This iteration count is the basis of scheduling and organisation within the adaptive control plugin.

When simulating the program, it was discovered that if there is considerable noise within the captured image of the plant specimen, the system will identify false positives of what appear to be dice pattern dots. This misidentification leads to essentially recording the same plant specimen twice within the final array with a different size but with the same prescribed water value as the specimen it is mimicking.

This misidentification of plant numbers does not appear to become a problem until the program moves to store the previously held values for comparison later in the program. Although the adaptive control plugin can recover the correct specimen number of the plant specimen, the residual corruption left by the previous iteration can cause the plant specimen to increase its prescribed water value. This increase in water value causes the plant specimens prescribed water value to misplaced by more than 10 percent shown below in Figure 18.

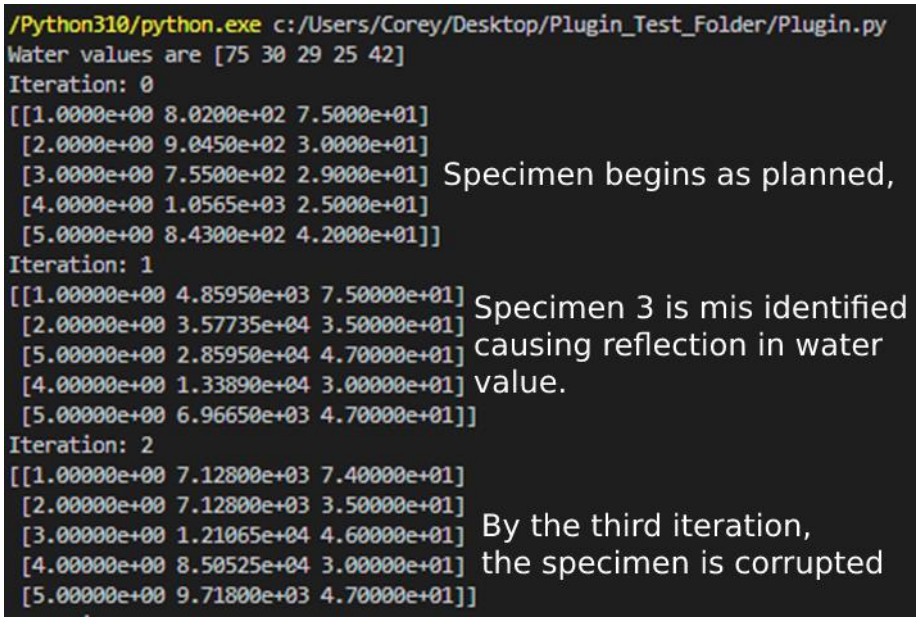

Figure 18: A sample output of the adaptive control plugin, demonstrating data corruption.

Another flaw seen in the execution of the adaptive control plugin can be derived from how the program rounds its output values. When the system is operating, the calcualted result will be rounded to the nearest whole or half number which has resulted in an error seen in the applied water levels of specimen 1.

```
    # Applied 5% more, Increased
if (Selected_Water == Increased_5): # Theres been a visable increase in water
            # Increase in size and applied 1%
        if (Previous_Area < Selected_Area):
            #print("Passed Size Check")
            if ((Calculated_Image_Array[2] + 1) >= 75):
                Calculated_Image_Array[2] = Calculated_Image_Array[2]
            else:
                Calculated_Image_Array[2] = Calculated_Image_Array[2] + 1
```

Figure 19: A sample threshold of the adaptive control plugin.

Within the adaptive control plugin, there are many points within the program where the system must evaluate if an application of water will place the currently applied water outside the scope of initial water values shown in Figure 19.

On the condition that anymore applied water will break this condition, the program blocks the system from applying anymore water. It is because of the “>=” (greater than or equal to) operator that the system can surpass this check by producing a value between 70 and 75. The program evaluates that the current applied water does not surpass 75, increasing its value and rounding the result.

This inconsistency in the adaptive control plugin could be resolved through increasing the resolution of the current thresholds in the program to include floating values. By including floating values, this would allow the program to evaluate floating points within values, preventing false positives. Although, the addition of floating-point values would also increase the amount of memory required for processing.

Chapter 5: Results

Chapter 5 utilises the created adaptive control plugin and applies it to living plant specimens. The methodologies of the final test are discussed, and the results of the test are evaluated. This chapter discusses how the inconsistencies found through simulated testing translate into a real scenario, with recommendations made on further mitigating these inconsistencies.

5.1 Overview of Final Testing Methodology

After the completion of the adaptive control plugin and the FarmBot gantry topology, final testing could now be performed using the FarmBot and the adaptive control plugin. For the final test, the FarmBot topology was arranged to house a total of 30 lettuce specimens. Twelve of these lettuce samples were chosen to be cos lettuce variety to represent a smaller lettuce sample, while the remaining 18 were chosen as butterleaf lettuce variety, to reflect the same plant specimen used in simulated testing. These plants were then laid out in groups of 6 over 2 beds with their labels placed next to the corresponding specimen shown in Figure 20.

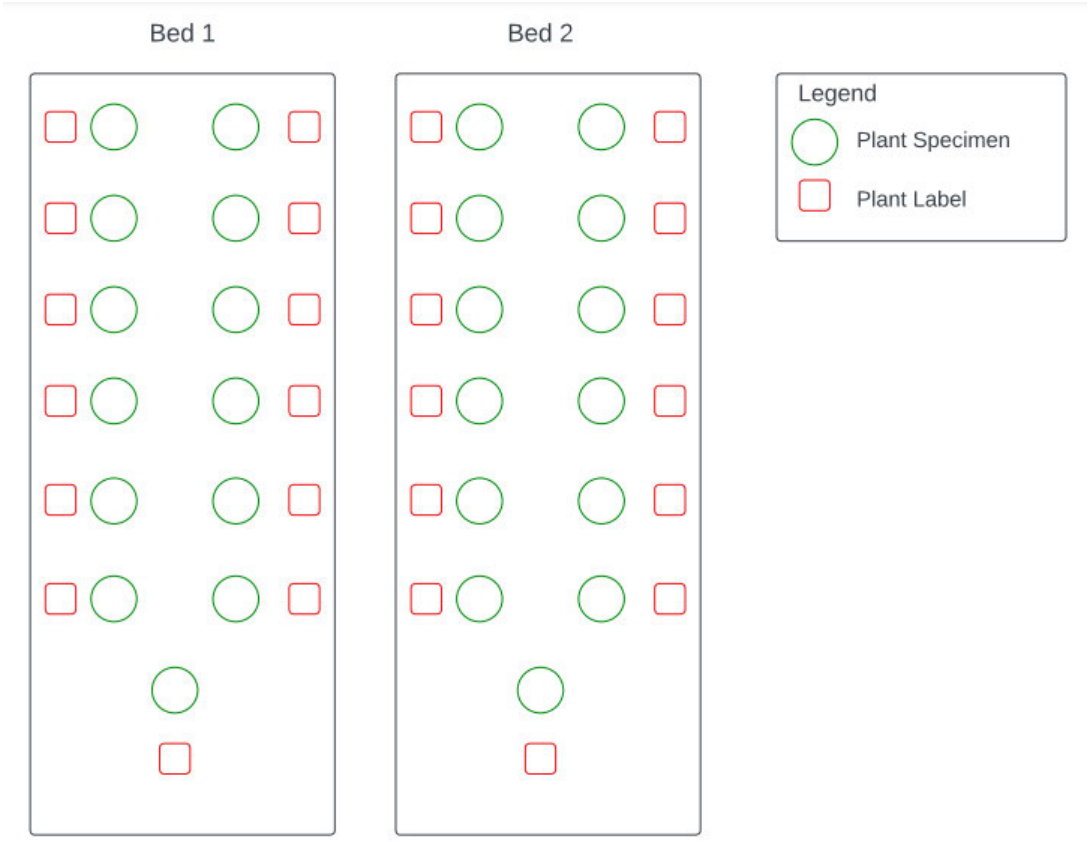

Figure 20: A diagram of the final test layout. Where circles represent plants and squares represent labels.

These groups of 6 were grouped following the same methodology as the simulated test, with 30 plants manually swapped with the original 6 to simulate plant growth. By manually swapping and the capturing plant specimens, a physical demonstration of the adaptive control plugins abilities to calculate and adapt irrigation amounts to plant area was able to be demonstrated.

On top of each of these plant beds, a Viper spectra grow light was placed in the centre of each bed on a medium intensity level to provide adequate lighting to each of the plant specimens. Once this was complete, the FarmBot API was used to position the camera above each specimen to capture an image (Figure 23). Images of the final testing environment and a captured image example are shown in Figures 21 and 22.

Figure 21: Final testing environment demonstrating grow light positioning.

