The Role of Artificial Intelligence, Object Detection, and Environmental Control in Greenhouse Automation

**Gerardo Blanco Bernal**

**Student ID: 180000301**

**AC40001 Honours Project**

**BSc (Hons) Computing Science**

**University of Dundee, 2021/22**

**Supervisor: Dr. Jacky Visser**

***Abstract* –** *Greenhouses are key to providing controlled climates in which to grow crops year round. However, it is not an easy task to provide the sown seeds an optimal environment at all times throughout the day, and even more so while balancing a full-time job with this hobby. Not only are the crops susceptible to dramatic changes to the conditions inside the greenhouse, but they can quickly become malnourished if weeds appear in the greenhouse. In this study, the researcher aims to develop a solution to this problem by means of environmental control automation, object detection, and artificial intelligence to explore the feasibility of such a system. Through the control of high-voltage hardware using household electricity and general purpose sensors using a Raspberry Pi, the greenhouse’s internal conditions can be regulated effectively. By using a camera module and training a custom weed detection model, the system can recognize and alert the user of weeds in the greenhouse. The system operates an intelligent personal assistant which uses Telegram’s Bot API to communicate with the user. The researcher can conclude that the developed system successfully provides a straightforward way for a newly started gardener to balance their hobby with their day-to-day life.*

# Introduction

Greenhouses are structures built to protect plants from harsh climates, provide warmth throughout the year and to get a jump start on the growing season by planting early and later moving the plants out to the garden. Additionally, they help to save water in water scarce regions; the protective covering on a greenhouse entraps moisture, reducing evapotranspiration rates [[1](#_References)], and provides the opportunity for drip irrigation, closer crop spacing, and abbreviated crop cycles.

To maximize the obtained yield of crops, the greenhouse must maintain a controlled microclimate that protects plants from the variability of open-air conditions. Only then will greenhouse crops be produced more predictably, in greater quantity, and with less water than crops grown in the open air. So, while there is a consensus that greenhouses greatly contribute towards a more optimal growing environment, these conditions are not always given, and generally, not always easily achieved.

For instance, the beneficial effects of incident solar radiation which passes through transparent roof and walls can quickly become detrimental to the health of the crops if not controlled within an adequate threshold. As the structure is not naturally open to the atmosphere, the warmed air cannot escape via convection, so the temperature inside the greenhouse rises and is then absorbed by the contents of the greenhouse. The resulting heat can quickly produce temperature extremes under which the plants will suffer severe stress. Extended exposure to high temperatures can lead to denaturation or aggregation of proteins [[2](#_References)], ultimately causing a gradual death of the cells and tissues of plants, and in some cases, death of the whole organism.

This is only one of the many factors the greenhouse gardener must learn how to control in order to ensure ideal conditions necessary for success, in many cases, for all stages of a plant’s life cycle.

It therefore comes as no surprise that gardening in a greenhouse is a considerable time investment. Yet, having a greenhouse is on the wish list of many backyard horticulturists,and it can be incredibly rewarding once you become proficient at it, even being a beneficial supplementary in reducing the severity of stress and depression for veterans and students struggling with their mental health [[3](#_References)].

In this study, the researcher aims to alleviate this particular problem, that is, providing a means of reducing the necessary human intervention in the extensive environmental control required for adequate plant growth, by means of a cost-effective solution to greenhouse automation without needing to resort to a fully equipped industrial scale greenhouse. This solution is obviously intended to aid the stakeholders of this problem – the newly started gardeners who might struggle balancing their full-time day job and their gardening hobby adequately.

Once having evaluated the degree of involvement required from the end user to control each of the necessary components for plant growth in a greenhouse, the tasks which should be automated could be determined, and from these, tailor the resulting prototype design.

Subsequently, the researcher set out the objective of creating a physical computing-based solution whereby an intelligent system capable of perceiving and responding to its surroundings would provide a means to regulate the internal conditions of the greenhouse, perform accurate weed prevention and detection, and communicate with the end user via an intuitive mobile user interface.

The self-irrigating system would be the heart of the prototype; harnessing mains electricity and using a measure of sensors as well as actuators, the degree of internal temperature and humidity, ventilation, soil moisture, and irrigation tank levels would be controlled automatedly while abstracting the user from the process.

With help of an external camera module, an object detection model would be trained, evaluated, and deployed on the system to serve as the main pre-emptive measure towards weed detection.

The end user, by-way-of their mobile phone, would use a text-based assistant who not only would notify the user in case of possible weeds within the crops, but also would allow for a mixture of natural language and hard-coded commands as input.

# Background

As many as 55% of all adult citizens of the U.S.A spent time outdoors gardening, and another 20% were reportedly contemplating doing the same during the early months of the COVID-19 pandemic [[4](#_References)]. With around 20 million first-time adopters of the hobby in 2020 alone [[5](#_References)], growing food and other ornamental plants at home was turning into the new norm, with a variety of herbs, vegetables, and fruits at the top of many gardeners’ lists. Miracle-Gro determined that the main reasons the newly started greenkeepers decided to turn to this vocation was to have access to fresh food and reduce stress, with as close as 49% of the gardeners also wanting to experience a sense of accomplishment [[4](#_References)].

Though it is often seen as a barrierless pastime which anyone can take on regardless of physical limitations, access to land or previous experience, forums such as Reddit, and media outlets the like of Alabama NewsCentre were flooded with questions regarding everything from growing vegetables, to landscaping and even pest control. This reflected the existing skill gap which newcomers would have to overcome, and had already proven in the past to result in moderately high rates of abandonment of the hobby, mirrored by the endless posts in gardening oriented discussion websites [[6](#_References)].

Consequently, problems often arise when technicality is accepted without a full understanding, and even more so when there is a lack of knowledge of the available variety of crops that can be grown, and an ignorance towards the identification and control of weeds/pests.

The solution the researcher aimed to develop intended to tackle the various problems encountered during the initial stages of greenhouse gardening in addition to the year-round plant care. It was designed to lower the technical barrier between the involved parties, as some might not be familiar with the specific components necessary for the growth, health, and good condition of the plants. With this additional assistance, the goal of producing crops which would succeed in providing produce at the end of their season could more easily be attained. Thus, serving as a means of auxiliary support should the crops be temporarily neglected or mishandled, easing the struggle that comes with such steep learning curve.

In order to fully grasp the scope of the prototype that was to be produced, further research was conducted on the relevant existing literature in an effort to determine: Had a solution to this problem already been attempted? How had this same issue been approached by other scholars? What were their findings? Were their discoveries applicable to this specific domain, and if so, what relevance do they bear?

Many similar solutions were found, but none without their respective drawbacks. Though some did deal with environmental control in a greenhouse, they were tailored towards other purposes, such as the long-term storage of fruit by taking the physiological status of the fruit into consideration for example [[7](#_References)], so they were not as applicable to the objective of the researcher.

The minority of studies that did attempt to provide a solution to the issue at hand would seemingly always overlook one of the few facets a prototype for this problem should ideally have. The platform developed by the Department of Agricultural Machinery Engineering at the University of Tehran [[8](#_References)] would efficiently deal with the automation of an irrigation system by controlling the duration, frequency and quantity of water used in a session. However, research proves that there are more elements that play a critical role in a plant’s health other than just soil moisture [[9](#_References)], so this solution still leaves a large room for human error and failure.

Nonetheless, one commercially successful solution was found that was to all intents and purposes complete. MultiGrow by AutoGrow [[10](#_References)] is a multiple grow area controller fit to monitor and alter the relative humidity, soil moisture, temperature, and carbon dioxide levels amongst many other additional features in a greenhouse. MultiGrow would come equipped with remote management and access tools, a modular design which would encourage scalability, and a plethora of sensor controlling capabilities. By all appearances, this product already achieved what the researcher had set out to accomplish. However, upon further inspection, it seemed to no longer be the case.

AutoGrow claims to provide systems for single compartment environments, and that the “backyard gardening enthusiast has vastly different needs to the industrial growers”, so they ensure that the products they provide “match the need of the grower at all times.” [[11](#_References)]. With a starting price of £19,999.00 [[12](#_References)], it is immediately obvious that their target customers are those who own very large greenhouses, if not industrial scale ones. The product will not appeal to a newly started gardener who is using a greenhouse as a hobby, not a business.

After analysing their product, the researcher concluded that it is easy to overshoot the requirements for optimal plant growth past a degree of substantial noticeability to a point at which the differences in performance plateaus.

For instance, take the ability to control carbon dioxide concentrations using MultiGrow. The benefits of carbon dioxide enrichment in greenhouse cultivation to enhance plant growth have been largely documented. However, enrichment only becomes effective where, by Liebig's law, carbon dioxide has become the limiting factor, as it states that growth is dictated not by the total resources available, but by the scarcest resource (limiting factor) [[13](#_References)]. Enrichment of carbon dioxide would only be necessary in sealed grow rooms with no air exchange with the outside environment [[14](#_References)], which would not be the case for the average gardening enthusiast’s greenhouse nor for the prototype the researcher was developing as it had ventilation windows.

Artificial illumination is also another unnecessary expense as only certain crops will benefit from a prolonged exposure to lighting, or when grown out of season. The prototype did’t aim to stray away from the roots of horticulture. It is designed to help the end user reap the crops they wish to sow, but will only display the full potential of its abilities when the selected plants are within the realms of the appropriate species for the current season.

In short, despite MultiGrow seemingly being the major contender in this field, its target audience is vastly different to those which the researcher aims to help, and surprisingly, even with the range of features the most equipped products on the market can provide, none seem to employ a means of automated weed detection. This is ultimately the major distinguishing factor of the researcher’s prototype from the existing solutions, in conjunction with the aforementioned intended demographic.

Finally, considering the highly interactive nature of the solution, the researcher concluded that not only should user feedback during the development stage be a possibility, but it was a necessity. A reluctance to consider alternatives to one’s preferred line of thought would lead to a poorly developed prototype which would likely not fulfil the true user requirements or needs, and ultimately fail to consider the usability in the implementation of the system. User testing would be carried out on-site under the supervision of the researcher. The gathering of external recommendations and feedback would take place throughout all of the development, but primarily during the testing stage prior to the finalisation of the solution design.

# Specification

The prototype would need to be able to process data recorded by the utilized sensors and the user interface too. The data for the internal as well as external temperature of the greenhouse, the water depth in the irrigation tank, the soil moisture and whether the ventilation window had been opened or not would be collected. This would be achieved with use of waterproof DS18B20 digital temperature sensors, HC-SR04 ultrasonic depth sensors and moisture sensors using an LM393 voltage comparator boards respectively. The model should provide the end user a means of interpreting that data if requested, and make available the context of the information too. This means that if the user would like to know the average soil humidity levels for example, additional relevant information should be provided such as the last time the irrigation system was activated, and whether it will be activated again soon or not.

The researcher had to decide how this information was to be displayed, and how the gardener should interact with the system. Two viable alternatives would have been a dedicated mobile application or a website, however, after much consideration, the chosen platform for the user interface was to be Telegram, the mobile messaging app. This was primarily due to how easily customizable and flexible the bot ecosystem was, as the user interface was to be initially designed so it functioned as a simple Q&A bot with a set of commands it could understand.

Based on the inherent features of the solution to be delivered, the researcher decided to follow an agile approach using a modified SCRUM methodology workflow with Unit Testing practices. What this meant was, Daily Scrum Stand-Up meetings would not be employed, but the remaining aspects of SCRUM would be used to enforce a release cadence and ensure predictability of the software. The prototype would come together in a very modular manner where the relationship between a set of sensors and actuators would be determined and tested first before merging their functionality with the other features. Using a Unit Testing process would encourage simple designs for modules with associated usage procedures. The researcher must consider how to test the application for the desired functionality first, leading to a deeper and earlier understanding of the product requirements. This way, using a combination of both programming practices, a suitable work schedule could be planned, with fixed deadlines for each deliverable and specific tests to prove each ‘unit’ works accordingly.

Following this decision, the researcher proceeded to annotate the single persona involved into specific user stories. This involved splitting their wants and needs into separate, workable features for the researcher to develop.

The individual user stories were first brainstormed and organized on paper. They were later added as Issues on GitHub ([See Appendix A](#_Appendices)) to visualise them in a more accessible manner. This would also provide a quantifiable way of measuring the weekly and overall progress made. Using the risk and value ratio of each issue, they were then organized using a MoSCoW prioritization technique (Must, Should, Could and Won’t) to prioritize the tasks for each respective sprint. Finally, a T-Shirt sizing method was used to judge the size of each task at hand and how long it would take to implement. The researcher could then organize each user story into sprints that would be completed each week. Intermittent user testing and feedback would replace the usual client meetings that would normally conclude each sprint of development.

From this, a project development plan was made as a timeline for the product. This would indicate each phase of development so it could provide the researcher a better understanding of the estimated time to complete the sprints, the important milestones and deadlines to meet, in addition to the resources needed to accomplish each feature. ([See Appendix B](#_Appendices))

# Design

**4.1 Technologies and Languages**

Before the researcher could begin the project, the software development tools, technologies and languages had to be considered in order to ensure the delivery of a high-quality, intuitive, and usable product by the end of the given timeframe.

Firstly, and most importantly, the researcher had to choose which board best suited the needs of the project. The market for Internet of Things (IoT) boards for development and prototyping is nothing short of scarce at the time of writing; with a wide variety on offer, one can choose from microcontroller-based boards (MCUs), Single-Board Computers (SBCs) or even Field-Programmable Gate Arrays (FPGAs) to design a network of interacting devices. These are three different kinds of electronic platforms, each with their respective advantages and disadvantages.

The researcher had no prior experience with Hardware Description Languages (HDL) such as Verilog or VHDL used with FPGAs. HDLs are visually similar to software but perform a very different function. While software provides a set of instructions to execute on hardware, HDL tells the FPGA how to configure its internal circuitry. Although this wasn’t an immediate deterrent to the use of FPGAs, the familiarity of Linux distributions available with SBCs or the possibility of coding standard software in C with MCUs was undeniably more attractive. For this reason, the researcher discarded FPGAs as a possible board on which to develop the project.

Microcontrollers were perhaps the simpler of the two devices remaining. They frequently carry Atmel’s AVR microcontroller chips which make programming easy by including a bootloader that allows a computer and a microcontroller to communicate and install sketches (programs) directly. However, most microcontrollers tend to be single core processors, so true multithreading is impossible. This simplicity normally makes them fairly inexpensive, but their functionality is limited compared to that of SBCs, making them a sub-par choice for computationally intensive tasks or anything other than a single program intended to run repeatedly.

SBCs are usually complete with an operating system (normally a distribution of Linux), microprocessors, memory, input/output (I/O) and other features customary of functional computers. Single board computers are most commonly used in industrial situations where they are embedded within other devices to provide automation and interfacing. Because of the very high levels of integration, reduced component counts and reduced connector counts, SBCs are often smaller, lighter, more power efficient and more reliable than comparable multi-board computers. [[15](#_References)] They are also used in applications for process control, like complex robotic systems and processor-intensive applications, hence often considered an excellent alternative to microcontrollers in projects where their processing power falls short.

Running object detection models, an intelligent ChatBot, and greenhouse environmental control concurrently was too big of a task for an MCU to cope with. For this reason, the researcher decided to find an appropriate SBC on which to develop the solution.

When picking an SBC, a standout feature that had to be considered was the connectivity options; ultimately the developed prototype would be largely defined by its connectivity. In order to fulfil the user stories and requirements, the hardware needed to achieve the necessary features had to be determined in advance so that the connectivity options could be decided.

At the very minimum, the SBC needed to have Wi-Fi connectivity so that the board could communicate with the user’s mobile device using a wireless internet connection. Other necessary features were the likes of special protocols such as I2C for the use of OLED displays, a large number of General-Purpose Input/Output (GPIO) pins to accommodate for the needed accessories for the environmental control, and finally the possibility of using Hardware Attached on Top (HAT) modules for the harnessing of mains electricity to power external water pumps, solenoid valves, and DC motors. Although there were more hardware specifications that could help with the usability of the system, these few requirements were all the researcher needed to begin the development of the prototype.

The researcher found that the single most appealing product which comfortably met these requirements was the Raspberry Pi SBC, more specifically the Raspberry Pi 4 Model B, developed by the Raspberry Pi Foundation. The Raspberry Pi Foundation provides Raspberry Pi OS, a Debian-based Linux distribution, which is actively maintained. Raspberry Pi OS is known for its compatibility, reliability, and adaptability to most general purpose projects boasting an extensive ecosystem of software tools and packages that can be leveraged to create more complex projects.

Python would be used as the main programming language due to its huge collection of libraries and the vast amount of Raspberry Pi specific online resources from its highly driven community. Almost every piece of hardware for the Raspberry Pi comes with a Python library to support it. Although alternative languages were considered, support for C/C++ is sometimes unavailable, and support for other languages seemed to be non-existent. Python is neat, easy to understand and to implement, so it was the clear option to use.

A Raspberry Pi can be controlled remotely without using an external monitor, keyboard, and mouse by using a Virtual Network Computing (VNC) or Secure Shell (SSH) client from the main PC. This screenless installation is sometimes called a ‘Headless’ Raspberry Pi setup. By learning how to achieve this setup, the researcher could forego the extra peripherals and directly control the Raspberry Pi wirelessly from any other computer with no major disadvantage. PuTTY, the leading SSH client for Windows, was used for the initial set-up of the Raspberry Pi. RealVNC, a server and client application for the VNC protocol was used to provide remote access to the Raspberry Pi.

Thonny Integrated Development Environment (IDE) came preinstalled with Raspberry Pi OS. The Graphical User Interface (GUI) was very simple and clean, making it very beginner friendly, and Thonny also provided a simple debugger with good representation of function calls and the ability to check how shell commands affect the Python variables. However, the researcher quickly realised that it was not going to be the best environment in which to develop the prototype. The interface was visibly oriented to beginner programmers, with very limited functionality and a very slow creation of plugins. Although it would have probably sufficed for the scope of the project, the researcher had previously used another Python dedicated IDE in the past which provided everything Thonny could and more, making it a more appealing option. PyCharm (Community Version) IDE, created by JetBrains, yields an intelligent code editor with smart code navigation as well as outstanding code auto-completion and assistance. The debugger was notoriously powerful and the virtual environment management, which the researcher would need for the separation of the different software libraries, was incredibly straightforward. All of these features made PyCharm a stand-out IDE perfectly suited for this project.

**4.1.1 Greenhouse Environment Control**

As for the environment control design, the researcher had to initially find a suitable greenhouse on which to test, develop and install the features required for its automation. The structure of a greenhouse is made with a transparent material, normally one of glass, polycarbonate, or polyethylene. The aim was to deviate as little as possible from the most commonly purchased types of greenhouses, so that the prototype could be representative of what the end user could expect in terms of performance and durability of the design. This way, the prototype was not limited to a specific model and type of greenhouse, but instead could be coupled onto a pre-existing greenhouse to endow it with automated functionalities. Thusly, if the gardener already owned a greenhouse, they would not have to purchase the developed automated greenhouse as a whole, and could instead incorporate the embedded system individually onto theirs. Following this guideline and keeping the requirements in mind, the researcher decided to develop the prototype on a polycarbonate-based greenhouse with a fixed rigid structure and a ventilation window ([See Appendix O](#_Appendices)).

Polycarbonate is a tough, dimensionally stable, transparent thermoplastic. It maintains its properties over a wide range of temperatures, with the highest impact resistance of any thermoplastic, while having low water absorption, an internal transmission of light nearly in the same capacity as glass, and self-extinguishing properties [[16](#_References)]. These qualities made it ideal so as to not interfere with the internal relative humidity of the greenhouse and in the face of electrical fires.

A fixed structure would make it less susceptible to external conditions, this way preserving the integrity of the greenhouse and the prototype. Additionally, the need for a ventilation window was non-negotiable to the researcher as this was the most cost-effective way of regulating the internal temperatures of the greenhouse without resorting to horizontal air flow fans or HVAC systems [[17](#_References)].

The researcher initially intended to supply the greenhouse with two separate tanks, water, and liquid fertilizer respectively, for irrigation ([See Appendix C](#_Appendices)). By combining both fluids when watering the crops, the researcher could ensure that the crops were supplied with the necessary nutrients in the case of their absence in the soil. However, after weighing out the long-term feasibility and cost of a system like this, a change to solid state granulated fertilizer was made. This decision was in part due to the properties of liquid fertilizer, as they are more susceptible to volatilization, so their long-term storage is not easy to achieve. On the contrary, granulated fertilizer, not only are more cost effective, but can be easier to store as they do not ‘settle out’ [[18](#_References)]. Because of the slow-release characteristics of granulated fertilizers, the user would most likely only have to apply the dry fertilizer once at the beginning of the season, as opposed to having to refill the irrigation tank liquid supply at a much higher frequency. With user experience in mind, it was deemed that this task wouldn’t be particularly enjoyable to carry out multiple times every month, so the decision to use granulated fertilizer was finalised.

With one less tank to worry about, the researcher now had to design a control system for the water levels in the water drum. For this process, water would be drawn from a water outlet and allowed to flow into the drum by a 12V DC Normally Closed (NC) solenoid valve. When the solenoid valve is connected, the plunger would open, allowing water to flow into the cavity port (inlet) and out the body orifice port (outlet) into the tank. If the solenoid valve is disconnected, the orifices would close, and consequently the flow of water through the valve would stop.

From the water drum, the water would be drawn out using a small 3W pump, and sent through the irrigation system towards the crops. The main characteristic the researcher had to consider when choosing a water pump was its maximum flow, measured in litres per hour. Using drip irrigators, the flow rate for each plant could be regulated to be anywhere from 0-60 l/hour, so assuming that a minimum rate of one litre per hour was used per plant, 180 l/hour would be a more than sufficient flow rate for a standard sized greenhouse.

A resistive moisture sensor connected to a comparator board would be placed in the soil next to a row of crops to determine if the irrigation system should be activated or not. Moisture sensors can be capacitive or resistive in nature. Resistive sensors coupled with comparator boards provide a High/Low digital output that can be adjusted by using the potentiometer on the board to alter the threshold at which it registers a wet environment or not. On the other hand, capacitive moisture sensors would need an analogue-to-digital converter to make them compatible with a Raspberry Pi with the trade-off being an increased accuracy in the readings. Though they can admittedly be used interchangeably, capacitive sensors are better suited for long term use as resistive sensor probes are very prone to corrosion, not just because it is in contact with the soil but also because there is a flow of DC current which causes electrolysis of the sensors [[19](#_References)]. However, the availability of resistive sensors and their low price made them more convenient to use for this project, this way saving money on analogue-to-digital converters which were necessary for capacitive sensors. As the product to be developed was only intended to by a prototype for the solution needed, there was no reason why capacitive sensors couldn’t be used in further iterations of the product better equipped for a more long-term use.

Both the water pump and the NC solenoid valve were high voltage devices that could not be powered by the Raspberry Pi’s 3.3V and 5V GPIO pins, so in order to control them, a Relay HAT had to be used. Relay HATs are add-on boards designed to give a Raspberry Pi the ability to control high voltages/ high current devices. The relay board integrates three screw terminals each with three pins for connecting external circuits. All terminals are low active. When the Raspberry Pi outputs Low Level from its I/O, the relay NO (normally open contacts) close and the NC (normally closed contacts) open, so as to change the ON/OFF status of the external circuit [[20](#_References)].

Icon

Description automatically generated Icon

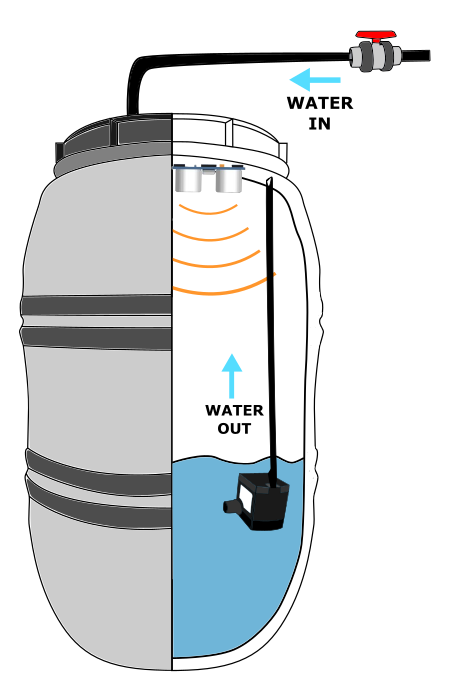
Description automatically generated

*Figure 1 on the left shows a HAT terminal before the relay action. Figure 2 on the right shows a HAT terminal after the relay action.*

The diagrams from Figure 1 and Figure 2 show how the relay allows for current to flow through the circuit, in this case turning a lightbulb on.

The water pump would be connected directly into mains electricity. However, in order to operate the 12V DC solenoid valve, a step-down AC to DC adapter was used.

Coupled with the solenoid valve, a HC-SR04 ultrasonic distance sensor was used to gauge the water levels in the drum and signal the solenoid valve to allow for water to fill it up or not. This was the most cost-efficient way of determining the water levels; with an accuracy of ±0.035cm and a range of up to 200 cm [[21](#_References)], there was no reason not to utilize this amazing piece of hardware. The distance sensor was attached to the inner side of the water drum lid so as to orientate the ultrasound pulse perpendicular to the water surface.



*Figure 3 above shows the placement of the depth sensor, water pump and solenoid valve.*

In Figure 3 above, the solenoid valve can be seen next to ‘Water In’, the water pump can be seen next to ‘Water Out’. The depth sensor can be seen pointed towards the surface of the water in the drum so that the ultrasound waves can reflect perpendicular to the surface of the water.

Two DS18B20 temperature sensors were used to measure both the external and internal temperature of the greenhouse. This data would then be used to open or close the ventilation window if the temperature wasn’t within the optimal range for plant growth. The DS18B20 has an accuracy of ±0.5°C in the range of -10°C to +85°C. As it is a sealed digital probe, it can withstand prolonged exposure to wet environments, making it ideal for the high humidity inside the greenhouse and the possible rain outside. The communication of this sensor was achieved through a one-wire bus protocol which uses one data line to communicate with an inner microprocessor. W1ThermSensor, a Python package and CLI tool specifically designed to work with temperature sensors that operate with this communication protocol, was used to obtain readings from the DS18B20 in a single line of code. The more traditional alternative would have been to use Python modules such as ‘os’ to use operating system dependent functionality to let the researcher read 1-wire sensors. However, opting for a 40-line [[22](#_References)] alternative seemed cumbersome for such a limited use that the sensors would provide, so the researcher avoided this method and used the neater approach provided by W1ThermSensor.

To allow for ventilation inside the greenhouse, a 12V DC motor was used to open and close the window when needed. The DC motor would be connected to the AC to DC adapter through the Relay HAT and a L298N dual H-Bridge motor driver (better seen in Section 5.1.1) which allows speed and direction control of the DC motor. This was necessary in order to open and close the ventilation window using the same DC motor so that it could wind and unwind the pull-string. It would operate in conjunction with a HC-SR04 sensor pointed at the window and naturally also be activated depending on the readings from the external and internal temperature sensors. The depth sensor would help in determining if the DC motor should remain operating or not depending on how open the window was.