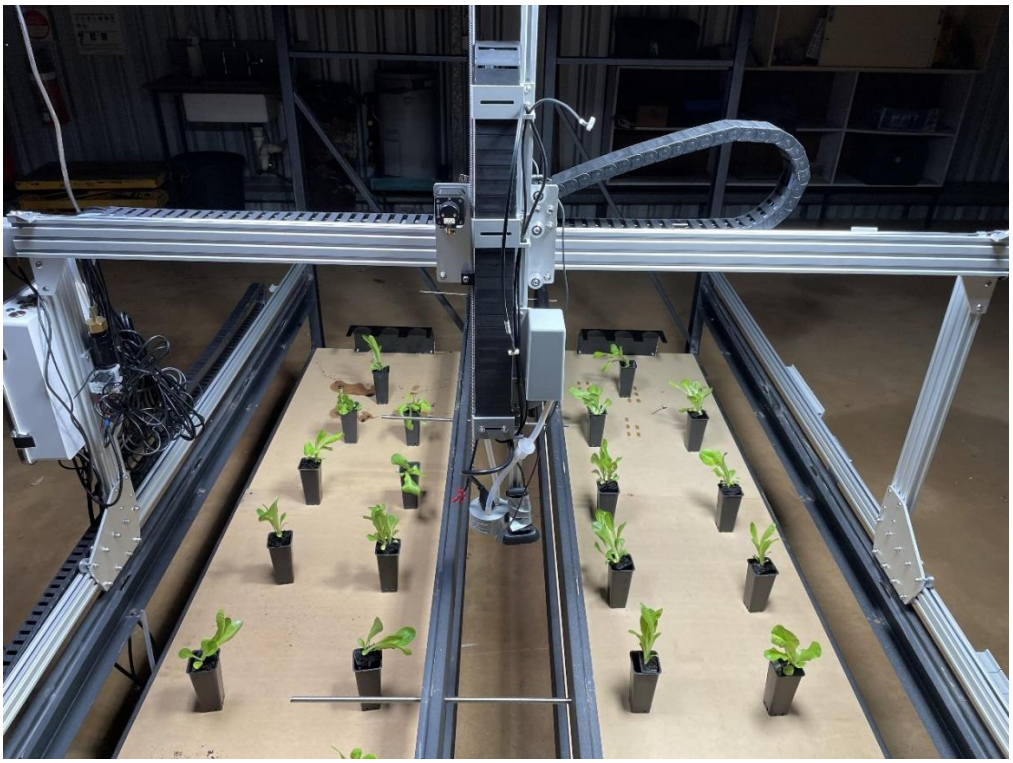

Figure 22: Final testing environment following diagram arrangement.

Figure 23: Example of a plant specimen captured using the FarmBot and camera.

5.2 Final Testing Results and Recommendations

Once the physical data had been collected, each specimen's growth data was collated and sorted into their corresponding iteration folders. However, when applying the collected data into the adaptive control plugin results were unable to be generated. This failure to compile the characteristics of each image derives from a design choice made when first creating the adaptive control plugin.

When analysing each plant specimen, the corresponding specimen number used for organisation and storage is displayed using a dice pattern. While utilising this methodology under simulated and ideal conditions yields usable results. When applied to a realistic environment, noise seen around the extremities of the captured image produce false positives of the dice pattern, resulting in Figure 24.

```
Water values are [25 75 71 68 50]
Traceback (most recent call last):
  File "c:\Users\Corey\Desktop\Plugin_Test_Folder\Plugin.py", line 143, in <module>
    Calculated_Image_Array = np.array([Number_Detected, Combined_Area, Water_Values_Array[Number_Detected-1]])
IndexError: index 27 is out of bounds for axis 0 with size 5
PS C:\Users\Corey\Desktop\Plugin_Test_Folder>
```

Figure 24: Resulting error after analysis of physical data.

Due to the overloading of the number detection portion of the program, the system iteration count was utilised to cycle through each growth stage of the plant specimens. This allowed the system to display the contoured images in Figure 25 and their resulting binary images in Figure 26.

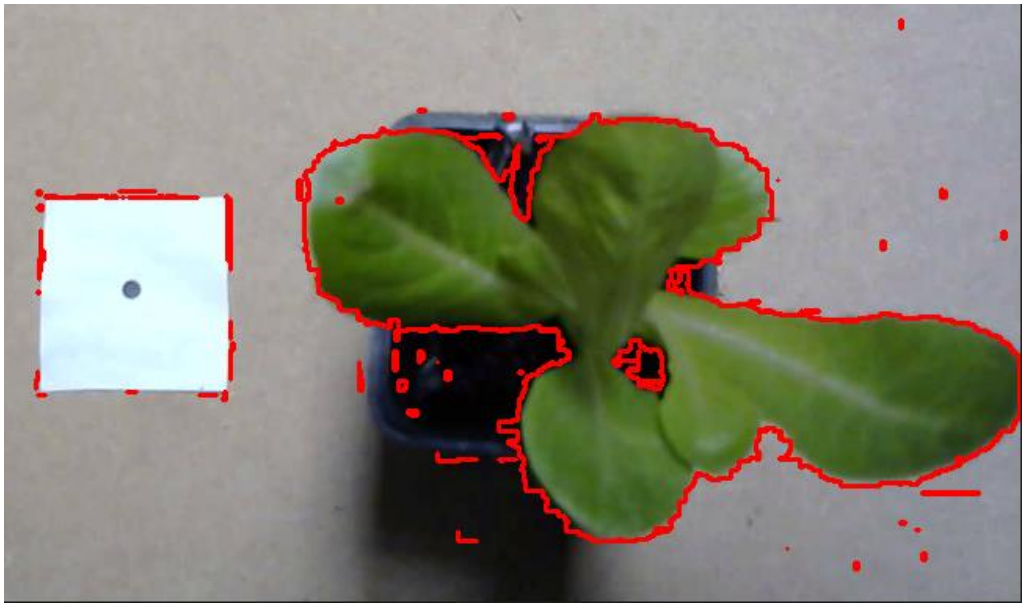

Figure 25: A produced output of the systems area analysis

Figure 26: False positives shown in the resulting detection count for specimen labelling.

It was found that due to the broad values specified in the programs HSV colour thresholds, noise can be seen when calculating the area for the plant in addition to its corresponding label identification.

However, because the data within the images was able to be processed, the images were implemented into the adaptive control plugin to produce the following output (Figure 27):

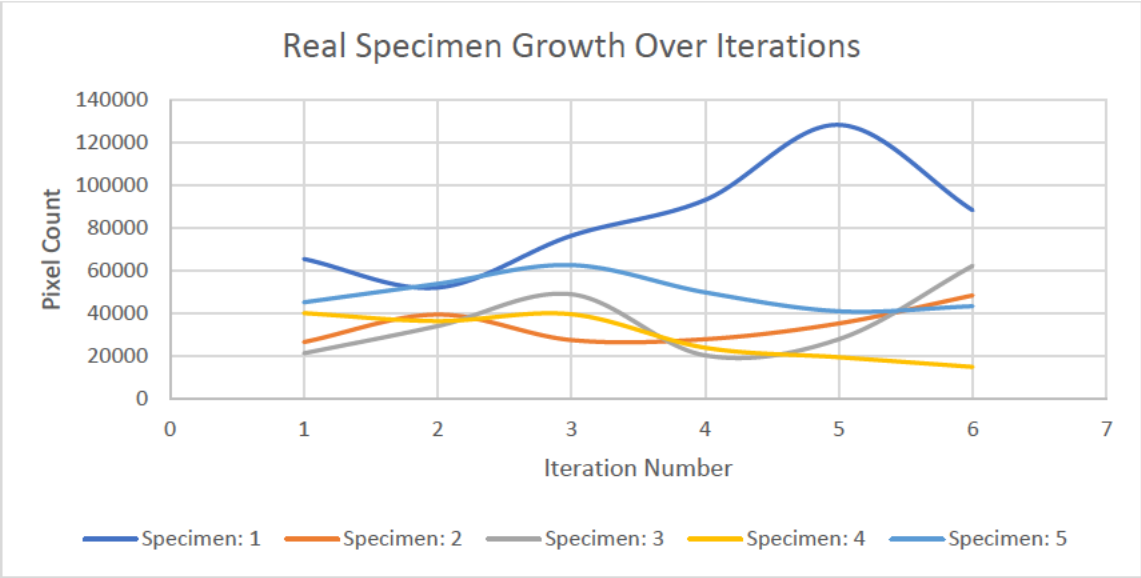

Figure 27: Adaptive control output of analysed growth over iterations.

After implementing the image data into the adaptive control plugin, specimens 2 through 5 demonstrated a consistent growth pattern. However, specimen 1 shows a steep increase in growth from iterations 2 to 5. Following the received area data for each plant specimen, the adaptive control plugin was able to produce the following (Figure 28):

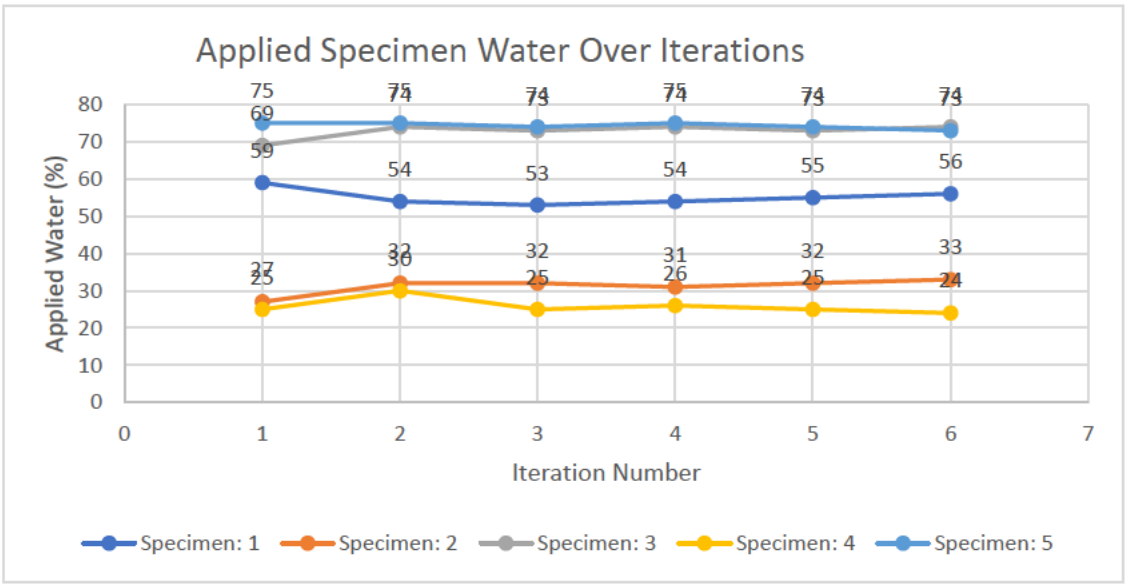

Figure 28: Adaptive control output of applied water over iterations.

After analysing the adaptive control output, the system was able to function as intended, producing results that were free of any corruption observed in simulated testing. The initial randomly assigned water percentage values were able to achieve a coverage of the 25% to 75% while remaining within the initial threshold.

However, it was only through the exclusion of the number detection algorithm that these results were able to be attained, leading to recommendations that can be made on the systems functionality.

5.2.1 Recommendation 1: Number Identification Through Deep Learning.

The first recommendation that can be made to overcome this mis-identification issue is the implementation of a trained model to identify numbers rather than count dots. In this scenario, when creating specimen labels, a font would be chosen with numbers 0 through to 9 printed on a white background.

These images would then be selected and displayed to the trainer in a random order, with the user inputting the corresponding image to the system. Once the system could correctly identify the desired number, the process would be repeated under realistic conditions to refine its identification capabilities.

This method would be able to bypass the problems faced in the current version of the adaptive control plugin by increasing the complexity of the shape the system needs to identify.

5.2.2 Recommendation 2: Application of Different Colour.

The second recommendation that can be made on the identification method used within the adaptive control plugin, is the application of different colours used in the current dice pattern arrangement. With this iteration of the specimen identification method, noise experienced through the extremities of the camera and debris of soil would be avoided by the program due to the change of colour used in the specimen labels.

The colour chosen to replace the currently used black would need to be bold enough to be perceived in high saturation environments while also differing enough from the surrounding colours to avoid false positives from noise. A suggested colour for this application would be red, due to the contrast it has with the green specimens and black environment.

Chapter 6: Discussion and Further Work

6.1 Current State of Gantry Topology and Adaptive Control Plugin

At its current state, the FarmBot gantry topology and the accompanying adaptive control plugin can analyse plant specimens and apply water according to the current state of the plant specimen. However, there are a variety of improvements and adjustments that need to be made to both the physical gantry topology and the algorithms of the adaptive control plugin.

The FarmBot gantry topology still proves to be a viable system for maintaining and monitoring plant specimens. However, due to the current state of some components in the FarmBot topology, there is further work to be done in repairing or replacing components of the FarmBot to allow it to function as intended. If these physical limitations can be overcome, the viability of applying the same tests in this project on a larger scale is certain.

The adaptive control plugin can analyse and produce data of the plant specimen currently being examined; although there are multiple improvements that could be made to the system to allow greater functionality and accuracy. Through simulated and physical testing, one of the critical flaws the adaptive control plugin has is its inability to perceive and interpret depth.

When designing the adaptive control plugin, it was assumed that lettuce specimens would grow with their leaves fanning out facing upwards towards the camera; however, in practice that is not the case. During the growth cycle of a lettuce specimen, leaves initially grow upwards before becoming large enough to move to the sides of the plant. As the angle of these initial leaves isn't completely perpendicular to the camera, calculations of area are induced with minimal error. Through implementation of a stereo vision camera, the adaptive control plugin would benefit greatly from the ability to decern what is a leaf and what is the stem of the plant.

Another aspect of the adaptive control plugin made apparent during testing was the systems lack of awareness of other specimens. Regardless of the starting point, each plant specimen will undergo the same process to determine the optimal water value for a plant specimen. It is hypothesised that through the implementation of a form of communication between plant specimens, the process used to find the optimal water value could be further refined.

In this scenario, if a neighbouring plant specimen was on the optimal growth path based on the amount of water applied to the specimen, the system would adjust its current trajectory to match the highest performing plant specimen.

6.2 Further Physical Work

6.2.1 Redesign of the FarmBot Irrigation Tool

The first area of potential work that could be done on the FarmBot's physical design is the redesign of the current irrigation tool that the FarmBot uses. This would include the design of an irrigation tool that injects into the soil of the plant specimen, rather than the traditional spray method the FarmBot employs.

6.3 Further Digital Work

Before utilising a compatible plugin for the FarmBot API was considered, development commenced with initial research of the chosen FarmBot gantry topologies GitHub repository. This including its main end-user release of the API and the more obscure testing and development branch of the program.

Analysis of these branches began by first evaluating the core components of the FarmBot program to further investigate and identify pivotal subroutines and functions that could possibly hinder the integration of the system into a localised environment. This analysis provided valuable insight into the methods that the FarmBot API utilises in developing the routines and functions constantly used in average system operation. Specifically, it was identified that the previously anticipated subroutines and functions within the code are instead built upon the initial setup up parameters of the physical FarmBot gantry topology, relying on the provided dimensions to build custom routines that fit the physical parameters of the gantry topology.

Following the programs progression of development there were a variety of key files and documents that provided a clearer scope on the pivotal areas of the FarmBot API's code. This analysis revealed that there are three main components to the FarmBot's Web Application: The Database, The Server, and the GUI.

6.3.1 The Database

The first component of the FarmBot API is the database that is constructed upon launch of the API to house and manage the systems relevant data. The database is responsible for the maintenance and cataloguing of the users custom created routines, the users garden bed mapping, and characteristics as well as information and care of the individual plant specimens within the users' garden. This data is managed by PostgreSQL, an open-source database management system that utilises the SQL language to safely store and manage complicated data workloads (PostgreSQL,2022). The system can manage a variety of data types and formats, which is why PostgreSQL is seen throughout the entirety of the FarmBot's directory.

6.3.2 The Server

The second component of the FarmBot API is the server that houses the database and GUI assets used continuously during system operation. For the FarmBot API to maintain live access and communication with the physical topology the server utilises Ruby on Rails, a server-side web application framework responsible for launching and maintaining the HTML website server connection the physical FarmBot topology receives information from. Ruby on Rails can reliably interface with PostgreSQL due to both applications having the ability to interface with a variety of different data types and formats, this way the server can actively retrieve information from the system database without having to compromise the systems data integrity.

6.3.3 The GUI

Finally, the main component of the FarmBot API is the systems GUI or graphical user interface. The GUI in Figure 29 relies heavily on the performance of both the Ruby on Rails server performance and the data management of the PostgreSQL database application, to actively pull and display different areas of the FarmBot API from the user's garden bed to the numerical data and analytics acquired from the FarmBot's operation. The GUI acts as the main area of user interaction and the basis of further customisation of the system.

6.3.4 Integration of Adaptive Control as an Application

Through analysis of each of the FarmBot API's core components, it is possible to make the FarmBot and the adaptive control plugin in its current state more intuitive and easier to visualise by integrating it as an application into the FarmBot API.

This would allow the system to utilise physical resources more effectively by utilising the data management of the pre-existing Raspberry Pi used by the FarmBot topology. In addition, this would also make accessing and altering the adaptive control plugin to be more intuitive and less tedious.

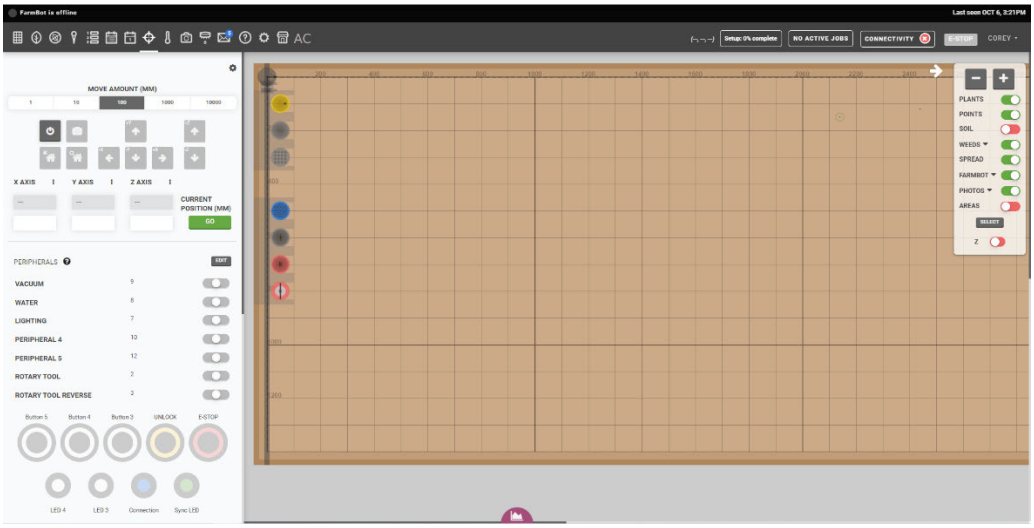

Figure 29: Concept start-up of the FarmBot API with added adaptive control application tab.

Ideally the implementation of the adaptive control plugin would be situated along the top of the page, similarly to the pre-existing applications already used in the FarmBot API. After selecting the adaptive control application, the parameters of the plugin's functionality could be changed along the left had side of the page. In the centre of the page, graphs showing each plant specimens current growth and their applied water levels would be displayed. It is envisioned that the final adaptive control GUI, would take the following layout shown in Figure 30.