Diagram

Description automatically generated with medium confidence

*Figure 4 above shows the design and placement of the depth sensor, pulley, and DC motor in the greenhouse.*

The greenhouse window was attached to the frame by two hinges which allowed for a relatively large degree of movement. The researcher would attach a pulley to the ridge beam of the greenhouse and loop a rope through it which would connect to the window on one end and the DC motor on the other. As mentioned above, a more detailed diagram of the DC motor and distance sensor wiring is provided in the implementation of these features ([Section 5.1.1](#_Implementation_and_Testing)).

For most GPIO sensor connections to the Raspberry Pi, the GPIO Zero Python library was used. Using this module, component interfaces were provided to allow a frictionless way to get started with physical computing. Although there exists an equally accredited library for GPIO pin control called RPi.GPIO, there were a few reasons why it wasn’t used in GPIO Zero’s place. RPi.GPIO is most useful for providing a beginner understanding of Broadcom (BCM) numbering for pins and basic workings of GPIO connection, which the researcher didn’t need, and although there is nothing inherently wrong with this library, GPIO Zero simplifies the syntax greatly making it easier to read and debug while keeping it as short as possible. This is because GPIO Zero is built on top of RPi.GPIO as a front-end language wrapper, simplifying GPIO setup and usage [[23](#_References)]. However, as the RPi.GPIO module is somewhat unsuitable for real-time or timing critical applications because you cannot predict when Python will be busy garbage collecting [[24](#_References)], the GPIO Zero module is too. Thankfully, no features of the prototype worked to such a high degree of timing accuracy where this would be a noticeable issue, so this was never a problem for the researcher.

Additionally, for all wiring and schematics that the researcher designed and would then include in the report, Fritzing and Inkscape were used. Fritzing is an open-sourced software for electronics hardware design; it has an overwhelming support for Raspberry Pi sensors, actuators, and all hardware in between, making it an ideal choice for simple and intuitive prototype circuitry design. Inkscape on the other hand is a vector graphics editor; the researcher primarily used Inkscape to draw and model the non-electronic hardware components of the prototype such as the greenhouse structure as can be seen in Figure 4, and the water drum as can be seen in Figure 3.

Finally, in order to model the relationship between the different features and hardware present in the prototype, the researcher designed a supplementary ER-Diagram ([Appendix K](#_Appendices)). This helped identify different system elements and their relationships with each other, further helping with determining requirements for the project.

**4.1.2 Weed Detection**

The earlier components of the prototype were originally designed on the Debian ‘Bullseye’ version of the Raspberry Pi OS. However, the researcher quickly ran into some issues.

In order to operate a camera module with the Raspberry Pi running Bullseye, the PiCamera Pyhton library had to be re-enabled. This was done by enabling legacy camera support under the Raspberry Pi configuration options. To the researcher’s surprise, once rebooted the system so the changes could take effect, RealVNC displayed an error message which stated that it could not ‘currently show the desktop’. Although many resolution configurations and display options were tried as per the general consensus on Raspberry Pi forums, there was no way to rid the VNC viewer of this error besides disabling the legacy camera support which the researcher needed. The only alternative was to flash the system with a ‘Legacy’ Raspberry Pi OS that tackled these problems and incompatibility issues of the general release operating system. The Legacy release based on Buster, the previous release of Debian, allowed for the reinstatement of legacy camera interfaces, and once installed, the VNC viewer would finally work as expected and the camera module could now be used.

The weed detection aspect of the prototype was comprised of a single piece of external hardware, the camera module, and Python libraries for object detection model training, testing, and deployment. One of the main Python libraries the researcher would have to learn to achieve this functionality was TensorFlow.

TensorFlow is an end-to-end open-source platform for machine learning, that is, the practice of helping software perform a task without explicit programming or rules [[25](#_References)]. The researcher would provide a set of examples and the Raspberry Pi would learn to identify patterns from the data. It uses Python to provide a convenient front-end API for building applications with the framework, while executing those applications as high-performance C++ binaries [[26](#_References)], which perfectly suited the researcher’s preestablished programming environment and language requirements. The single most important benefit that TensorFlow provided was the multiple layers of abstraction that made the overall logic of the application the main focus for the researcher.

However, most applications utilizing machine learning have models that require GPUs to conduct inference. TensorFlow Lite is the solution to enabling on-device machine learning which helps run models on mobile, embedded and edge devices such as the Raspberry Pi. Using this tool, the researcher could run the object detection models on the Raspberry Pi with relatively low latency without using an external API or server.

Although other machine learning frameworks exist that could have also been an appropriate tool for the prototype, such as PyTorch, they consumed too many resources in comparison to TensorFlow Lite, and had various limitations when it came to training models. Most importantly, the online resources and workflow support was nothing short of perfect for TensorFlow Lite, so it was a wise decision to use the framework that could be troubleshooted with more ease further down the line if there were ever any problems. It is important to note that at the time of writing, PyTorch Mobile, a mobile optimized version of PyTorch, is in its early stages and experimental release status, which didn’t make it the best choice considering the other available frameworks.

For the training of the object detection model, the researcher opted to use TensorFlow Lite Model Maker. The Model Maker library uses transfer learning to simplify the process of training a TensorFlow Lite model using a custom dataset [[27](#_References)]. This is achieved by retraining a TensorFlow Lite model with the researcher’s own custom dataset, which reduced the amount of training data required and hence would shorten the training time. The idea behind transfer learning for object detection is that a pre-existing model trained on a large and extensive dataset will serve as a generic model of the visual world. The researcher could then take advantage of these learned feature maps without having to start from scratch by training a large model on a large dataset [[28](#_References)].

The EfficientDet-Lite family of object detection models derived from the EfficientDet architecture would be used to train the model. The researcher aimed to test and utilize the different architectures interchangeably when determining the most efficient model during the implementation stage of this feature of the prototype.

The second tool used was OpenCV, a library of programming functions mainly aimed at real-time computer vision. It was used to capture and process the image input from the camera module which would then be given to the object detection model to make its predictions on.

Lastly, LabelImg, a graphical annotation tool written in Python, was used to manually annotate the different weeds which the researcher wanted the model to detect. The exported files were XML in PASCAL VOC format. These files together with the JPG images of the weeds were to be uploaded to TensorFlow Lite Model Maker as the training data.

The camera module would be pointed perpendicular to the ground so as to oversee the tops of the plants with a bird’s eye view. The only design limitation was the height at which it could be placed because if it was too far away from the weeds, it was more likely that the quality of the image would not suffice, and the model would fail to detect them. This trade-off was further assessed during the implementation of the feature.

**4.1.3 Assistance Bot**

The assistant ChatBot was designed to provide the gardener with any and all current information about the greenhouse conditions. The decision to use a ChatBot as opposed to a standard software application for interfacing with the system was in part rooted in the ability to provide proactive customer interaction. Instead of relying on passive user interaction to use a mobile application and stay engaged in the hobby, by reaching out and providing a simple means of support for general questions, the ChatBot can acquaint the user with the services and features of the system, keeping them engaged, and avoiding a hard to navigate GUI.

The ultimate goal of notifications is to alert the users to content provided by the software. This content needs to be consumed by opening the application, often referred to as conversion. The conversion rate of a notification is the rate at which a notification is deemed useful by the user and then the content is consequently engaged with by opening the relevant application. Martin Pielot from Telefonica Research determined that about two-thirds of messaging-related notiﬁcations lead to conversions, whereas the conversion rates for the other categories are notably lower, depicted by the table below showing the conversion rates per notification category [[29](#_References)].

Table

Description automatically generated

*Figure 5 above depicts the conversion rates for different notification categories.*

Figure 5 shows how much more of a priority messaging notification are given over the social/ non-social counterparts. The researcher aimed to take advantage of this phenomenon and captivate the user by means of messaging notifications as opposed to standard Social/Non-Social push notifications. However, it was beyond the scope of this prototype to develop a messaging app in the given time frame. Fortunately, there were plenty of established messaging services that the researcher could deploy an assistance ChatBot on, the two main ones being Discord and Telegram.

Ultimately, the researcher decided to deploy the ChatBot on Telegram due to its interface being far simpler to use than that of Discord. Although they both served similar purposes in the instant messaging industry, Telegram was far more simplistic and basic in its approach and user interface, which is what the researched needed. Although it wasn’t the determining factor, Telegram has over 500 million active mobile users [[30](#_References)], whereas Discord isn’t so popular as a mobile messaging app. This made it highly probable that the user would already have Telegram downloaded, making it less of an inconvenience for a large number of the users..

In order to communicate with the Telegram Bot from the Raspberry Pi, the researcher would learn how to use pyTelegramBotAPI, a Python framework for Telegram Bot API. Although Telepot was initially used, a different python Telegram Bot API library, it was not a good option for the long-term use of the prototype as this library is no longer maintained. Using this unsupported library would put the user experience at risk, which the researcher was not willing to do.

Before the user could interact with the bot, they would have to provide the Raspberry Pi with their unique API bot token. The researcher decided that the most intuitive way of doing this was by typing the token to the system using external push buttons, mimicking a multi-tap (multi-press) [[31](#_References)] text entry system for mobile phones. Although it was a rather basic approach, it was incredibly easy to troubleshoot and most importantly, the user could see exactly how their input was being handled, making it very instinctual. Other alternatives were considered, such as QR code scanning, but it was nowhere near as effortless as this method; the logistics of forcing every user to use third party tools on their phones to convert their token string into a QR code was just not feasible nor was it practical.

A 0.91" OLED module would be used to display the real-time user input. The Adafruit\_ssd1306 library was used to format the size, font, and position of the text on the display.

The researcher wanted to design the ChatBot so as to accept a range of hard-coded commands (all preceded with the character ‘/’) and natural language. In order to begin with the processing of natural language the researcher would need to use the Natural Language Toolkit suite of libraries to tokenise and stem each word from the user input. By tokenising the input, the researcher could divide the string into all the separate substrings (words). By stemming the words, the root word could be obtained. This is further explained in the implementation of the feature in Section 5.

To then make the predictions from the input string of text, the researcher had to build a model for which TensorFlow was used, which the researcher was already using to achieve the weed detection capabilities. However, even with TensorFlow the researcher faced the choice of which front-end framework to use: barebone TensorFlow, TFLearn or Keras. For this feature, using straight TensorFlow would have been slightly verbose, whereas both Keras and TFLearn were deep learning libraries written on top of TensorFlow designed to provide a higher-level API to TensorFlow which provided cleaner syntax with a dedicated focus on fast experimentation. Although the researcher did not have previous experience using either framework, a good reason to have chosen Keras would have been because of its ability to use TensorFlow backend without actually requiring the researcher to learn it, effectively shallowing the learning curve. However, TFLearn worked in a similar way, and not only achieved better performance than Keras, but the syntax was simpler, so it made more sense to opt for the alternative that would put less strain on the deadlines for the deliverables. However, one downside to using TFLearn was that it did suffer from a lack of easily integrated pre-trained models, but as the researcher was going to design and train the model from scratch, this was not an issue.

**4.2 Project Management Tools**

The researcher immediately made the decision to use GitHub to store the code and appendices for the project and to manage version control. For local version control, the researcher used Git. GitHub and Git are industry standard tools for version control and code storage, critical for any form of high-level, professional software development.

As the project was only being developed by a single researcher, there was no need for complex Git GUI tools such as GitKraken, which allow for branch visualisation using graphs and trees. The researcher would only be working off a single testing branch before merging the changes into the main branch, so such powerful software was excessive. If the development of the prototype would have been performed in a team, this alternative would have been more reasonable. Consequently, all version control using Git was done through a command line interface (CLI).

Although the user stories, software requirements and sprint planning were all initially done on paper, the researcher quickly transferred these to a digital format for a better organization and so that the tasks for each sprint could be easily rearranged if a higher priority feature required developing before the other pre-assigned tasks. This was done using GitHub Projects. Accessible from the same repository as the source code, the researcher could create and manage a Kanban style board with excellent user story (issues) management features which helped with the visualization and prioritisation of work. The user stories could be organized using the classifications methods the researcher originally used on paper, that is, MoSCoW and T-Shirt Sizing. These issues could then be arranged into the different sprints for each week depending on the size of the task and its priority factor. Once the features were implemented, these issues could be closed, providing a very black and white means of visualising the remaining tasks.

**4.3 Software Development Methodologies**

As mentioned in the Specification of the project, an agile approach using a modified SCRUM methodology workflow with Unit Testing practices was followed.

Using the SCRUM development framework, the researcher could arrange the major technical and physical milestones into the different time-boxed iterations (sprints) with the main aim of enforcing a release cadence. By having a set of goals to complete each week, the evolution of the prototype was a lot more predictable and controlled. The key SCRUM practice discarded was the daily SCRUM meetings and those adapted were the sprint reviews and sprint retrospectives. The reason for discarding the daily meetings was due to the nature of the development team, as it was only a single researcher working on the prototype. The sprint review was adapted in the sense that the researcher would still intermittently invite feedback about the prototype’s increment in components from the voluntary test users, but there was no single stakeholder involved to dictate the validity of the implemented features. As for the sprint retrospectives, a self-reflection on the progress made to date was still a valuable practice, especially when it could be supplemented by the weekly meetings with the project advisor. This would provide the researcher another point of view to help in determining what changes could be made for the following sprints.

The prototype would come together almost as three separate artefacts: the environment control system, the assistant ChatBot, and the weed detection. Due to this modular nature of the solution, it was in the researcher’s own interest to understand and test the relationship between a set of sensors and actuators before incorporating their functionality with the other product features. For this reason, Unit Testing practices would perfectly suit the development of the software. The smallest testable parts of the prototype (units) could be independently and individually scrutinized for proper operation. By isolating each part of the program, problems with the hardware or bugs with the software could be spotted earlier in the development cycle.

Using a combination of both programming methodologies, the researcher could arrange a suitable workflow and work schedule. The required deliverables for each sprint could be selected while maintaining flexibility and ensuring the development of the prototype would meet the deadlines.



# Implementation and Testing

The researcher had to address the order in which the tasks that made up the prototype were to be executed in order for the system to be efficiently implemented. Changes to the software were to be tested and introduced in stages as opposed to rolling them out simultaneously. This way, a gradual replacement and improvement of outdated features could be achieved while minimizing the impact of failed new features. Having made a project plan and abiding by the sprint deliverables, the researcher greatly reduced the chance of insufficient resourcing for the implementation of the system as the hardware and software requirements, reflected by the collected user stories, were understood from the beginning of development.

The main aspects of the prototype which took up the greater part of the development of the system were the implementation of the environmental control of the greenhouse, the weed detection capabilities, and the intelligent assistant ChatBot.

These features were not developed concurrently. This was done because the time the researcher had allocated for each sprint would not be sufficient to make noticeable progress in multiple facets of the prototype at once, whereas, if the researcher dedicated each development sprint to a specific set of related features the advancement would be more substantial and would help in forming a better picture of the state of the prototype.

With careful consideration, the researcher decided that the environmental control of the greenhouse would be the first feature to be tackled, followed by the Telegram ChatBot, and finally the weed detection. This order was chosen due to the dependant relationship the ChatBot and the weed detection had with the environmental control of the greenhouse; a user interface was meaningless if there was nothing for the user to control or interact with, and the weed detection was almost a subset of the environmental control, so it was not logical to implement it before other more impactful features. The researcher’s short-term prioritization of the capabilities which were crucial to the functionality of the system demonstrated the support for the long-term implementation of the prototype the researcher wanted to offer.

**5.1 Environment Control**

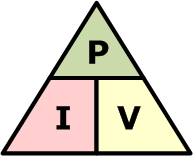
At the beginning of the greenhouse environmental control development, the researcher’s first objective was to implement a basic system with enough features so that the prototype could perform automated irrigation and ventilation. The researcher would not concern himself with the other aspects of the greenhouse automation until these features were tried and tested.

**5.1.1 Relay HAT setup for Irrigation and Ventilation**

As mentioned in the design of the automated irrigation and ventilation of the greenhouse, the main hardware utilized for this aspect of the prototype, other than the Raspberry Pi, was the Relay HAT, a water pump, a NC solenoid valve, and a DC motor. In order to operate both the water pump, the NC solenoid valve and the motor, the researcher had to learn how to control the relays on the Relay HAT so as to turn on/off each component.

Although explicitly stated in the design section of the report, one thing that wasn’t immediately apparent when the researcher was dry running the code to activate and deactivate the relays was that all the terminals were low active, meaning that they are triggered when the Raspberry Pi I/O output is low. This was very counterintuitive at first, and it wasn’t until the researcher thoroughly read the documentation for the Relay HAT that he realised the control signal had to be reversed in order for the relay to work as expected.

Once the terminals on the Relay HAT were opening and closing as intended, the researcher proceeded to connect the water pump (while it was unplugged, of course). As the wattage of the AC 220V water pump was only 3W, the current could be calculated using Watt’s Law and the power formula, where power in this case is the wattage of the water pump.



*Figure 6: Power formula using Watt’s Law*

Using the power formula from Figure 6, the water pump’s amperage could be calculated, in this case 0.0136 Amps. As is standard for a such a small amperage, a cable cross section of 1.5 mm² would more than suffice; this cable cross section is normally used for lighting, allowing for a maximum of 2300 W. From here, it was as straightforward as just attaching a male connector plug which would plug into the mains electricity via the relay hat. The live wire was connected to one of the terminals on the Relay HAT, and the neutral wire was connected directly to the male connector plug (better seen in Figure 7), completing all the wiring necessary to operate the water pump from the Raspberry Pi.

The NC solenoid valve would be connected following a very similar process. A 12V AC to DC converter was instead used to power the NC solenoid valve, wired in the same way as the water pump; a live wire was fed into the Relay Hat from the adapter, and out towards the solenoid valve. This was all that was required to finish assembling the minimum hardware necessary to control a basic automated irrigation system.

However, the Relay HAT still had one free terminal left to use, and there was only one other piece of hardware that could not be operated with the Raspberry Pi’s 3.3V and 5V GPIO pins - the DC motor. The DC motor would tackle the temperature regulation and ventilation in the greenhouse.

The wiring of the DC motor was identical to that of the solenoid valve since the researcher had already made sure that both of these artefacts that needed a DC power supply operated with 12V. The only change that the researcher had to make to the wiring of the solenoid valve was to feed the live and neutral wire into junction strips so that the DC motor’s wiring circuit could be attached to the adapter. Additionally, the researcher was going to use an L298N motor driver as an interface between the motor and the control circuits, this way allowing for speed and direction control of the motor.

For a better visualisation of the complete wiring of all three high voltage electrical components using the Relay HAT, please refer to the diagram below.

Diagram

Description automatically generated

*Figure 7: Wiring of all three high voltage electrical components using the Relay HAT*

Figure 7 above shows how the Relay HAT closes each respective circuit by bridging the live wire (denoted ‘L’ in the diagram) for both the 220V AC circuit for the water pump and the 12V DC circuit for the solenoid valve and DC motor.

It is worth noting that the L298N Driver in the diagram above is connected to the Raspberry Pi GPIO pins and Ground (GND) pins. This was necessary in order to use the motor driver adequately. The diagram below shows a more accurate representation of how the Raspberry Pi connects to the L298N driver.

Diagram, schematic

Description automatically generated

*Figure 8 shows the wiring of the L298N driver to the Raspberry Pi*

In Figure 8, the GND terminals of Raspberry Pi and L298N Motor Driver Module were made common. Now, since only a single DC Motor was being controlled, the researcher only needed to use a single channel of the L298N. In order to do that, the researcher connected the ENA pin of L298N to GPIO14 of the Raspberry Pi. Finally, when wiring the Inputs to the Motor, the researcher connected the IN1 and IN2 pins of the L298N Module to GPIO 15 and GPIO 18 of the Raspberry Pi.

**5.1.2 Automated Irrigation**

At this stage, the greenhouse now had a fully functioning water pump and solenoid valve that could be controlled from the Raspberry Pi. However, before the researcher could assemble these components onto the physical structure of the greenhouse, the automated irrigation had to be truly completed, and for this, the addition of three sensors were necessary.

As described in the design of the irrigation system, a resistive moisture sensor was used to gauge the wetness of the soil; if the soil was too dry, the water pump would be activated, and if too wet it would turn off. The sensor was placed in the soil with the degree of moisture that the researcher wanted to trigger the irrigation system response. By adjusting the potentiometer on the comparator board while it was placed in this soil, the moisture sensor’s detection threshold could be calibrated for its specific use.

Secondly, a HC-SR04 ultrasonic distance sensor would be placed on the inner side of the water drum used for irrigation. By doing this, the levels of water in the drum could be calculated and use this data to open or close the solenoid valve accordingly. Before the researcher could calculate the depth of the barrel, the sensor had to be wired up to the Raspberry Pi. However, it was important to note that the signal the HC-SR04 sensor outputs is of 5V while the input voltage to the GPIO pins are only 3.3V. The signal had to be converted from 5V to 3.3V so as to not damage the Raspberry Pi.

A voltage divider was added to drop the voltage reaching the GPIO pins down to 3.3V from 5V. By utilizing two resistors wired in series, an output voltage a fraction of its input voltage can be produced.

Chart, box and whisker chart

Description automatically generated

*Figure 9: Voltage divider example*

Figure 9 above shows the arrangement of the two resistors in series in order to produce a smaller output voltage.

Text

Description automatically generated with low confidence

*Figure 10: Voltage and Resistance formula*

Figure 10 shows the formula used to calculate the resistance needed for each resistor based on Ohms Law. Using a 1kΩ resistor for R1 in the formula above indicated that a 2.2kΩ was needed for R2.

When using the HC-SR04 distance sensor, the researcher had to take into account whether the solenoid valve had recently been opened or not. This was because ultrasound wave reflection on the water-air boundary layer exhibits a complex behaviour depending on the motion of the surface. With a smooth, undisturbed air-water medium, the ultrasound waves reflect symmetrically to the plane perpendicular to the boundary layer [[32](#_References)]. So, if the solenoid valve had been recently turned on, the researcher would not attempt to measure the water levels as it would not be accurate or representative of the real water levels. The duration which the researcher would wait between activating the solenoid valve and then the sensor would be determined by visually measuring and timing the dampening of the oscillations in the water drum.

The final wiring of all GPIO sensors used in the automated irrigation of the system can be seen in the diagram below.

Diagram

Description automatically generated

*Figure 11 shows the wiring for the sensors involved in the automation of the irrigation.*

The moisture sensor was fairly straightforward to connect. The VCC and GND pins on the sensor itself connected to the ‘+’ and ‘-‘ pins of the comparator board respectively. From the comparator board, VCC connected to the Raspberry Pi’s 3.3V pin, GND connected to the GND pin, and the signal pin was connected to GPIO 12 pin. As for the HC-SR04 sensor, VCC line connected to the 5V pin together with the sensors power line, and the same was the case for the GND line. TRIG (yellow wire) connected to GPIO13. ECHO (green wire) connected to GPIO 19 via the 1kΩ resistor. To this resistor connection, the 2.2kΩ resistor linked the ECHO line with the GND cable, hereby fulfilling the voltage divider purpose.

One thing that was not immediately apparent when designing the prototype was that the solenoid valve was directional. This meant that the orientation of the inlet and outlets would determine whether the valve would work or not while it was closed. Usually there is a sign “→” on the valve body to point out the direction of the medium flow, however, this wasn’t the case with the valve the researcher was using. Thankfully, the system was tested as soon as it was installed, so this mistake was corrected almost immediately. Once the orientation of the solenoid valve was fixed, it would work as expected and would not allow the flow of water when closed.

It is also worth noting that a pipe was connected to the outlet of the solenoid valve which stopped at the very bottom of the tank so that the water would not splash at the electrical components. Furthermore, the researcher drilled overflow holes on the side of the barrel to stay on the safe side in case the solenoid valve failed to close at any point, although this was already safeguarded by the fact the valve was normally closed.

**5.1.3 Automated Ventilation and Temperature Control**

It’s vital to keep the air within the greenhouse moving in order to balance the temperature and prevent fungus and other diseases. In every greenhouse, there are inevitably pockets of hotter and cooler air. Those pockets will impact the health of the plants, but the ability to open the upper panels to allow hot air to escape will eliminate this issue. In order to achieve ventilation and temperature regulation in the greenhouse, the researcher would take advantage of the strategic placement of the window and the physics of air convection to force the unwanted warm air to escape.

The greenhouse window was attached to the frame by two hinges which allowed for a relatively large degree of movement. The researcher would attach a pulley to the ridge beam of the greenhouse and loop a rope through it which would connect to the window on one end and the DC motor on the other.

Although originally the HC-SR04 sensor was going to be placed on the ridge beam next to the pulley, it was too much of an electrical hazard, so the sensor was instead placed inside the greenhouse pointing up at the window. When opening the window, if it measured a depth greater than that of the distance to the window when flush with the greenhouse, the DC motor could be deactivated. The researcher had to make sure that the distance sensor was pointed as close to the connecting hinges of the window as possible, otherwise it would incorrectly signal the DC motor to turn off when the window was not open to its full extent.

Lastly, two DS18B20 temperature sensors were used to measure the external and internal temperature of the greenhouse. One was placed at the highest point possible inside the greenhouse so that the temperature reading would always be a maximum due to the convection of hot air. The second temperature sensor was placed outside the greenhouse, with enough separation so that the radiating heat from the greenhouse would not affect the reading.

A 4.7kΩ resistor was used as a pull-up resistor on the signal line in order for the 1-Wire protocol to work. Otherwise, the Raspberry Pi won’t be able to recognise the sensor and fail at the communication protocol. In other words, the signal line must be high when idle [[33](#_References)] which can be achieved with the pull-up resistor.

A picture containing chart

Description automatically generated

*Figure 12 shows the wiring for two DS18B20 temperature sensors in addition to the motor driver and depth sensor that would all work together to facilitate the ventilation inside the greenhouse.*

Figure 12 shows how both DS18B20 sensors were connected by wiring each VCC pin to the Raspberry Pi’s 3.3V pin. This connection would also be bridged to the data wire by the 4.7kΩ resistor. The data wire would connect to GPIO pin 4 (physical pin 7), and one additional cable would connect the sensor’s ground pin to a ground pin on the Raspberry Pi.

A complete wiring diagram for all components used (depth sensor, motor driver, temperature sensors) to achieve the ventilation in the greenhouse can be found below and under [Appendix N](#_Appendices) (Ventilation Components Wiring (Minus DC Motor and Relay)).

Diagram, schematic

Description automatically generated

*Figure 13. Wiring for depth sensor, temperature sensor and motor driver.*

Referring to the diagram above, the depth sensor’s echo and trigger pins connected to GPIO 6 and 5 respectively, and followed the same VCC and Ground wiring as the sensor used in the water tank. The L298N wiring and temperature sensor wiring has already been discussed above.

After completing the wiring of the different set of moisture, depth, and temperature sensors, in conjunction with the high voltage actuators through the Relay HAT, the researcher had completed the installation of all the features required to truly automate a greenhouse. The initial user stories could be compared to the created prototype and verify whether the original requirements had been achieved or not.

The researcher was pleased to confirm that the prototype lived up to the performance which it intended to achieve; it could successfully control the ventilation, temperature, and soil moisture automatedly.