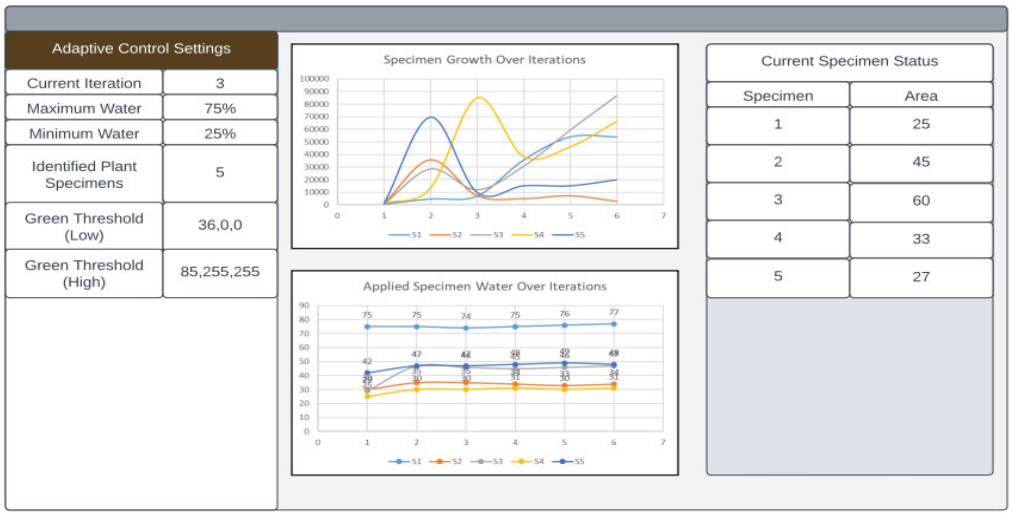

Figure 30: Proposed Layout of the Adaptive Control Plugin.

Chapter 7: Summary and Conclusion

Through the completion of this project there are a variety of conclusions that can be made on the results achieved by the adaptive control plugin and how they reflect upon the projects specifications. These specifications, in addition to the actions to achieve them are summarised in this chapter.

1. Conduct literature review on precision agriculture topologies, microgravity agriculture, current gantry agriculture applications and manipulators.

During the initial stages of the project's development, a literature review was conducted on current precision agriculture topologies, their applications, and technologies. This provided a valuable scope of the state of current automated agricultural systems and their technologies. Furthermore, this literature review greatly aided in identifying potential pre-existing topologies within the market.

2. Identify core components of a remote automated microgravity greenhouse based on existing gantry topologies, including sensors, algorithms, manipulators, and server communications.

Through evaluation of these pre-existing topologies, core components that facilitate automated agricultural functionality were identified. These core components were identified over an array of commonly seen automated functionality components, from physical hardware to software functionality and communications methods.

3. Identify a suitable topology for a microgravity greenhouse with gantry robot which can operate locally with reduced reliance on external communications.

Through the identification of these core components, the FarmBot Genesis XL was identified as the most suitable robotic topology for projects application. This FarmBot was chosen due to its ability to operate within a local area using only one external communication dependency to the internet. This set the FarmBot apart from other topologies, that primarily used cloud computing and IoT systems with a higher reliance on external communications.

5. Construct a functional prototype of the desired gantry topology and assess physical capabilities of the prototype for plant monitoring actions.

After identifying the FarmBot as a suitable topology for the application, the robotic prototype was able to be constructed and initialised. This initialisation allowed the physical capabilities of the FarmBot to be assessed, providing valuable insight into the limitations of the FarmBot's ability to monitor plant specimens.

4. Develop a localised, stable system software and hardware framework and integrate with an existing gantry robot.

Using the FarmBot's electrical hardware in conjunction with a provided internet modem, a stable and reliable connection was able to be established between the FarmBot's hardware and the internet modem. This stability within the hardware components, allowed the created adaptive control plugin to receive image data from the FarmBot for use in its system software.

6. Test candidate methodologies for monitoring of plants over a minimum of one week to evaluate topology performance.

Before applying the adaptive control plugin to living specimens, its methodologies were tested using data obtained from a 21-day timelapse of a butterleaf specimens' growth. This testing demonstrated key flaws seen within final adaptive control plugin topology and helped formulate recommendations and further work for continued development of the plugin's topology.

7. Gather and evaluate experimental data on topology components and report on functionality within physical, software and system level performance.

Following the simulated testing trials, living butterleaf and cos lettuce specimens were arranged as per the designed layout. This layout allowed the FarmBot topology gather experimental data to be used in the adaptive control plugin. The gathered experimental data was evaluated, highlighting significant design flaws within the algorithms and performance of the adaptive control plugin.

Furthermore, through application of the FarmBot's monitoring capabilities to living specimens, flaws within the physical topology components were able to be identified. These flaws were reported upon with potential substitutions evaluated and subsequently implemented into the final design.

8. Refine data collection and processing algorithms to increase validity and reliability of experimental data as required.

Over the course of this projects design, the adaptive control plugin topology went through multiple iterations of development. These changes revolved around refining data collection and reliability through image thresholds and changes in how the system can process multiple images. These iterations in development allowed the resulting adaptive control plugin to successfully process plant growth images and output them to the system serial window.

9. Further develop and implement graphical user interface to enhance functionality for future research of plant monitoring in a microgravity greenhouse with gantry robot.

Finally, concepts for the FarmBot graphical user interface were developed for use in further implementation of the adaptive control topology to merge with the pre-existing FarmBot API. These GUI concepts were evaluated with arguments raised on the benefits of further development into this area in the system topology.

References

1. Keeter, B, 2020, *Long-Term Challenges to Human Space Exploration*, NASA, P viewed 15/09/2021, <https://www.nasa.gov/centers/hq/library/find/bibliographies/Long-Term_Challenges_to_Human_Space_Exploration>.
2. Tan, L, 2016, *Cloud-based Decision Support and Automation for Precision Agriculture in Orchards*, Issue 16, Elsevier, Science Direct, viewed 20/09/2021, <https://www.sciencedirect.com/science/article/pii/S240589631631624X>.
3. Baohua, Z, Xie, Y, Wang, K, Zhen, Zhang, 2020, *State-of-the-art robotic grippers, grasping and control strategies, as well as their applications in agricultural robots: A review*, Volume 177, Elsevier, Science Direct, viewed 21/09/2021, < https://www.sciencedirect.com/science/article/pii/S0168169920311030>.
4. Shafi, U, Rafia, M, García-Nieto, J, Ali Hassan, S, Ali Raza Zaidi, S, Iqbal, N, 2019, *Precision Agriculture Techniques and Practices: From Considerations to Applications*, 1st Edition, MDPI, MDPI, viewed 20/09/2021, < https://www.mdpi.com/1424-8220/19/17/3796>.
5. Tian, C, Zhang, D, 2020, *A new family of generalized parallel manipulators with configurable moving platforms*, Volume 153, Elsevier, Science Direct, 21/09/2021, <https://www.sciencedirect.com/science/article/pii/S0094114X20302184>.
6. Ngo, Q, Lacey, R, Chunajiu, H, 2003, *Growing Plants for NASA — Challenges in Lunar and Martian Agriculture©*, IPPS, Departments of Horticultural Sciences and Biological & Agricultural Engineering, Texas A&M University, College Station, Texas 77843-2133, viewed 02/10/2021, <https://aggie-horticulture.tamu.edu/Faculty/davies/research/abstracts/pdfs/IPPS-2003-53-NASA.pdf>.
7. Wheeler, R, 2018, *NASA Activities in Controlled Environment Agriculture*, NASA, Kennedy Space Centre, Florida, viewed 02/10/2021, <https://ntrs.nasa.gov/api/citations/20180007212/downloads/20180007212.pdf>.
8. Yamashita, M, Hashimoto, H, Wada, H, 2009, *On-Site Resources Availability for Space Agriculture on Mars*, 1st Edition, Springer, Springer, viewed 03/10/2021, <https://link.springer.com/chapter/10.1007/978-3-642-03629-3_18>.
9. Hava, H, Zhou, H, Mehlenbeck, C, King, A, Lombardi, E, Baker, K, Kaufman, A, Nikolaus, C, 2020, *SIRONA: Sustainable Integration of Regenerative Outer-space Nature and Agriculture. Part 2 — Design Development and Projected Performance*, 1st Edition, Elsevier, Science Direct, viewed 03/10/2021, <https://www.sciencedirect.com/science/article/abs/pii/S0094576520304185>.
10. Sharath, G, Hiremath, N, Manjunatha, G, 2021, *Design and analysis of gantry robot for pick and place mechanism with Arduino Mega 2560 microcontroller and processed using pythons*, Volume 45, Materialstoday:PROCEEDINGS, Science Direct, viewed 06/10/2021, <https://www.sciencedirect.com/science/article/pii/S2214785320396413>.
11. Takara, G, Trimble, Z, Brown, S, Gonzalez, H, Mora, C, Arata, R, 2021, *An inexpensive robotic gantry to screen and control soil moisture for plant experiments*, Volume 9, Elsevier, Science Direct, viewed 10/10/2021, <https://www.sciencedirect.com/science/article/pii/S2468067221000031>.
12. History.com, 2020, The Space Race, A&E Television Networks, New York, viewed 09/10/2021, <https://www.history.com/topics/cold-war/space-race>.