Although it was implemented after the researcher conducted the user testing of the prototype, there was one helpful recommendation which the researcher wanted to implement so that the implementation process of the system could be as user centred as possible. The suggestion was to notify the user if any of the components were not working as they should, i.e., faulty hardware, so that the user could inspect them and possibly contact customer support to replace them. This recommendation was converted into a user story and later added into GitHub Issues.

The implementation of this feature was achieved by testing the expected state of each sensor after the activation of each actuator. If the state did not correspond with the state the system expected it to be, the prototype would notify the user of a possible malfunction. This would work both ways for sensors and actuators. Take the ventilation system for example, if after activating the DC motor so that the window opened, the distance sensor would read the same depth to the window, the issue would most likely originate from the DC motor as it was likely that it was not reeling in the string through the pulley. On the other hand, if the sensor was simply not providing readings when accessed by the Raspberry Pi, it would be clear that the sensor was faulty.

**5.2 Intelligent Assistant ChatBot**

The Telegram bot was designed to fulfil the specific purpose of answering questions and commands. It worked as a means of customer support, providing information about the live state of the greenhouse environment, and allowing for the user to override the automated tasks if requested. The researcher would first start with the implementation of a user input system, and for this feature, push buttons were used together with an OLED display to show the user input.

**5.2.1 User Input and Bot Syncing**

Before the user could pair their bot with the Raspberry Pi, they had to create a new bot using Telegram. The user would have to search for @BotFather, the one bot to rule them all [[34](#_References)], on their Telegram app. On the chat with BotFather, a new bot could be created using ‘/newbot’. After providing a name and unique username for the bot, the token required to access the HTTP API would be provided. From here, the user would simply have to input this token into the system for the Raspberry Pi to pair with their phone.

Although using a complete QWERTY keyboard to gather the user input was the design that would put the most emphasis on the usability of the system, it was not the most accessible nor was it the most practical to make. This was the case mainly because this bot token was only needed a single time to configure the system, so it would render a complex one-time-use keyboard obsolete after the process was completed. For this reason, the researcher decided to only use a set of 6 push buttons to achieve the same effect. After studying the characters that composed a Telegram Bot

API token, the researcher could decide which symbols and/or characters were a necessary part of the vocabulary. Each button would allow the user to alter the input string in a different way as seen in the diagram below.

Text, application, icon

Description automatically generated

*Figure 14 shows each pushbutton and respective button cap designs for the different actions.*

Referring to Figure 14, there would be a button for each of the following actions: lower case alphabet characters, integers from 0 through to 9, upper case alphabet characters, symbols, backspace and finally enter.

Although mentioned in more detail in Section 6.2, the researcher would go on to improve the accessibility of the prototype by adding braille onto the button caps using correction fluid to achieve the raised relief necessary. The braille used was a direct translation of the buttons labelled ‘ABC’, ‘123’, and ‘? ; :’. As for the upper case letters, braille doesn't have a separate alphabet of capital letters. Instead, there's a “code” that tells the reader the next letter is capitalized. That “code” is a dot-6. The word ‘back’ would represent Backspace and the word ‘go’ would represent Enter.

The way in which the researcher designed this feature was so that these buttons would a mimic multi-tap (multi-press) text entry system for mobile phones. For each button, if pressed consecutively, it would loop through the available characters until the user paused for long enough so that the system registered that pause as a signal to store that character. This way, with only 6 buttons, the user could intuitively enter their bot token and edit it as necessary.

This design was thoroughly evaluated prior to and after implementing it so that it could be as user-centred as possible. There was no need for any additional buttons after assessing the usability of this method of user input, and neither were there any buttons that were dispensable, so the researcher continued with the implementation of the feature, now developing a suitable user interface using the OLED display.

As the OLED display was only 0.91” in size, there was not enough room to create a fully-fledged home screen. Instead, the researcher opted for a rather simple greeting message that would prompt the user for their bot token and if the connection was successful, the screen would display a welcome message. If the connection was unsuccessful due to an invalid bot token, the system would notify the user of this. No more functionality was needed from these features, as this was all that was necessary to begin using the Telegram bot.

A picture containing text, electronics

Description automatically generated

*Figure 15, OLED display prompting the user for their bot token*



*Figure 16, OLED display correct setup message*

Figure 15 above shows the display as the user is typing their bot token. Figure 16 above shows the display once the bot token has been successfully used to pair the Raspberry Pi with their Telegram bot.

Diagram

Description automatically generated

*Figure 17: pushbuttons and OLED module wiring*

Figure 17 above depicts the wiring for all six of the pushbuttons in addition to the OLED module. All buttons had the left pin made common with the Raspberry Pi’s ground pin, and the right pin connected to a GPIO pin ( in this case GPIO 17, 27, 22, 10, 9 and 11). The OLED’s power line connected to the 3.3V pin, SDA and SCL lines connected to their equally named counterpart pins on the Raspberry Pi, and ground pins were made common.

The buttons and OLED display would be placed under a more aesthetically pleasing case which would house the button caps with the Braille as seen in Figure 18 below.

A picture containing text

Description automatically generated

*Figure 18 shows the final appearance of the token input multi-press keyboard*

Although the researcher had tested the verification of the token so that no empty tokens were allowed, there was an edge case which crashed the program of which the researcher was initially unaware of. The researcher assumed that attempting a connection with an empty token and an invalid token would throw the same error, but this was not the case. An empty bot token would throw a ‘Bot token is not defined’ exception, however, an invalid token would throw a HTTPS error code, in this case 404 when the token didn’t exist, or 401 when the access to the token was unauthorised. This wasn’t initially an issue, but the researcher quickly found out that there was no way of catching this error with a normal ‘try, except’ block of code.

For this reason, the researcher resorted to the Requests HTTP library for Python. Requests allowed the researcher to send HTTP requests extremely easily to the following Telegram URL:

*https://api.telegram.org/bot[TOKEN]/getMe*

where ‘[TOKEN]’ would be replaced by the user provided HTTP Telegram Bot API Key. Following this method, if the status code returned by the request was either 404 (incorrect token) or 401 (unauthorised token), the researcher could forcefully throw an exception in the code by printing a non-declared variable, this way allowing for the researcher to catch this error and use it to determine if the token was valid or not.

Graphical user interface, application

Description automatically generated

*Figure 19 shows a snippet of code displaying the Requests library functionality*

Figure 19 shows the specific snippet of code which is described above. Using this method, if the HTTP request status code was one of either 404 or 401, the researcher could prompt the user for the token once again.

**5.2.2 Commands and Natural Language Processing**

As highlighted in the design of the ChatBot, a Python library called pyTelegramBotAPI was used for the implementation of the Telegram Bot API. In as little as 4 lines, the researcher could use the token from the user input and set up the bot to start polling the Telegram servers for new messages. Using the ‘message\_handler( )’ function from this library, the researcher could begin the processing of user input.

Initially, only commands were accepted as forms of user input, so any message that was not preceded by a ‘/’ was not processed by the Raspberry Pi. This was achieved by using the ‘commands’ parameter for the message handler function; from here, the researcher could set a fixed response for this command which would almost immediately appear on the Telegram chat, and also use the command to alter the state of the system depending on what was requested. Although this would already work as a means of customer support and to provide information, it was by no means a pleasant experience. The researcher wanted to provide the end user with the familiarity of an intelligent human capable of understanding their requests without having to prefix every command with a forward slash. Not only would this provide a more meaningful interaction between the system and the end user, but it was also a far more scalable option for the long-term longevity of such prototype. As more companies embrace intelligent systems, it only made sense to integrate such intelligent language processing capabilities into the prototype so that it would not feel outdated within a matter of years. User experience was a priority, and the usability of the system had to be at the forefront of the development. For this reason, the researcher would decide to incorporate natural language processing skills into the system.

This extension of the ChatBot had to fall in line with the same user requirements that the original version fulfilled. It was still designed to provide information about the system, only that now the user would have a much easier time requesting it. As described in the design of this feature, the researcher would use TensorFlow and TFLearn to train a machine learning model using data that was first pre-processed by using NLTK Python library. The processed data would be used to make predictions on what the user meant, otherwise known as intents, and reply accordingly.

Since the ChatBot would only serve a limited purpose and answer only domain related questions, there was no need to download external datasets for training as this would be excessive. The data on which the model would be trained on would be written by the researcher. This data would be structured as Python dictionaries inside a JSON file, and would store data in ‘key:value’ pairs. Within each dictionary, the data would be further split up into classes (tags) which would represent the specific context of the message from the user, i.e., activate irrigation, temperature query, water level query, etc., where each tag would reflect the purpose of the message. The dataset would also store patterns of expected input that would belong to the context represented by the tag. Finally, the dataset would also hold the adequate responses the system should provide the user with when the model has decided what the intention of the message is.

Text

Description automatically generated

*Figure 20. A code snippet of an example* ***mock*** *implementation of the JSON file structure.*

Figure 20 above shows example training data structured as described (See [Source Code](#_Appendices) – intents.json for training data used in practice), where each category of messages is accompanied by the expected patterns the model should be trained to recognise, and the responses the system should provide for that tag. By structuring the data in this manner, the researcher could make sure that as long as the intent was correctly determined, the responses would always make sense as they were hard coded as opposed to being a naturally generated response by the model. The researcher could not ensure that if the system was to generate the A screenshot of a computer

Description automatically generatedresponses intelligently they would make much sense, so in order to achieve the familiarity of an intelligent human response which the researcher was required to do, it was far more feasible to do this by providing the system with a limited number of logical responses from which it could choose from.

When training the model, by eliminating the extra characters that might make the word different, such as an apostrophe or hyphen, the model can become noticeably more accurate. This is what the researcher would achieve by stemming (reducing) the words to their root word. Before the researcher could stem the words, the NLTK library was used to tokenise each entry. After stemming the tokenised training data, all duplicate entries were also removed which helped give the researcher a rough estimate of the vocabulary size.

Neural networks and machine learning algorithms require numerical input to train models, so the list of stemmed words would not be a valid input for the model. In order to overcome this, the researcher followed the very common practice of converting this string input into a bag of words (BOW). Using a bag of words, the researcher could represent each pattern within a tag as a list the length of the total amount of distinct words in the JSON file (vocabulary). Each position in this list would represent a specific word from the vocabulary, and a 1 or a 0 would denote whether that word exists in the specified pattern from the tag or not – also known as one-hot encoding. Now that the input was formatted, the output also had to follow the same structure. Therefore, a similar procedure was followed for all the tags in the dataset, where each position in the list of tags represents a distinct tag, and a 1 or a 0 would denote whether the tag corresponds to the input pattern or not. This way, the researcher could numerically represent which patterns belonged to which tags by using the instances of each word in the pattern as the determining factor.

With all the data pre-processed and in a format which the model could be trained on, the researcher now had to develop this model. The goal of the model was to associate each bag of words to the tag they belong to. A feedforward neural network with two hidden layers was used to achieve this. The reason for using a feedforward neural network was because of its ease to learn the relationship between independent variables which serve as inputs to the network, such as the bags of words, and the dependent variables which serve as outputs to the network, such as the tags. The architecture of the model was defined using TFLearn.

*Figure 21. A code snippet in which the text classification model is trained.*

Figure 21 above shows the creation of the neural network used to train the ChatBot classification model. The input\_data( ) function is used to feed the data to the network, where the shape it is expecting for the model is the length of the bag of words. The second line in Figure 21 adds a fully connected layer to the network. This layer starts after the input data and would have 10 neurons in this hidden layer. The third line follows the same principle, only that it now fully connects to the already created hidden layer from the second line. The output layer shown in the fourth line is the last layer with a number of neurons equal to the number of tags in our training data. The softmax function is used as the activation function in the output layer of neural network models that predict a multinomial probability distribution, used to decide whether a neuron should be activated or not calculating weighted sum and further adding bias with it. Softmax was the best suited activation function for this scenario as it is used as the activation function for multi-class classification problems where class membership is required on more than two class labels [[35](#_References)]. This way, the researcher could get a percentage certainty (probability) for each tag of the output layer. By then applying a regression layer, the researcher could model the relationship between the input and the output, and predict the output of the dependant variable, in this case the tags. Finally, using DNN( ) the researcher could train the model using this neural network.

After training the model by feeding it the bags of words as inputs, and the tags as targets, the researcher obtained a relative accuracy of 97%. This was fairly impressive, but was expected as the model had been trained on a limited amount of data. After the completion of user testing, the researcher would go back to the intents and further refine the patterns for each tag, in addition to expanding on the number of tags the system was trained on. This would bring the accuracy of the model slightly down to 94%, but not enough to have an impact on the usability of the feature.

Once the trained model was saved, the researcher could now start making predictions. As the model was trained on bags of words, it would need the user input to be formatted in the same way in order to make predictions; the user input would be tokenised and stemmed in the same way the training data was. The user input could be represented by a bag of words that encoded all the words from the training data, this way if a word from the user input existed in the training data, it could be represented with a ‘1’ in the bag of words. This bag of words could now be fed to the model for it to make its prediction on.

The output of the model prediction was a list in which it would assign a probability value to each tag from the intents. By obtaining the highest value from the model predictions for the list of tags, the researcher would know which specified tag was awarded the highest probability by the model, and hence display an appropriate response to the user input.

At this point, the ChatBot was now capable of providing adequate responses to hard coded commands (preceded with a ‘/’) and also to a natural language, in this case it was exclusively English. It could now discern the intent that the user had in mind when asking a question, which was the main user story that this added intelligent functionality aimed to tackle.

**5.3 Weed Detection**

In order to provide the prototype with weed detection capabilities, the researcher had to first set up an off-the-shelf object detection model on the Raspberry Pi. For this, a pre-trained TensorFlow Lite model was downloaded and deployed on the Raspberry Pi. This model was used as a steppingstone on which to later use transfer learning methods to reuse this pre-trained model and customize it for the given task at hand. This pre-trained model used the EfficientDet-Lite 0 architecture. It is a state-of-the-art object detection model for edge devices such as the Raspberry Pi, although the researcher would later test different architecture iterations so that this feature would work with the one best suited for the job; it was clear from the beginning that inference speed was not as high priority as model accuracy as weed detection would only be performed once or twice every day.

Using a camera module for the Raspberry Pi and the OpenCV computer vision library, the researcher could feed an image to the model, and output the score associated to each label from the pre-trained model. The camera was placed so that it could see the tops of three different stacks of plant pots, this way maximising the possibilities of detecting weeds in any of the three columns ([Appendix O WeedCameraPOV](#_Appendices)).

The object detection model would also set the bounding box coordinates for the detected object. Although this was a functioning object detection in and of itself, it was not tailored to the user requirements and would not distinguish between different plants, so the researcher had to use this pre-trained model as a template and customize it by training it on new data, in this case images of weeds.

The first step was to collect and label training images so that a new training dataset could be used on the model. It would be far too laborious, if not impossible, to manually collect images of all the weeds that were common to all different regions and climates of the world. For this reason, the researcher decided to simply focus on a specific weed that grew in the area and climate in which the prototype was being developed, and this would serve as a good indication of how well the object detection model could distinguish weeds from other plants, essentially serving as a proof of concept.

The researcher would use a digital camera to capture as many different images as possible of the specific type of weed that that model was going to be trained to detect. These images would then be loaded into LabelImg, an open-source graphical image annotation tool, to manually set the bounding boxes and labels for the weeds in each image. In general, the more images collected, the better the model would be trained. However, because the researcher was using a transfer learning technique to train the model, the dataset could be relatively smaller and still work.

Graphical user interface

Description automatically generated

*Figure 22. LabelImg being used to annotate images with labels for each object.*

The image above shows LabelImg being used to annotate each image with the required label. Although the researcher only wanted to detect a single type of weed, an attempt to differentiate between the different stages of the weed’s life cycle was going to be made. This way, if the model could detect the weed while still in its early life before the seeds from a mature plant were scattered across the greenhouse, it could save the end user from having to deal with weeds continuously. Consequently, the weed’s life cycle stages were to be divided into ‘WeedStage1’, ‘Weed’, and ‘WeedBloom’, where WeedStage1 would denote the seedling stage, Weed would denote the yellow flower stage and WeedBloom would denote the final stage of the weed’s life cycle with the white dandelion-flower-like appearance. Once this annotation process was complete, the training data would be composed of JPG images and XML annotations for each image. The researcher would split the collected data into an 80:20 ratio of training and validating data as was the general rule of thumb when dividing datasets [[36](#_References)]. The training dataset would hold the image examples used to fit the machine learning model during the learning process; for a classification task like this one, a supervised learning algorithm would help in determining which combinations of variables and features generate a good predictive model. As for the validation dataset, it would be a sample of data held back when training the model which would then be used to give the researcher an unbiased estimate of the skill of the final tuned model.

As stated in the design of this feature, the researcher would then use TensorFlow Lite Model Maker, a Python library that allowed for the training of an object detection model using custom datasets in just a few lines of code. The model would be trained on Google Collab (see [Appendix D](#_Appendices)), an online tool from Google that gave the researcher free access to computing resources to train machine learning models.

After loading the training and validating datasets into Google Collab, the specific iteration of the EfficientDet-Lite family of model architectures could be selected.

Table

Description automatically generated

*Figure 23. Shows the different performances of the EfficientDet-Lite model architecture iterations.*

When selecting the EfficientDet-Lite model architecture, latency (inference speed) was not a determining factor as stated previously. The only influencing factor was the Average Precision. In Figure 23 above, the average precision is measured against the COCO 2017 validation set [[37](#_References)]. For this reason, it was only logical to use the model architecture that would provide the highest precision, in this case EfficientDet-Lite4, when it came to the detection of weeds hereby improving the performance and reliability of the weed detection.

Once selected a model architecture, the researcher could begin training the model. The number of epochs used would be tested to see how it affected the validation loss values; validation loss would indicate how well the model fits new data. To better visualize the relationship between the validation loss for the model and the number of epochs used, the researcher would test and then plot these values against each other. This was all done to avoid a machine learning phenomenon called overfitting. Overfitting refers to a model that learns the detail and noise in the training data to the extent that it negatively impacts the performance of the model on new data and its ability to generalize. This is because the random fluctuations in the training data are learned as features for the objects when in reality these do not apply to any new data.

Chart, line chart

Description automatically generated

*Figure 24. Graph of Validation Loss vs. Epochs when training the object detection model.*

From the graph, the researcher could learn the optimal number of epochs to use for the training of the model by observing the point at which the validation loss stops noticeably decreasing. The researcher could interpret from the graph that the validation loss started to plateau after 22 epochs, so that was the minimum number of epochs that would be used when training the model.

Using the now trained model, the researcher could evaluate the model using the validation dataset. The mean Average Precision (mAP) for the object detection model using each of the different model architectures was also drawn up and visualised with a graph; this would give a good indication of the accuracy of the model.

Chart, line chart

Description automatically generated

*Figure 25. Graph of mean Average Precision vs. EfficientDet-Lite iteration*

The graph would confirm the architecture predictions that the researcher had drawn out when choosing an iteration from the Efficient-Det Lite architecture. The best architecture for this purpose was the Efficient-Det-Lite-4 as inference speed (latency) was not of priority, but accuracy was.

Finally, the object detection model could be exported to the Raspberry Pi and tested in real time. Using the same pre-trained object detection script used for the pre-trained TensorFlow Lite model, the researcher could now set the model parameter to be the newly exported weed detection model.

The model could now detect the different stages of a weed’s life cycle it was trained on because the researcher retrained the pre-trained model to recognise these new objects.

A hand holding a yellow flower

Description automatically generated with low confidence

*Figure 26. Test detections on stock image of similar but not identical weeds to those which the model was trained on*

Figure 26 above shows a stock image of a similar, but not identical weed, and the model proved to the researcher how well it could generalise to new domains as it effectively detected the weeds it was trained to recognise. However, as is obvious, the initial seedling stage of the weed was not recognised in this image. This will be tackled more in depth in Section 10, but the researcher suspected it was because of the way the pictures were taken on which the model was trained on. The images were taken with a backdrop of more weeds and greenery, which the researched thought may have interfered with the detection of the seedling as the colours would blend into each other and would make it harder to determine the edges of the object.

The Raspberry Pi could now effectively detect weeds, and the researcher could now use any weed detections to notify the end user of their existence, this way fully delivering on the expected features from the user stories. The researcher had now created a dynamic system capable of providing the end user the security of a weed-free growing environment. The main benefit of this feature was how scalable it was, as there were no limitations to the number of different weeds and plants the system could detect, other than the effort the researcher would have to put into the laborious classification and labelling of each plant within an image, although TFLite only supported a maximum of 25 detections per frame.

**5.4 Responsible AI: Ethical, Social and Professional Concerns**

In 2018, Google introduced its AI Principles [[38](#_References)], which guide the ethical development and use of AI in research and products.

In line with these principles, the researcher aimed to design the ChatBot and the weed detection adhering to Responsible AI (RAI) practices. The technology was not designed to cause harm, and it would not gather information for surveillance violating internationally accepted norms. The researcher’s prototype would most definitely not contravene widely accepted principles of international law and human rights.

Understanding and trusting AI systems is important to ensure they work as intended. For this reason, neither the weed detection model nor the AI ChatBot were trained off of sensitive data that needed privacy preserving safeguards. Additionally, as the researcher had studied the user demographic and tested the artefact in depth while user testing, the true impact of the predictions, recommendations, and decisions of the intelligent system could be assessed and predicted in advance. It is worth noting that there existed no corporate ownership of any personal data provided by the test users as any private information was disregarded when evaluating the system from the testing results. Any underlying biases in the data that could contribute to the reinforcement of existing stereotypes were nullified, such as with the ChatBot it was vital that there were no derogatory terms or offensive language in the hard coded responses. The researcher was aware that it’s important to maintain the security of the bot’s received messaged and responses dataset in order to avoid the loss of sensitive corporate information or private user information. For this reason, none of the inputs were saved, and the outputs were stored locally on the Raspberry Pi.

The prototype aimed to be socially beneficial, taking into account a broad range of social and economic factors of the users so that it was not discriminative and could be used in as wide of a range of scenarios as possible. For example, by using a drip irrigation system which the prototype supported, it could drastically help in reducing water waste and usage for the end user in water-scarce regions. Additionally, the inclusion of braille writing on the surface of each button was made in order to better the societal impact of the product, this way providing a more inclusive experience for all users alike. Furthermore, although it was a minor aspect of the design, the software was at no point developed to be incrementally rewarding the more a gardener used it so as to not provide an addictive design which could cause unhealthy habits.

Finally, professional concerns were evaluated prior to the development of the prototype so that the researcher was certain of the legal context in which the work was to be carried out. Caution was taken so as to completely understand the relevant applicable laws and property rights of the software used.

**6.3 Accessibility, Usability and User-Centred Design**

The researcher aimed to implement as many accessibility-driven features as possible. Although the prototype wasn’t developed in order to aid those with disabilities when it came to gardening, the researcher realised that the prototype was indirectly doing just that. By automating the irrigation, ventilation, and water tank levels of the system, it was now accessible to a wide range of users with accessibility needs as they would now not have to go through the strenuous task of performing the environmental control which the prototype could provide.

Additionally, it was worth noting that all of Goggle Chrome, Android and iOS support Telegram and they all provide a plethora of accessibility driven features such as Switch Control, Guided Access, Text to Speech, and more. This made the prototype incredibly useful for those who rely on these features to navigate software and websites, further cementing the range of users which the prototype could be used by.

Finally, by adding braille to the button caps, the researcher safeguarded the only external interactive feature which could not draw from an operating system’s accessibility tools. This way, all interactive features of the prototype would deliver on the user-centred design requirements for accessibility in the implementation.

**5.5 Unit Testing**

Using unit testing practices throughout the entirety of the implementation of the prototype features would be helpful in order to save the researcher a lot of time and headaches in the long-term maintenance of the product. Writing good unit tests would give the researcher more confidence that the updates and refactoring of code didn't have any unintended consequences or break the code in any way. If for example, an update to a function in the project breaks several sections of the code, even if the function itself is still working, unit tests will notify the researcher of this problem so it can then be fixed.

As opposed to using print statements multiple times in the code and occasionally testing to see if the program will still reach that section, unit tests can automate this process and more importantly, they are far more scalable and easy to maintain. Additionally, if the researcher was to test multiple functions at once using the print statement method, there wouldn’t be a way to see at a glance which sections failed and which succeeded; each missing print statement would have to be accounted for when reviewing the console output, which means that a missing print statement might not be obvious at first.

The researcher would use the ‘unittest’ Python testing framework which was in the standard Python library, so there was no need to install it. Test cases for each of the functions and scripts that the researcher wanted to test were created. In order to do this, a class which inherited from unittest.TestCase was made. This gave the researcher access to a variety of tools and testing capabilities.

All methods used to test the separate units required a very specific naming convention – the methods needed to start with ‘test’ in order for the library to know which methods represent unit tests. If a method didn’t start with the word ‘test’ it wouldn’t be executed.

Text

Description automatically generated

*Figure 27. Test functions using unittest library.*

Referring to Figure 27, by inheriting form unittest.TestCase, the researcher had access to a variety of assert methods. An assertion method compares the actual value returned by a test to the expected value, and throws an AssertionException if the comparison test fails. If this happened, the unittest testing framework would then identify the test as a ‘Failure’, denoted by an ‘F’ in the output console.

Initially, the tests were run from the CLI using “python -m unittest script.py” where “script.py” was the test which the researcher was testing. In order for the tests to be run from the IDE, a slight change to the testing file was made. Python, unlike other languages, has no main( ) function that gets run automatically - the main( ) function is implicitly all the code at the top level. Scripts have a built-in variable ‘\_\_name\_\_’ which evaluates to the name of the current module. However, if a module is being run directly, then ‘\_\_name\_\_’ instead is set to the string "\_\_main\_\_".

A picture containing graphical user interface

Description automatically generated

Thus, by testing if the variable ‘\_\_name\_\_’ is equal to the string “\_\_main\_\_”, the researcher could check if the file was being run directly by the python script or if it was being imported. If the conditional is met, the script would run unittest.main( ), this way, the researcher can access all of the tests directly from the IDE when running the tests from the unit testing script.

With this technicality out the way, the researcher completed setting up the unit tests for each testable feature. The unit tests were created in conjunction with the main prototype features while they were being developed. When the tests failed, the researcher would know straight away where the bug in the code was.

The researcher wanted to test some edge cases too for some of the features. In the case of the water pump for example, the researcher wanted to test whether the pump would not be activated when the water levels were below a certain point in addition to the main use case of the pump, which is turning on if the soil is deemed dry. The goal wasn’t to write as many tests as possible, the goal was to write good tests that could account for as many scenarios as possible. As opposed to shooting for full coverage of the edge cases, it was best to just make sure to write tests that were good enough to catch mistakes.

If there were ever any errors found through standard debugging of the code, the researcher would follow the good practice of updating the unit tests with a test that would have caught the problem that was just found. This way the researcher would make sure to not revisit the same bugs repeatedly.

# Testing and Evaluation

As the system began to grow bigger in features, it soon became necessary for the researcher to explore different user testing methods. It was not feasible for the researcher to be the only tester of the product, so involving a third party was necessary. An initial idea was to have an in-person focus group where members of the focus group could each have a chance at using the system and collaboratively agree on positive and negative points, from which the researcher could take notes and make further progress with their feedback. Due to the physical nature of the prototype, this focus group approach made the most sense as the researcher needed to observe users interacting with the system in order to obtain valuable feedback. At the time of testing, there were no coronavirus restrictions in Spain so long as the testing was carried out in open space; if the testing was to take place indoors for any particular reason, masks would have to be used. Another method that was explored was one-on-one interviews where a participant could openly give their thoughts. However, the researcher decided not to pursue this method as it was harder to draw conclusions or implement meaningful features to the product only based on open feedback.