13. Zhao, Y, 2016, *Dual-arm Robot Design and Testing for Harvesting Tomato in Greenhouse,* Issue 16, Elsevier, Science Direct, viewed 02/10/2021, <https://www.sciencedirect.com/science/article/pii/S2405896316315932 >.
14. Zych, A, 2021, *Programming of Welding Robots in Shipbuilding*, Volume 99, Elsevier, Science Direct, viewed 02/10/2021, <https://www.sciencedirect.com/science/article/pii/S2212827121004091 >.
15. Golovin, I, 2020, *Modeling and discrepancy based control of underactuated large gantry cranes*, Issue 2, Elsevier, Science Direct, viewed 02/10/2021, <https://www.sciencedirect.com/science/article/pii/S2405896320324800 >.
16. Wrest Park History Contributors, 2009, *Chapter 8 Horticultural engineering*, Edition, Elsevier, Science Direct, viewed 02/10/2021, <https://www.sciencedirect.com/science/article/pii/S1537511008003474 >.
17. Arian, A , 2017, *Kinematic and dynamic analysis of the Gantry-Tau, a 3-DoF translational parallel manipulator*, Volume 51, Elsevier, Science Direct, viewed 02/10/2021, <https://www.sciencedirect.com/science/article/pii/S0307904X17304043>.
18. Maroli, A, 2021, *Applications of IoT for achieving sustainability in agricultural sector: A comprehensive review*, Volume 298, Elsevier, Science Direct, viewed 02/10/2021, <https://www.sciencedirect.com/science/article/pii/S0301479721015504 >.
19. Maggi, F, Pallud, C, 2010, *Space agriculture in micro- and hypo-gravity: A comparative study of soil hydraulics and biogeochemistry in a cropping unit on Earth, Mars, the Moon and the space station*, Volume 58, Pages 1996-2007, Elsevier, Science Direct, viewed 06/01/2022, < https://www.sciencedirect.com/science/article/abs/pii/S0032063310003077>.
20. Maggi, F, Pallud, C, 2010, *Martian base agriculture: The effect of low gravity on water flow, nutrient cycles, and microbial biomass dynamics*, Volume 46, Pages 1257-1265, Elsevier, Science Direct, viewed 06/01/2022, < https://www.sciencedirect.com/science/article/abs/pii/S0273117710004849>.
21. Shusaku, N, Masayasu, N,Akifumi, I, 2021, *Mechanism for enhancing the growth of mung bean seedlings under simulated microgravity*, Volume 7, ProQuest, ProQuest, viewed 07/01/2022, <https://www.proquest.com/docview/2551800362?fromopenview=true&pqorigsite=gscholar>.
22. Oluwafemi, F. A., & Olubiyi, R. A. (2019). *Investigation of Corn Seeds Growth under Simulated Microgravity*. ARID ZONE JOURNAL OF ENGINEERING, TECHNOLOGY AND ENVIRONMENT, 15(SPi2), 110-115. Retrieved from https://azojete.com.ng/index.php/azojete/article/view/17
23. Rubyonrails.org, 2022. Ruby on Rails. [online] Ruby on Rails. Available at: <https://rubyonrails.org/> [Accessed 15 May 2022].
24. PostgreSQL.org, 2022. PostgreSQL: About. [online] Postgresql.org. Available at: <https://www.postgresql.org/about/> [Accessed 15 May 2022].
25. RS Components, 2022. DLE-RG-0001 | Igus Linear Robot 3 Axis, 500 x 500 x 100mm | RS Components. [online] Au.rs-online.com. Available at: <https://au.rs-online.com/web/p/gantry-robots/1268963> [Accessed 16 May 2022].
26. Sage Automation, 2022. Gantry Robots - Sage Automation Inc.. [online] Sage Automation Inc. Available at: <https://www.sagerobot.com/gantry-robots/> [Accessed 16 May 2022].
27. RS Components, 2022. Gantry Robots | 2 & 3 Axis Gantry Robots | RS Components. [online] Au.rs-online.com. Available at: <https://au.rs-online.com/web/c/automation-control-gear/industrial-robots/gantry-robots/> [Accessed 15 May 2022].

28. Arduino, 2022. Libraries - Arduino Reference. [online] Arduino.cc. Available at: <https://www.arduino.cc/reference/en/libraries/> [Accessed 9 Jun. 2022].
29. Raspberry Pi Ltd, 2022. Raspberry Pi Documentation - Raspberry Pi OS. [online] Raspberrypi.com. Available at: <https://www.raspberrypi.com/documentation/computers/os.html> [Accessed 9 Jun. 2022].
30. Nagura, 2019, 'Water movement on the convex surfaces of porous media under microgravity, Advances in Space Research, vol. 63., no. 1., pp. 589-597, viewed 09/06/2022, < https://www.sciencedirect.com/science/article/abs/pii/S0273117718307518>.
31. Dixon, 2015, 'Water, irrigation and plant diseases', CAB Reviews Perspectives in Agriculture Veterinary Science Nutrition and Natural Resources, vol. 10., no. 9., viewed 09/06/2022, < https://www.researchgate.net/publication/276472454_Water_irrigation_and_plant_diseases>.
32. SimpliLearn, 2022, *C++ Vs Python: Overview, Uses & Key Differences*, SimpliLearn, viewed 14/08/2022, < https://www.simplilearn.com/tutorials/cpp-tutorial/cpp-vs-python#:~:text=C%2B%2B%20is%20faster%20than%20Python,a%20faster%20compilation%20of%20code.&text=Python%20is%20slower%20than%20C,the%20process%20of%20compilation%20slower.>.
33. PytorchCv, 2022, *pytorchcv 0.0.67: Computer vision models on PyTorch*, Pypi.org, viewed 15/08/2022, <https://pypi.org/project/pytorchcv/>.
34. OpenCV, 2022, *About*, opencv.org, viewed 15/08/2022, <https://opencv.org/about/ >.
35. Bloch, V, Degani, A, Bechar, A, 2018, *A methodology of orchard architecture design for an optimal harvesting robot*, 1st Edition, Elsevier, ResearchGate, viewed 06/09/2022, <https://www.researchgate.net/publication/322864949_A_methodology_of_orchard_architecture_design_for_an_optimal_harvesting_robot>.
36. Levine, H., 2010. *The Influence of Microgravity on Plants*. Available at: https://www.nasa.gov/pdf/478076main_Day1_P03c_Levine_Plants.pdf

Appendix

Appendix A: Project Specification

ENG4111/4112 Research Project

Project Specification

For: Corey Ahlers U1119246

Title: Localised system topology for robotic monitoring of plant specimens for deep space missions

Major: Electrical and Electronic

Supervisors: Dr Jacob Humpal, Dr Cheryl McCarthy

Enrollment: ENG4111 – EXT S1, 2022

ENG4112 – EXT S2, 2022

Project Aim: This project aims to develop and evaluate a localised system topology for a gantry robot in a microgravity greenhouse, for the purpose of reducing remote communications for monitoring plant specimens on space missions.

Programme: Version 3, 16th March 2022

1. Conduct literature review on precision agriculture topologies, microgravity agriculture, current gantry agriculture applications and manipulators.
2. Identify core components of a remote automated microgravity greenhouse based on existing gantry topologies, including sensors, algorithms, manipulators, and server communications.
3. Identify a suitable topology for a microgravity greenhouse with gantry robot which can operate locally with reduced reliance on external communications.
4. Develop a localised, stable system software and hardware framework and integrate with an existing gantry robot.
5. Construct a functional prototype of the desired gantry topology and assess physical capabilities of the prototype for plant monitoring actions.
6. Test candidate methodologies for monitoring of plants over a minimum of one week to evaluate topology performance.
7. Gather and evaluate experimental data on topology components and report on functionality within physical, software and system level performance.

If time and resources permit:

8. Refine data collection and processing algorithms to increase validity and reliability of experimental data as required.

9. Further develop and implement graphical user interface to enhance functionality for future research of plant monitoring in a microgravity greenhouse with gantry robot.

Appendix B: Provided Server Initialisation Code

NUMBER	RISK DESCRIPTION	TREND	CURRENT	RESIDUAL
1274	Machine Vision Robotic Crop Production - 3 axis gantry robot - P1, P6		Low	Low

RISK OWNER	RISK IDENTIFIED ON	LAST REVIEWED ON	NEXT SCHEDULED REVIEW
Jacob Humpal	06/06/2022	06/06/2022	06/06/2023

RISK FACTOR(S)	EXISTING CONTROL(S)	PROPOSED CONTROL(S)	OWNER	DUE DATE
This work involves low voltage < 24v electricity and batteries. Hazards could be caused by exposed parts, faulty equipment or incorrect installation.	**Control:** The equipment has fuses in place and is rated as waterproof. It will be tested and tagged as per USQ requirement after assembly.	**No Control:**		
The 3-axis gantry system moves and contains potential pinch points. Assembly of robot will require use of hand drill and tools.	**Control:** Gloves will be worn when working around pinch points and appropriate PPE will be worn when using hand tools. **Control:** Training will be provided in use of tools prior to assembly.	**No Control:**		
Chemical fertiliser may be applied.	**Control:** PPE will be worn (glasses, gloves) when handling any chemicals. **Control:** Chemicals will be stored, handled and disposed of appropriately as per USQ regulations. Chemical safety training will be undertaken to use chemicals.	**No Control:**		
Plants will be grown and harvested. These plants are limited to non-GMO, commercially available food species.	**Control:** No controls required as these are commercial, safe food plants. Plants will be disposed of appropriately as per USQ regulations.	**No Control:**		
Due to the nature of the work, water will be present and may present a slip hazard. The student will often be working alone.	**Control:** Any spills will be promptly cleaned up. **Control:** When the student is working alone they shall check in through security or through the use of the USQ safe app.	**No Control:**		

Appendix C: Provided Server Initialisation Code

```
# IMPORTANT NOTE: Resources are limited and Farmbot, inc. cannot provide

# longterm support to self-hosted users. If you have never administered a

# Ruby on Rails application, we highly advise stopping now. this presents an
```

```
# extremely high risk of data loss. Free hosting is provided at
# https://my.farm.bot and eliminates the risks and troubles of self-hosting.
#
# You are highly encouraged to use the my.farm.bot servers. Self hosted
# documentation is provided with the assumption that you have experience with
# Ruby/Javascript development.
#
# Self-hosting a Farmbot server is not a simple task.


# Remove old (possibly broke) docker versions
sudo apt remove docker-engine
sudo apt remove docker docker.io containerd runc


# Install docker
sudo apt update
sudo apt install ca-certificates curl gnupg lsb-release -y
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg
echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt update
sudo apt install docker-ce docker-ce-cli containerd.io docker-compose-plugin -y
sudo docker run hello-world # Should run!
# Install docker-compose
sudo mkdir -p /usr/local/lib/docker/cli-plugins
```

```
sudo curl -SL "https://github.com/docker/compose/releases/download/v2.4.1/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/lib/docker/cli-plugins/docker-compose

sudo chmod +x /usr/local/lib/docker/cli-plugins/docker-compose

sudo docker compose version # test installation


# Install FarmBot Web App

# ⚠ SKIP THIS STEP IF UPGRADING!

git clone https://github.com/FarmBot/Farmbot-Web-App --depth=5 --branch=main


cd Farmbot-Web-App


cp example.env .env # ⚠ SKIP THIS STEP IF UPGRADING!


# == This is a very important step!!! ==

#

# Open `.env` in a text editor and change all the values.

#

# == Nothing will work if you skip this step!!! ==


nano .env          # ⚠ SKIP THIS STEP IF UPGRADING!

# ^ This is the most important step

# READ NOTE ABOVE. Very important!


# Install the correct version of bundler for the project

sudo docker compose run web gem install bundler

# Install application specific Ruby dependencies
```

```
sudo docker compose run web bundle install

# Install application specific Javascript deps

sudo docker compose run web npm install

# Create a database in PostgreSQL

sudo docker compose run web bundle exec rails db:create db:migrate

# Generate a set of *.pem files for data encryption

sudo docker compose run web rake keys:generate # ⚠ SKIP THIS STEP IF UPGRADING!

# Build the UI assets via ParcelJS

sudo docker compose run web rake assets:precompile

# Run the server! ٩(^‿^)۶

# NOTE: DONT TRY TO LOGIN until you see a message similar to this:

#   "✨  Built in 44.92s"

# THIS MAY TAKE A VERY LONG TIME ON SLOW MACHINES (~3 minutes on DigitalOcean)

# You will just get an empty screen otherwise.

# This only happens during initialization

sudo docker compose up


# At this point, setup is complete. Content should be visible at ===============

#  http://YOUR_HOST:3000/.


# You can optionally verify installation by running unit tests.


# Create the database for the app to use:

sudo docker compose run -e RAILS_ENV=test web bundle exec rails db:setup

# Run the tests in the "test" RAILS_ENV:

sudo docker compose run -e RAILS_ENV=test web rspec spec
```

```
# Run user-interface unit tests REQUIRES AT LEAST 4 GB OF RAM:
sudo docker compose run web npm run test


# === BEGIN OPTIONAL UPGRADES
  # To update to later versions of FarmBot,
  # shut down the server, create a database backup
  # and run commands below.
  git pull https://github.com/FarmBot/Farmbot-Web-App.git main
  sudo docker compose build
  sudo docker compose run web bundle install   # <== ⚠ UPGRADE USERS ONLY
  sudo docker compose run web npm install      # <== ⚠ UPGRADE USERS ONLY
  sudo docker compose run web rails db:migrate # <== ⚠ UPGRADE USERS ONLY
# === END OPTIONAL UPGRADES ^
```

Appendix D: Example Environment File

```
# You will hit issues if any of these are set to the wrong value.
# Please read each line of this file before starting the server.
#
# When you are done, save this file as `.env` at the root of the
Farmbot-Web-App
# directory.
#
#  Again, PLEASE READ ALL ENTRIES. This is the most important setup
step.
#
================================================================
===========
#
```

```
# Where is your MQTT server running? 99% of setups will use the same
value

# found in API_HOST. Heroku users will not use the same value.

# Use a REAL, PUBLIC IP ADDRESS if you are controlling real bots.

MQTT_HOST=98.76.54.32

# Set the max pool size for Passenger.

# Only needed if using Heroku. FarmBot, Inc. uses Heroku. Self
hosters do not.

MAX_POOL_SIZE=2

# If your server is on a domain (eg=my-own-farmbot.com), put it
here.

# DONT USE `localhost`.

# DONT USE `127.0.0.1`.

# DONT USE `0.0.0.0`.

# Use a real ip or domain name.

API_HOST=12.34.56.78

# 3000 for local development. 443 is using SSL. You will need `sudo`
to use PORT

# 80 on most systems.

API_PORT=3000

# Every server needs to set this. This is the password to the entire
database.

# NOTE: Must be less than 100 characters long.

POSTGRES_PASSWORD=

# MUST REPLACE. MUST BE A VERY RANDOM VALUE.

# 128 CHARACTERS LONG, HEXADECIMAL STRING (0-9, A-F)

DEVISE_SECRET=Used for devise. Generate a new value using `openssl
rand -hex 64`.

# Every server has a superuser.
```

```
# Set this to something SECURE.

ADMIN_PASSWORD=

# Secret key used by Rails.

# Generate a new value using `openssl rand -hex 64`

SECRET_KEY_BASE=

# Set this to production in most cases.

# Setting this line to production will disable debug backtraces.

# Please delete this line if you are submitting a bug report, as
production mode

# will not give detailed crash reports.

RAILS_ENV=production

# Set this if you don't want to deal with email verification of new
users.

# (self hosted users)

NO_EMAILS=TRUE

# If you wish to opt out of https:// (we wish you wouldn't), you
can

# delete this line. Be aware that by not using SSL, users will
transmit their

# passwords without encryption, making it very easy for attackers
to see

# user passwords. Consider buying a domain and using a free
certificate from

# Let's Encrypt.

FORCE_SSL=Remove this if not using HTTPS://


#  MOST USERS SHOULD DELETE THE REST OF THIS FILE.

#  Continue reading if you:

#   * work at FarmBot, Inc.
```

```
#   * need email notification support

#   * pay for managed database / file hosting (Google Cloud)

#   * use the test suite to write new features

#   * run your own NervesHub instance for custom FBOS updates

# If running a FarmBot setup for personal use or none of the above apply, you

# can safely delete the rest of this file.



# Only relevant if you use Heroku or pay a 3rd party vendor for Redis hosting.

# Most users can delete this.

# If your Heroku Redis vendor uses a custom `REDIS_URL` ENV var such as

# `REDISTOGO_URL`, set the value here. If you  delete this line, the app will

# default to `REDIS_URL`.

WHERE_IS_REDIS_URL=REDISTOGO_URL  # Just an example. Change or delete.

# Delete this if you  don't use 3rd party Redis hosting. See WHERE_IS_REDIS_URL

REDIS_URL=redis://redis:6379/0



# For email delivery. Who is your email host?

SMTP_HOST=smtp.sendgrid.net

# Optional with default of 587

SMTP_PORT=587

# FarmBot, Inc. uses SendGrid to send emails.

# Delete these if you aren't a send grid customer.

SENDGRID_PASSWORD=Used by FarmBot, Inc
```

```
SENDGRID_USERNAME=Used by FarmBot, Inc

# If you're using other SMTP server (like Gmail) use this.

#SMTP_USERNAME=email@gmail.com

#SMTP_PASSWORD=password


# Used by people who pay for managed database hosting.

# Most users should delete this.

DATABASE_URL=postgres://user:password@host:5432/db_name

# Google Cloud Storage API Bucket for image data.

# Deleting this will save to disk.

# Most self hosting users will want to delete this.

GCS_BUCKET=GOOGLE_CLOUD_STORAGE_BUCKET_NAME_FOR_IMAGE_FILES

# Google Cloud Storage ID for image data.

# Deleting this will save images to disk.

# Most self hosting users will want to delete this.

GCS_ID=GOOGLE_CLOUD_STORAGE='interop' id

# Most self hosting users will want to delete this.

GCS_KEY=GOOGLE_CLOUD_STORAGE='interop' key

# Can be deleted unless you are a Rollbar customer.

ROLLBAR_ACCESS_TOKEN=____

ROLLBAR_CLIENT_TOKEN=____

# This can be set to anything.

# Most users can just delete it.

# This is used for people writing modifications to the software,
mostly.

DOCS=Set this to any value if you want to generate API docs after
running tests
```

```
# Most self hosting users will want to delete this.

HEROKU_SLUG_COMMIT=This is set by Heroku, used by Frontend to show
current version.

# If you are a software developer and you wish to run integration
tests, set the

# ENV below to true.

# Most users will not want this enabled.

RUN_CAPYBARA=true

# Self hosting users can delete this line.

# If you are not using the standard MQTT broker (eg=you use a 3rd
party

# MQTT vendor), you will need to change this line.

MQTT_WS=ws://DELETE_OR_CHANGE_THIS_LINE/ws

# If you are using a shared RabbitMQ server and need to use a VHost
other than

# /, change this ENV var.

MQTT_VHOST=/

# If you run a server with multiple domain names (HINT=You probably
don't),

# you can list the names here. This is used by FarmBot employees so
that they

# can securly host the same server on multiple domain names

#     ex=my.farm.bot, my.farmbot.io

EXTRA_DOMAINS=staging.farm.bot,whatever.farm.bot

# Some hosts (Eg=FarmBot, Inc.) run the RabbitMQ management API on
a

# non-standard host.

# Include the protocol! (http vs. https)

# DELETE THIS LINE if you are a self-hosted user.
```

```
RABBIT_MGMT_URL=http://delete_this_line.com

# Allow only certain users on the server. If the user's email domain
is not

# on the list of trusted domains, they can not use the server.

# The example below only allows users with `@farmbot.io` or
`@farm.bot` emails

# to use the server.

# DELETE THIS LINE IF YOU RUN A PUBLIC SERVER.

TRUSTED_DOMAINS=farmbot.io,farm.bot

# Self hosting users can safely delete this (a new key will be
created).

# This key is used to exchange secrets between bots and MQTT servers
(important

# if you don't use SSL)

# SERVER WONT WORK IF YOU FORGET TO DELETE THIS EXAMPLE TEXT BELOW.

# ADD A REAL RSA_KEY OR DELETE THIS LINE!!

RSA_KEY=Change this! Keys look like `-----BEGIN RSA .........`

# Prevents JS/CSS build system from cleaning out old assets on start.

# This speed up boot time by one minute, but may put you at risk of

# loading stale versions of the application.

NO_CLEAN=true

# FarmBot uses DataDog for log analytics and for assesing overall
system health.

# Do not add this key if you  do not use DataDog on your server.

DATADOG_CLIENT_TOKEN=??

# Comma separated list of emails that wish to receive a daily

# report of new FarmBot installations (not new users, but

# actual FarmBot installations).
```

```
CUSTOMER_SUPPORT_SUBSCRIBERS=alice@protonmail.com,bob@yahoo.com

# URL to send user-generated feedback to.

FEEDBACK_WEBHOOK_URL=http://localhost:3000/change_this

# Email address of a "publisher account" that is used to

# publish shared sequences via `rake sequences:publish <id>`

AUTHORIZED_PUBLISHER=foo@bar.com
```