The final and chosen option the researcher used was a questionnaire, which provided the best of both worlds. The questionnaire could be given to participants and had a mixture of multiple choice and open-ended questions. Multiple choice questions allowed the researcher to draw meaningful and statistical conclusions, and open-ended questions were better tailored for feature feedback that can’t be adequately conveyed by means of multiple tick boxes or a Likert-scale where only a single answer can be chosen.

The user testing process was carefully carried out over the course of a week towards the end of the second to last development sprint, this way the researcher could still have plenty of time to consider and implement any of the new features that could be drawn from the feedback. Before the researcher could begin user testing, a questionnaire had to be drafted ([Appendix E](#_Appendices)) for testers to run through, in order for them to get a quantitative evaluation of what users thought of the site. Space was also left for users to leave direct feedback for the website, and some of these pieces of feedback were turned into user stories once testing was completed and added to the sprint backlog (see Appendix F).

All testing was performed under the constant supervision of the researcher, so that if there was any faulty software which could cause a negative experience, the researcher would not be left with the user’s description of the problem as the only means of solving the issue.

**6.1 Methodology**

Overall, seven users were surveyed for the testing of the website, and meaningful data was gathered while respecting the requirements of the Ethics declaration form. The researcher made sure that the pre-filled form submitted by the module director would cover the basic needs of the user testing, which it did as it was non-clinical research with little to no risks. The researcher made sure to provide the participants with adequate consent forms and questionnaires to fill in and sign where it stated that every piece of information gathered from the questionnaire would be used for the report and presentation and that absolutely no reference would be made concerning their identity and personal information other than their general demographic.

In the interest of being ethically considerate, and in line with the ethics regulations imposed upon the researcher, all participants were made fully aware that they did not have to take part in user testing and their participance was completely voluntary. All participants were also made aware that participance would require completing a questionnaire, and that their identity would remain anonymous in order for there to be no way to track the feedback to the test user. Participants names were omitted from all questionnaires and replaced with a participant number, which the researcher would then use classify and structure the suggestions and testing data. The participants were also made fully aware by the researcher that if at any time during the testing of the prototype that they no longer wanted to continue, they could stop, and their feedback would be discarded. Likewise, the participants could ask to have their responses discarded even after the completion of the testing process.

After ensuring the participants were fully aware of all of these conditions, each participant was handed a consent form for them to complete prior to starting the testing of the system. All participants completed in-person testing with the researcher to reflect as closely as possible the real-world scenarios a possible user would come across when using the prototype. All participants were also somewhat acquainted with basic gardening techniques and landscaping. However, not all of them had used Telegram in the past. This made for a very valuable set of test users, as they could point out higher level inconsistencies with the technical aspects of the environmental control that might affect a newly started gardener; they would also be of great value when setting up the ChatBot as the process could be somewhat troublesome for inexperienced users, and this could help the researcher in making the bot implementation more intuitive. It is worth noting that all participants were asked the same questions, but this did not restrict them from asking about additional features or functionality.

The quantitative data was compiled for the researcher to visualize and gain an understanding of the overall usability of the prototype. By the end of the testing and evaluation, the valuable feedback was turned into a total of 7 new user stories. These user stories were then put through the categorizing process of a MoSCoW board, risk evaluation, and t-shirt sizing that all previous user stories had been through. This allowed for novel aspects which would increase the degree of usability to be added. Having performed the user testing process with over a week to spare meant that these new features could be considered and implemented without being too pushed for time and maintaining the high design standards which were kept throughout the rest of the development process.

**6.2 Results**

Some notable pieces of feedback that was retrieved from user testing, without digressing too much into the results of the entire questionnaire, are discussed below.

Although the ChatBot was capable of providing logical responses to most messages, there was a huge edge case that the researcher hadn’t taken into account when designing it. If a nonsensical message was sent to the ChatBot, for example a one-word typo such as “hwllo” instead of “hello”, it would still reply to this message even if it was not very confident of the category it belonged to, many times miscategorising the message completely. This was problematic as it completely ruined the intelligent persona that the researcher was trying to create with the ChatBot. For this reason, the researcher decided to add a threshold to the confidence of the machine learning model; if the probability for the main predicted tag was of less than 70%, the ChatBot would simply reply with a message along the lines of: “Sorry, I didn’t get that” to indicate uncertainty in the response.

Additionally, as stated in Section 5.1.3, the user testing also gave rise to the suggestion of testing the integrity of the hardware periodically so as to make sure that no components were faulty and would cause the system to malfunction. This was almost immediately refactored into the software so that each sensor would test the state of the feature to be altered after the actuators had been activated. If their state was identical, the actuators were malfunctioning. On the other hand, if the readings from the sensors were not in between an expected range, i.e., temperature should be between 5 and 35 degrees Celsius, the sensors could be deemed faulty. This critical evaluation of the system would help with the long-term usability of the hardware and make the prototype scalable for future work on it.

Outside of user feedback regarding specific functionality of all features, a question about the usability of the system emerged that helped the researcher better place the prototype in its commercial context. One user wasn’t sure how the greenhouse could help grow different vegetables that required such vastly different needs at the same time, such as tomatoes and cauliflower for example, where one is more suited to a warm summer climate, and the other is more prominent during cold seasons. The researcher took this opportunity to clarify the objective of the prototype as it was obvious that the purpose of the automated greenhouse was still somewhat unclear. The greenhouse would only serve as a means of assistance to the user, in the sense that it could help with regulating the optimum temperature and other variables so as to keep them in their optimum range automatedly. What it was not designed to do was provide a system in which any plants could be grown at any time of the year regardless of region and climate. The gardener would still have to be aware of the appropriate crops for each season to some degree, as the greenhouse would simply provide a larger window of time in which these could be sowed and reaped, letting the user get a jumpstart on the season and plant earlier for example as is a very common practice. This interaction made it clear to the researcher that the greenhouse was being misrepresented, so consequently an effort was made in correcting the description of the system in the product report and instruction manual to better define the aims and objectives of this product.

This interaction only brought attention to one aspect of the prototype, but it gave the researcher room for reflection as to what future work and improvements could look like. Though this is explained in greater detail in Section 10, the researcher contemplated the possibility of controlling several environments at once to allow for exactly what the test user had inquired about – the simultaneous growing of crops belonging to vastly different climates or seasons. This would make the prototype a more direct competitor to MultiGrow as it could tackle several growing environments at once too.

Moreover, security measures were considered after a test user asked if the system would retain the last used bot token so that if rebooted, it would directly connect to the user’s phone and make it harder for an unauthorised user to substitute the first used bot token with their own. This seemed like a logical feature to have but the researcher did not think the latter part of the suggestion was reasonable. A system of this nature needed a fail-proof way of resetting the saved user inputs in case of the users losing or breaking their phone, otherwise it would render the whole prototype useless as there would be no way to re-sync the Raspberry Pi to the new device. For this reason, the researcher decided to only implement the first half of this suggestion so that on reboot it would directly connect to the last saved telegram bot by pressing ‘Enter’ upon the system turning back on, but by typing in a new token and then pressing ‘Enter’, the system would overwrite the previously saved token and use the most recent one instead.

Lastly, although it did not originate from the user testing sessions, a meeting between the researcher and the product supervisor lead to an interesting feature which would improve the product’s accessibility – the implementation of braille on the button caps. As described loosely in different sections of the report, this feature would allow for those users with visual impairment to effectively input their Telegram Bot API token and then proceed to use the native accessibility tools of their device’s operating system, such as screen readers and assistive touch, to interact with the ChatBot using Telegram.

Although the most features discussed above drawn from the user feedback were implemented, there were some suggestions that simply could not fit into the timescale and scope of the project. As mentioned above, the possibility of controlling separate environments was not an easy addition to make, as the Raspberry Pi was almost at its full capacity of GPIO pins, and the researcher would not have time to purchase and set up a whole new greenhouse to test this functionality on. This suggestion was best left as a possible feature for future iterations of the prototype as there were other aspects of the product that still required attention and it was not reasonable to begin the implementation of such a large feature with only over two weeks left till the submission deadline. Additionally, the possibility of detecting ripened crops and pests was also entertained by the researcher. However, the prototype’s ability to detect weeds already worked as a proof of concept, so this added functionality wouldn’t be so much of a breakthrough feature as it be merely an add-on to the pre-existing one. Also, this suggestion would not make or break the prototype, as it already performed sufficiently well without it, so the researcher decided to prioritize the feedback which directly referred to malfunctioning or incomplete features as opposed to that which was simply an interesting addition.

In conclusion, the user testing provided the researcher with invaluable feedback that allowed the researcher to refine the product into something more fitting for the users, most importantly the researcher was able to add features which were not initially part of their user stories into the final version of the system. Without the user testing, the researcher would not have been able to recognise some of the issues that were raised. The feedback from participants was generally positive, with a vast majority of participants noting how easy it was to use the prototype. However, the researcher was aware that with such a small testing group, it was hard to confidently generalize the evaluation outcomes to a wider population. Despite this, because the testing group were so incredibly representative of the demographic the researcher aimed to target with the developed product, the feedback was received with much glee. The user testing might not account for every scenario the prototype could be put through, but it helped solve a number of issues that the researcher would not have noticed without it. In particular, those issues closely related to the performance of the system, such as the intelligent response to illogical messages, were given utmost priority as they were the main culprits of poor short-term usability.

# Description of the final product

Below is a full description of the final product. For more information on how to make use of the product, please refer to the product manual ([Appendix E](#_Appendices)). Additionally, for pictures of the final product, please see [Appendix O](#_Appendices). Finally, to see the final product in action, please see [Appendix H](#_Appendices).

**7.1 Environmental Control**

The developed product is almost unnoticeable at first glance. With the Raspberry Pi enclosed within a waterproof casing inside the greenhouse, other than the cables that come out from it, there isn’t much to give away the existence of an automated system. The irrigation tank looks no different to what it did prior to the addition of the solenoid valve, water pump and HC-SR04 depth sensor, as all of these components are hidden within the tank itself, making it ever so slightly more discreet. The DC motor is also out of the user’s initial line of sight, and the temperature sensors blend into the monochrome architecture of the greenhouse, so it all makes for an incredibly subtle system with astonishingly powerful features.

The final prototype can effectively control the temperature, soil moisture and ventilation inside a greenhouse. By means of external high voltage hardware, it can keep the irrigation water supply constantly topped up, the greenhouse ventilation window opened or closed, and the soil at the optimum moisture at all times throughout the day.

**7.2 Telegram ChatBot**

The major functionality of the ChatBot is to provide the user with assistance and support during the 24 hours of the day, 7 days a week.

The system uses a sleek 6-button arrangement for the typing and editing of the user’s bot token which can be visualised in real time on a small 0.91” OLED display. This feature is small and discreet as it only serves a very limited purpose which most times only needs to be performed once, thus not getting in the way of the user throughout the rest of the prototype’s lifespan.

As Telegram is one of the world’s top 10 most downloaded apps with over 550 million active monthly users [[39](#_References)], it is highly likely that the gardener would already be a user, which makes the ChatBot far more convenient to use as there would be no additional downloads required in this scenario.

The ChatBot provides a clean interface for the prototype with simple hard-coded commands that encompass most of the available functionality. Additionally, its natural language processing ability makes the ChatBot feel more genuine and intelligent, acting almost like a gardening partner as opposed to a cold piece of technology. Although the ChatBot cannot account for every piece of user input that it may receive, it fairs well with generalised system-specific queries and commands.

Overall, the Telegram ChatBot can be used as a simple and unobtrusive way to check on the greenhouse environmental variables and to provide a friendly growing companion which will alert the end user if it detects any weeds so that these can be removed as soon as possible

**7.2 Weed Detection**

Though this feature primarily works off the groundwork that the Telegram ChatBot lays, it’s complex enough to warrant its own dedicated section.

The weed detection capabilities of the system allow for the camera module connected to the Raspberry Pi to detect weeds in a given space. Though the object detection model was only trained on a single species of weed, it has proven to generalise well to similar varieties and can therefore be easily improved so as to account for a larger number of weeds.

When the Raspberry Pi detects a weed in a frame, it will send a message to the user via the Telegram ChatBot notifying them of the possibility of a weed growing within their crops.

This feature is trained to recognise three distinct life-cycle stages of the same weed in order to maximise the possibility of the weed being detected prior to the seeds scattering throughout the greenhouse.

# Summary and Conclusions

After a thorough background research, the researcher aimed to take advantage of the existing literature and products on the market in order to develop a unique solution that could take the best of the existing prototypes, and provide additional functionality too.

The measures taken by the researcher to develop an automated greenhouse in order to aid newly started gardeners has culminated into a physical computing-based solution for the environmental control needed in a greenhouse to succeed with sowing and reaping crops. The prototype has alleviated the difficulties which many gardeners face when first taking up horticulture.

The three main features (environmental control, ChatBot and weed detection) of the prototype work together in a well-coordinated manner. The prototype is adaptable to a wide range of greenhouse designs, so the researcher was confident that it could be used as an add-on to a gardener’s pre-existing greenhouse as much as it could work if it was to be sold as a greenhouse and an automated system all together.

The report provided a means of reflection and helped the researcher understand the positive qualities and drawbacks of the developed prototype. Though its limitations are not many, there is plenty of room for improvement, and even more room for reconsideration of the technology used and features implemented, this is further discussed in Section 9.