Appendix E: Final Adaptive Control Plugin

```
# Initialisation of libraries and functions

import cv2 as cv

import numpy as np

import os

import math

# Initialise global variables


No_of_plant_specimens = 5 # Change this value for specimen count

Initial_State = 0; # Upon initialisation the state is 0


# NOTE: CHANGE THS CODE TO STATE BASED


################################################################################
######


# Initialisation of Values / Beginning of Program


# Phase 1: Create, draw-from and assign water by volume amounts to specimens
```

```
# NOTE: This is another database that should be agreed upon with Jacob and
Cheryl.

# However for now, the range will be from 25% to 75%


Potential_Initial_Water_Volumes = list(range(25,75))


def Water_Values_Selection():

    Selected_values = []

    for count in range(No_of_plant_specimens):

       Selected_value = np.random.choice(Potential_Initial_Water_Volumes)

       Selected_values.append(Selected_value)

    return Selected_values


# Assembling the Water Values array used in initialisation

Water_Values_Array = np.array(Water_Values_Selection())

# Finding Minimum and assigning it as a guaranteed 25

Minimum_Water_Value = np.min(Water_Values_Array)

Water_Values_Array[Water_Values_Array == Minimum_Water_Value] = 25

# Finding Maximum and assigning it as a guaranteed 75

Maximum_Water_Value = np.max(Water_Values_Array)

Water_Values_Array[Water_Values_Array == Maximum_Water_Value] = 75

print("Water values are", Water_Values_Array)


##########################################################################
######


# NEED TO SET UP A WHILE LOOP HERE TP CHANGE FOR LOOP CHARACTERISTICS
```

```
# Phase 1: Collect and identify plant image and area



rootdir = "C:\\Users\\Corey\\Desktop\\Plugin_Test_Folder\\Official_Test"



# Collect the image from designated folder (These will be Farmbot scans)

path                                                                    =
"C:\\Users\Corey\\Desktop\\Plugin_Test_Folder\\Official_Test\\Official_Test_1"

# NOTE: This is a secondary pathway used for simulating purposes

path_2 = "C:\\Users\Corey\\Desktop\\Plugin_Test_Folder\\Official_Test_2"

Current_Iteration = 0; # Set Base count responsible for building final results
array

Final_Assigned_Values = np.empty(shape=(No_of_plant_specimens,3))

Previous_Values   =   np.empty(shape=(No_of_plant_specimens,3))   #   Initialise
previous values array

Past_Iteration_Water  =  np.zeros(No_of_plant_specimens)  # Initialise  previous
values array

Past_Iteration_Area  =  np.zeros(No_of_plant_specimens)  #  Initialise  previous
values array

Past_Iteration_Water_HOLD   =   np.zeros(No_of_plant_specimens)   #   Initialise
previous values array

Past_Iteration_Area_HOLD   =   np.zeros(No_of_plant_specimens)   #   Initialise
previous values array

Iteration_Count = 0 # When the system begins the count is zero. This lets the
intial water values only be used once

Final_Array_Log =  np.empty(shape=(No_of_plant_specimens,3))



Array_Selection_Area  =  np.array([1,4,7,10,13]) # This  is  composed  for  the  5
specimens NOTE: Possibly automate the creation of this array

Array_Selection_Water = np.array([2,5,8,11,14])



# Iterate through images based on how many scans are present
```

```
for subdir, dirs, files in os.walk(rootdir):
# Second file pathway for LOOP here!



        for file in files:


                        # Select current working image based on file pathway
                        input_path = os.path.join(subdir, file)
                        image = cv.imread(input_path)


                        # Convert working image to HSV
                        hsv_image = cv.cvtColor(image, cv.COLOR_BGR2HSV)


                        # Defining thresholds for detection
                        low_green = np.array([36, 0, 0])
                        high_green = np.array([85, 255, 255])
                       # Creating a mask to store allowed colours in threshold
                     Green_mask = cv.inRange(hsv_image, low_green, high_green)
                        Green = cv.bitwise_and(image, image, mask=Green_mask)


                       # Now that we've detected the image isolate its contours
                        contours, _ = cv.findContours(Green_mask, cv.RETR_TREE,
cv.CHAIN_APPROX_NONE)
                        # Connect the countour points together
                        cv.drawContours(image, contours, -1, (0,0,255), 2) # -
1 draws all contours
```

```
                        # Calculate the area of the contour


                        def find_contour_areas(contours):
                            areas = []
                            for count in contours:
                                contour_area = cv.contourArea(count)
                                areas.append(contour_area)
                            return areas


                        # Output areas in pixels
                                        #print("Contour  areas  individual",
find_contour_areas(contours))
                        Combined_Area = sum(find_contour_areas(contours))
                        #print("Total combined contour area", Combined_Area)


                         # Phase 2: Draw from the same image bank and identify
dice dots and count


                        # Collect the image from the images folder (These will
be the Farmbot Dice Scans)
                        dice_image = cv.imread(input_path)
                        # Convert image to hsv colour standard
                                    hsv_dice_image  =  cv.cvtColor(dice_image,
cv.COLOR_BGR2HSV)
                        # Define thresholds for detection
                        # NOTE: This will need to be tweaked in practive based
on light saturation and angle
                        low_black = np.array([0, 0, 0])
                        high_black = np.array([0, 0, 75]) # for now using grey
                        #Create mask to identify values within range for mask
```

```
                            Black_mask = cv.inRange(hsv_dice_image, low_black,
high_black)

                               Black = cv.bitwise_and(dice_image, dice_image,
mask=Black_mask)



                          # Identify contours to count dice dots

                        DiceContours, _ = cv.findContours(Black_mask, cv.RETR_TREE,
cv.CHAIN_APPROX_NONE)

                          # Connect the contours

                         cv.drawContours(dice_image, DiceContours, -1, (0,0,255),
2)



                            # Now that we have detected the circles, we need to
count them

                             # To do this we just count how many contours were
detected

                          Number_Detected = len(DiceContours)

                      # print("This is the number:", Number_Detected)



                            # Phase 4: Take the Plant area, Number and assigned
water to put them into an intial array that the system will build on

                          # First Column: Plant Number

                          # Second Column: Plant Area

                          # Third Column: Assigned Water Value

                         # Need to save a copy of the past values from the second
iteration



                        if Initial_State == 0: # If the system is in intialisation

                               Calculated_Image_Array = np.array([Number_Detected,
Combined_Area, Water_Values_Array[Number_Detected-1]])
```

```
                            #Begin slicing calculated values into the array

                                Final_Assigned_Values[Current_Iteration, :] =
Calculated_Image_Array

                        else:

                            if Initial_State == 1: # If the system has passed
initialisation and is now adapting







                                    # From this point the system has passed
intialisation and can now move to adapt (Adaptive state)



                                    #NOTE: THIS COMPARISON IS DONE USING THE
CURRENT ITERATION VALUE



                                                        Calculated_Image_Array  =
np.array([Number_Detected, Combined_Area, Water_Values_Array[Number_Detected-
1]])



                                    # Iteration 1 | Scenario 1: Examined specimen
is below 50% and has reduced in size

                                    # Process: Identify plant specimen number,
Identify change in growth, recognise previous water

                                    # level value and INCREASE it by 5%



                                    # Call current selections

                                                            Selected_Area   =
Previous_Values[Current_Iteration,1]

                                                        Selected_Water   =
Previous_Values[Current_Iteration,2]
```

```
                                    Past_Iteration_Area[Current_Iteration] =
Previous_Values[Current_Iteration,1] # Guarenteed starting comparisson AREA

                                   Past_Iteration_Water[Current_Iteration] =
Previous_Values[Current_Iteration,2] # Guarenteed starting comparisson



                                if (Calculated_Image_Array[1] < Selected_Area):
# Theres been a visable reduction in size

                                    # Reduction in size and below 50%

                                    if Selected_Water <= 50:

                                        if ((Calculated_Image_Array[2] + 5)
>= 75):

                                                Calculated_Image_Array[2]
= Calculated_Image_Array[2]

                                         else:

                                                Calculated_Image_Array[2]
= Calculated_Image_Array[2] + 5

                                    else: # If water isnt below 50% it must
be above

                                        if ((Calculated_Image_Array[2] - 5)
<= 25):

                                                Calculated_Image_Array[2]
= Calculated_Image_Array[2]

                                         else:

                                                Calculated_Image_Array[2]
= Calculated_Image_Array[2] - 5



                                if (Calculated_Image_Array[1] > Selected_Area):
# Theres been a visable increase in size

                                    # Reduction in size and below 50%

                                    if Selected_Water <= 50:

                                        if ((Calculated_Image_Array[2] + 5)
>= 75):
```

```
                                                    Calculated_Image_Array[2]
= Calculated_Image_Array[2]

                                            else:

                                                    Calculated_Image_Array[2]
= Calculated_Image_Array[2] + 5

                                        else: # If water isnt below 50% it must
be above

                                            if ((Calculated_Image_Array[2] + 5)
>= 75):

                                                    Calculated_Image_Array[2]
= Calculated_Image_Array[2]

                                            else:

                                                    Calculated_Image_Array[2]
= Calculated_Image_Array[2] + 5



                                    Final_Assigned_Values[Current_Iteration, :] =
Calculated_Image_Array # Re-slice array to reflect corrections




                                                #Calculated_Image_Array  =
np.array([Number_Detected, Combined_Area, Water_Values_Array[Number_Detected-
1]])

                                else:

                                    if Initial_State == 2: # Moving into second
area and further control



                                        # Phase 2: Adaptive Control of delivered
water levels depandant on plant area



                                            # Step 1: Memorise previous specimen
table

                                        # Step 2: Correlate plant increase with
postive water result.
```

```
                                        # Step 3: Correlate plant decrease with
negative water result.

                                         # Step 4: Implement condition where a
plant can only be watered +/- 10% of its initial starting value.

                                        # Step 4: Justify increase or decrease:
Is the plant being over watered, or under watered?

                                         # Step 5: Based on justification, make
appropriate judgement. e.g: the plant is dying but is being watered in the top
10% range. Instead of maxing it, drop the water level.



                                    # Create values to select from past values

                                                          Selected_Area   =
Previous_Values[Current_Iteration,1]

                                                        Selected_Water   =
Previous_Values[Current_Iteration,2]

                                    Past_Iteration_Area_HOLD[Current_Iteration]
= Previous_Values[Current_Iteration,1] # Guarenteed starting comparisson AREA

                                   Past_Iteration_Water_HOLD[Current_Iteration]
= Previous_Values[Current_Iteration,2] # Guarenteed starting comparisson



                                                     Calculated_Image_Array  =
np.array([Number_Detected, Combined_Area, Selected_Water])



                                                          Previous_Area   =
Past_Iteration_Area[Current_Iteration]

                                                        Previous_Water   =
Past_Iteration_Water[Current_Iteration]



                                    Increased_5  = Previous_Water + 5

                                    Increase_1 = Previous_Water + 1

                                    #print("Past water is:", Previous_Water)

                                    # Iteration 2: This iteration is the major
turning point for each specimen. If there is no continued
```

```
                                        # visable growth it is reduced by 10%.
If the plant shows growth the scope is reduced



                                       # This iteration is checking: was 5% of
water increased or reduced, has it grown or shrunk



                                        # Applied 5% more, Increased

                                    if (Selected_Water == Increased_5): # Theres
been a visable increase in water

                                                # Increase in size and applied
1%

                                            if (Previous_Area < Selected_Area):

                                                #print("Passed Size Check")

                                                if ((Calculated_Image_Array[2]
+ 1) >= 75):

                                                     Calculated_Image_Array[2]
= Calculated_Image_Array[2]

                                                else:

                                                     Calculated_Image_Array[2]
= Calculated_Image_Array[2] + 1

                                                else: # Increase in water and
reduction in size

                                                if ((Calculated_Image_Array[2]
- 10) <= 25):

                                                     Calculated_Image_Array[2]
= Calculated_Image_Array[2] - 5

                                                else:

                                                     Calculated_Image_Array[2]
= Calculated_Image_Array[2] - 10



                                     if (Previous_Water == Increase_1): # Theres
been a visable increase in water

                                            #print("Passed Increase Check + 1")
```

```
                                                # Increase in size and applied
1%
                                            if ((Calculated_Image_Array[2] + 1)
>= 75):
                                                    Calculated_Image_Array[2] =
Calculated_Image_Array[2]
                                             else:
                                                    Calculated_Image_Array[2] =
Calculated_Image_Array[2] + 1
                                        else: # Increase in water and reduction in
size
                                            if ((Calculated_Image_Array[2] - 1)
<= 25):
                                                      Calculated_Image_Array[2]
= Calculated_Image_Array[2]
                                             else:
                                                      Calculated_Image_Array[2]
= Calculated_Image_Array[2] - 1


                                    if (Previous_Water == Previous_Water - 10):
# Theres been a visable increase in water
                                           #print("Passed Increase Check -
10")
                                             # Increase in size and applied
1%
                                            if (Previous_Area < Selected_Area):
                                                 if ((Calculated_Image_Array[2]
+ 1) >= 75):
                                                      Calculated_Image_Array[2]
= Calculated_Image_Array[2]
                                                 else:
                                                      Calculated_Image_Array[2]
= Calculated_Image_Array[2] + 1
```

```
                                    Final_Assigned_Values[Current_Iteration, :]
= Calculated_Image_Array # Re-slice array to reflect corrections



                                else:

                                    if (Initial_State > 2):

                                          # Create values to select from past
values

                                                                    Selected_Area  =
Previous_Values[Current_Iteration,1]

                                                                   Selected_Water  =
Previous_Values[Current_Iteration,2]



                                                               Calculated_Image_Array  =
np.array([Number_Detected, Combined_Area, Selected_Water]) # CURRENT ASSESSOR



                                                                    Previous_Area  =
Past_Iteration_Area_HOLD[Current_Iteration]

                                                                   Previous_Water  =
Past_Iteration_Water_HOLD[Current_Iteration]



                                       #print("Past water is:", Previous_Water)

                                          # Iteration 2: This iteration is the
major turning point for each specimen. If there is no continued

                                             # visable growth it is reduced by
10%. If the plant shows growth the scope is reduced



                                             # This iteration is checking: was
5% of water increased or reduced, has it grown or shrunk



                                           # Applied 5% more, Increased

                                        #print("Going for check")
```

```
                                    # print("Selected_Area:", Selected_Area,
"Previous Area:", Previous_Area)



                                     if (Previous_Area < Selected_Area): #
Theres been a visable increase in water

                                             #print("Passed Increase Check
+ 1")

                                                  # Increase in size and
applied 1%

                                               Calculated_Image_Array[2] =
Calculated_Image_Array[2] + 1

                                             Past_Iteration_Area_HOLD[Curre
nt_Iteration] = Previous_Values[Current_Iteration,1]

                                           else: # Increase in water and reduction
in size

                                               Calculated_Image_Array[2] =
Calculated_Image_Array[2] - 1

                                             Past_Iteration_Area_HOLD[Curre
nt_Iteration] = Previous_Values[Current_Iteration,1]



                                     if (Previous_Water == Previous_Water -
10): # Theres been a visable increase in water

                                             #print("Passed Increase Check
- 10")

                                                  # Increase in size and
applied 1%

                                        if (Previous_Area < Selected_Area):

                                                 Calculated_Image_Array[2]
= Calculated_Image_Array[2] + 1

                                               Past_Iteration_Area_HOLD[C
urrent_Iteration] = Previous_Values[Current_Iteration,1]



                                     if (Previous_Area == Selected_Area): #
Set simulation to end
```

```
                                                Initial_State == -1


                                    Final_Assigned_Values[Current_Iteration, :]
= Calculated_Image_Array # Re-slice array to reflect corrections




                        #print(contours)

                        cv.imshow("image",image)

                        cv.imshow("Green", Green_mask)

                        #cv.imshow("dice_image", dice_image)

                         #cv.imshow("Identified Dots", Black_mask)




                        # Increment Count

                          if Current_Iteration < No_of_plant_specimens - 1: #
Indicies start at 0

                            Current_Iteration = Current_Iteration + 1

                        else:

                                                        print("Iteration:",
Initial_State)                                          # This is the point
where all 5 plants are evaluated and the final array for the sample set is
generated

                            Initial_State = Initial_State + 1 # New State

                            Current_Iteration = 0 # Reset count so a new array
can begin

                            print(Final_Assigned_Values)



                              # Now that the array has been fully calculated,
write to Array log text file
```

```
                            a_file = open("Array_Log.txt", "w")
                            for row in Final_Assigned_Values:
                                np.savetxt(a_file, row)

                            a_file.close()

                               Previous_Values = Final_Assigned_Values # Save
previous values
                           # np.concatenate((Final_Array_Log, Previous_Values))
# Create Log of all arrays generated

                            if Initial_State == -1:
                                print("Simulation Complete")
                               break # End the simulation once a plant reaches
the end

                        # Debug Code to destroy all windows
                        cv.waitKey(0)
                        cv.destroyAllWindows()
```