The outcomes of this individual project have provided an unbiased set of findings that point towards the many advantages of greenhouse automation, including the achievable societal impact it could have. The researcher managed to enhance his professional skills due to the technology and tools adopted. Having taken up widely used frameworks for the problems that were tackled, the researcher has gained valuable experience with these industry standard tools, allowing for a reassessment of the project to be approached from a more knowledgeable perspective. Overall, this project has provided the researcher the chance to merge the overlapping fields of Computer Vision, Artificial Intelligence and Robotics into a single solution that perfectly encompasses the potential each of these disciplines have when used simultaneously.

**8.1 Changes in Design and Implementation from Mid-Term Report**

The final developed product closely resembles the one depicted in the Mid-Term report, however there are various design aspects that changed during the implementation of the system. After progressively obtaining a deeper understanding of the utilised tools and the project context, some previous design decisions were ultimately discarded and through processes such as the user testing, new user stories were created which gave way for new designs and implementation of these features.

Firstly, the background context and market were studied in far more depth before finalising the environmental control design which was being worked on at the time of writing the Mid-Term report. This provided the researcher with a wider range of previous, similar, or related work whether it was through journal papers, market research or statistics found online. This vantage point allowed the researcher to view and better compare similar systems to the one which was being developed, and helped with the assessment of previously used (common) features in addition to the relevant ethical and social considerations in place.

As described in Section 4.1.1, the initial plans of diluting liquid fertilizer into the water upon activating the irrigation system were superseded by the more accessible approach of using granulated fertilizers.

Additionally, as the project plan described in the Mid-Term report did not take into account the intelligent ChatBot or weed detection features, the researcher feels that the original technical milestones were slightly underwhelming, and they did not reflect a thorough understanding of the desired features to be implemented nor a desire to go above and beyond the basic requirements for a system of this nature. This is most noticeable in the description of a user messaging system via SMS which is very vaguely described and not evaluated properly. In retrospect, the researcher is glad that the final product boasts a variety of additional features which were not originally considered as they have made this prototype all the more unique and have provided a learning opportunity for a number of industry standard tools in the fields of Computer Vision, Artificial Intelligence, and Robotics.

# Appraisal

If the researcher were to tackle this problem again using the knowledge of all the tools used and the skills learned, there are several features that would be approached differently.

If the budget for the development of the solution was larger, a better quality camera could be used so to increment the possibilities of detecting weeds in the greenhouse. Additionally, as was mentioned in Section 4, if the researcher were to tackle this problem again, capacitive moisture sensors would be used due to their longevity and near to no corrosion and rust. This replacement would make the prototype far more suitable for an intensive long-term use.

Lastly, the researcher would design the window ventilation feature slightly different as it was not always accurate, and the depth sensor was hard to place in an optimal position. A different approach using lasers and LDR would be a suitable alternative as this would provide pinpoint precision as to when the window has opened or shut completely.

Hardware aside, the researcher would not change any of the software used for the model detection or the intelligent ChatBot. Instead, a more in-depth understanding of the frameworks would be achieved first so that the researcher could adapt any online resources with more ease and freedom, this way being able to implement more powerful features.

Although it would not have been a limitation to the development if it wasn’t for the bad optimisation of the VNC Viewer, the researcher would have liked to work on the project using Raspberry Pi OS Bullseye so that the system could be as up to date as possible with the most recent firmware updates and patches. The only way to circumvent this issue was to use an external monitor and keyboard which the researcher did not own; this could be a justifiable purchase for future iterations of the prototype.

Additionally, the researcher would train the object detection model using a much larger set of images. Although the transfer learning methods allowed for a set of only 60 training images to be used for this object detection model, there is no doubt that the accuracy would be greatly improved if there were more images of the different stages in the weed’s life cycle for it to train on. Furthermore, the researcher would take all training and validating images on a plain white background so that the contour and edges of each object are more defined, this way allowing the weed detection model to detect the first stage seedling with more ease.

In terms of project management, some more advanced tools and more in-depth planning could have offered better task allocation, allowing for more room for deviation when developing features, especially nearing the end of the development timescale. The researcher learned a variety of new things through the course of this project. The agile methodology is a good example of this since the researcher was able to enhance his understanding of how agile worked and how it is beneficial to short projects such as this one. Having short sprints allowed the researcher to have a very fluid plan, so tasks were able to be changed easily on the spot if needed. A sprint retrospective at the end of each sprint then allowed any inconsistencies for that particular sprint to be highlighted and improved before starting the next sprint.

There were no major deviations from the initial project plan, although it did constantly evolve with the growing possibilities of the addition of new features. With the benefit of hindsight, the main lesson learned throughout the whole course of the development was the real complexity of intelligent systems. It was not as easy as the researcher had anticipated to create and train a ChatBot using a neural network, or to train an object detection model using existing model architectures for that matter. For this reason, the researcher would recommend any other scholars looking to tackle this same issue using the same approach to first develop, or at least thoroughly understand, the intelligent systems used in the prototype, and then progress onto the physical implementation of the environmental control functionality.

# Future Work

The researcher has several follow-up ideas and features that could make this prototype ever so slightly more appealing as a commercial product to not only newly started gardeners, but also industrial scale plant nurseries.

The first of the many possible future additions would be to allow for different environmental control for multiple greenhouses. This way the product could provide different climates for different crops and would not limit the gardener to only the ones in season. With this addition, it is only natural that a stronger water pump and additional motors would be needed in order to achieve the same control on a larger scale.

Secondly, the researcher would like to try different model architectures for the weed detection in order to favour an accurate but slow model. Although the EfficientDet family of model are very versatile, the researcher only really needed to test a single image every 12 hours to check for any weeds. Although EfficientDet aims to cover as many positive qualities as possible, it is inevitable that there will be trade-offs, and high frame rates were not the objective, so an even slower object detection model would be far more suited for the task at hand and would help with the detection of the seedling stage of weeds as the utilized architecture was not entirely able to detect them.

Furthermore, the researcher would like to work with larger natural language datasets in order to attempt natural language generation for the responses. Although the idea of using a small JSON dictionary for the limited number of use cases the ChatBot would need to handle worked as a proof of concept for this iteration of the prototype, a means of producing intelligent responses together with an intelligent understanding of the user input would make the system that much more flexible to the incoming messages and would allow for a larger variety in the responses which would further the impression of an intelligent personal assistant to the end user.

Additionally, the researcher would like to develop a dedicated app for the system so that it would not need to work off of Telegram’s Bot API. To warrant this change, the app would have to provide more functionality than the ChatBot, or at least present the greenhouse data in a more intuitive manner. Consequently, by using a dedicated app, the researcher could follow through with the discarded design decision of using QR codes to synchronise the user’s phone with the Raspberry Pi, which would provide a quicker way of performing the first time set up of the prototype.

Lastly, still appealing to budget gardeners, the researcher wishes to adapt the prototype so that it could be used with polyethylene greenhouses that lack the physical structure that polycarbonate greenhouses have. With both of these types of greenhouses covered, the prototype would be much more appealing as the polycarbonate family of greenhouses are the pricier of the two types mentioned above.

## Acknowledgments

The author would like to thank Dr. Jacky Visser for being incredibly helpful and responsive during the course of this project, and for providing endless feedback during the supervisor meetings. The author would also like to thank his forever person, Izabella Reyes, his sister, Gloria Blanco, his mother, and the entirety of M.G. for providing a voice of reason and guidance throughout.

# References

[1] [Modes of Greenhouse Water Savings](https://www.researchgate.net/publication/308242989_Modes_of_Greenhouse_Water_Savings), Nolan O’Connor

[2] [Plant tolerance to high temperature in a changing environment](https://www.frontiersin.org/articles/10.3389/fpls.2013.00273/full" \l ":~:text=Climatological%20extremes%20including%20very%20high%20temperatures%20are%20predicted%20to%20have,resulting%20in%20wide%20spread%20famine), Front. Plant Sci., 31 July 2013

[3] [The Effects of Greenhouse Activities on Psychological Stress, Depression, and Anxiety among University Students Who Served in the U.S. Armed Forces](https://journals.ashs.org/hortsci/view/journals/hortsci/52/12/article-p1834.xml), HortScience

[4] [Americans are Growing More than Ever](https://www.miraclegro.com/en-us/library/americans-are-growing-more-ever), Miracle Gro

[5] [The Power of Plants](https://www.theceomagazine.com/executive-interviews/agriculture-farming/michael-sutterer/), Mike Sutterer

[6] [All Things Gardening forum: Giving up](https://garden.org/thread/view/48850/Giving-up/), National Gardening Association

[7] [An intelligent control for greenhouse automation](https://www.sciencedirect.com/science/article/abs/pii/S0168169900001332), Tetsuo Morimoto

[8] [Intelligent control based fuzzy logic for automation of greenhouse irrigation system and evaluation in relation to conventional systems](https://www.researchgate.net/publication/236484174_Intelligent_control_based_fuzzy_logic_for_automation_of_greenhouse_irrigation_system_and_evaluation_in_relation_to_conventional_systems), Javadi P Kia

[9

] [Needs of Plants](https://letstalkscience.ca/educational-resources/backgrounders/needs-plants), Let’s Talk Science

[10] [MultiGrow](https://autogrow.com/products/multigrow), Autogrow

[11]

[Why AutoGrow](https://autogrow.com/why-autogrow/why-autogrow), AutoGrow

[12] [AutoGrow - MultiGrow Multiple Room Environment Controller](https://www.hg-hydroponics.co.uk/autogrow---multigrow-multiple-room-environment-controller-13487-p.asp), HG-Hydroponics

[13] [Liebig’s Law of the Minimum](https://en.wikipedia.org/wiki/Liebig%27s_law_of_the_minimum), Wikipedia

[14] [What is CO2 enrichment and is it for you?](https://generalhydroponics.com/what-is-co2-enrichment/), General Hydroponics

[15] [Single-board computer](https://en.wikipedia.org/wiki/Single-board_computer), Wikipedia

[16] [Understanding Flame Retardant Polycarbonate](https://www.totalplastics.com/understanding-flame-retardant-polycarbonate/" \l ":~:text=UL%2094%20Flammability%20Rated%20Polycarbonate&text=The%20Horizontal%20Burn%20(HB)%20rating,the%20material%20as%20self%2Dextinguishing.), TPI

[17] [5 tips for temperature control in your growing environment](https://www.greenhousemag.com/article/2020-structures-guide-5-tips-for-temperature-control-in-your-growing-environment/), Greenhouse Management

[18] [Pros and cons of granular and liquid fertilizers](https://www.canr.msu.edu/news/pros_and_cons_of_granular_and_liquid_fertilizers), Jim Isleib, Michigan State University Extension

[19] [Capacitive v/s Resistive Soil Moisture Sensor](https://www.hackster.io/devashish-gupta/capacitive-v-s-resistive-soil-moisture-sensor-e241f2), hackster.io

[20] [RPi Relay Board](https://www.waveshare.com/wiki/RPi_Relay_Board), WaveShare

[21] [HC-SR04 Tests R0](https://app.box.com/s/sj7du1n32in2777rcoi2), rigonz2@netscape.net

[22] [Temp\_sensor.py](https://github.com/gerardoblanco/Greenhouse-Automation/blob/master/temp_sensor.py), gerardoblanco/Greenhouse-Automation

[23] [Why GPIO Zero is Better than RPI.GPIO for Raspberry Pi Projects](https://www.makeuseof.com/tag/gpio-zero-raspberry-pi/" \l ":~:text=The%20RPi.,first%20system%20for%20beginner%20coders), MUO

[24] [RPi.GPIO documentation](https://sourceforge.net/projects/raspberry-gpio-python/), SOURCEFORGE

[25] [Why TensorFlow](https://www.tensorflow.org/about), TensorFlow

[26] [What is TensorFlow?](https://www.infoworld.com/article/3278008/what-is-tensorflow-the-machine-learning-library-explained.html), InfoWorld

[27] [TensorFlow Lite Model Maker](https://www.tensorflow.org/lite/guide/model_maker), TensorFlow

[28] [Transfer Learning](https://www.tensorflow.org/tutorials/images/transfer_learning), TensorFlow

[29] [Dismissed!: a detailed exploration of how mobile phone users handle push notifications](https://www.researchgate.net/publication/327216693_Dismissed_a_detailed_exploration_of_how_mobile_phone_users_handle_push_notifications), Martin Pielot

[30] [Telegram Usage Ranks](https://www.similarweb.com/app/google-play/org.telegram.messenger/statistics/), similarweb.com

[31] [Multi-tap](https://en.wikipedia.org/wiki/Multi-tap), Wikipedia

[32] [Ultrasonic wave interaction with air-water boundary layer](https://www.isud-conference.org/proc/split/ISUD-08_019_Schmitt.pdf), Philippe Schmitt

[33] [1-Wire](https://en.wikipedia.org/wiki/1-Wire), Wikipedia

[34] [Telegram Bots](https://core.telegram.org/bots), Telegram

[35] [Softmax Activation Function with Python](https://machinelearningmastery.com/softmax-activation-function-with-python/), Machine Learning Mastery

[36] [Training and validation dataset division](https://stackoverflow.com/questions/13610074/is-there-a-rule-of-thumb-for-how-to-divide-a-dataset-into-training-and-validatio), Stack Overflow

[37] [COCO 2017](https://colab.research.google.com/github/google-coral/tutorials/blob/master/retrain_efficientdet_model_maker_tf2.ipynb" \l "scrollTo=Gb7qyhNL1yWt)

[38] [Responsible AI](https://www.blog.google/technology/ai/ai-principles/), Google

[39] [Telegram Usage](https://en.wikipedia.org/wiki/1-Wire), BackLinko

# Appendices

Note: All appendices can be found in the GitHub repository for this project labelled accordingly.

The link to the repository is the following:

*<https://github.com/gerardoblanco/AC40001>*