Development of an automated Controlled Environment Agriculture (CEA) platform for the horticultural industry

A thesis presented in partial fulfilment of the requirements for the degree of

Master of Engineering

In

Mechatronics

At Massey University, Albany,

New Zealand

Shreyas Borgaonkar

2023

Abstract

Farming methods have evolved over time, with many countries shifting from traditional to organic farming. The increase of organic farming means that there is more land required. This increased land requirement is solved by deforestation; however, this increases negative climate change impacts due to greenhouse gases. Moreover, the land available for organic farming is decreasing due to urbanisation.

Controlled Environment Agriculture (CEA) is seen as a solution to this problem. CEA is the farming of crops in a technologically enhanced environment wherein variables required for plant growth e.g., light, water etc, are monitored and controlled. Vertical Farming is a newer method of Controlled Environment Agriculture, wherein plants are provided with controlled nutrients while being suspended in a soilless medium.

Currently, while vertical farming has been implemented in small and large forms; it has not been explored in great depth. Hence, due to less familiarity with implementation and technology, there are various gaps and barriers observed in the method. As each plant variable is monitored and controlled, the system uses sensors, actuators, microcontrollers, and communication methods which increases technological complexity. Due to this, farmers shifting to a more modern farming method is slower and less likely. There are five overall barriers observed in the transition to modern technological farming. They are: lack of knowledge, high implementation cost, cyber security risks, need for human intervention, and lack of temperature control implementation. These barriers and gaps are preventing the vertical farming industry to develop and become a more common method of farming.

A small-scale controlled environment was developed and proposed as a potential method to reduce these barriers. The system included all modern technologies such as ROS (Robot Operating System), website implementations, and various components, such as sensors, LEDs, fans, etc, controlled using a microcontroller. The main objective was to develop an effective, technologically-enhanced solution, which could control temperature using a PID controller. Using these results, the solution could be improved to better match industrial standards and specifications which would allow upscaling of the solution for large scale implementation. Finally, the successful results of the solution provided a robust answer towards reducing the gaps and barriers identified.

Acknowledgements

I would like to express my deepest appreciation to my supervisor, Dr Frazer Noble, whose experience, support, and guidance laid the foundation for this research and thesis. Also, many thanks to Massey University for providing the opportunity, technical resources, and funding to complete this project.

Lastly, I would be remiss in not mentioning my parents, Santosh & Prakruti Borgaonkar, who have been immense pillars of support and encouragement. Without their motivation, this thesis and my accomplishments would not have been possible.

Many thanks to all the well-wishers for their support.

Table of Contents

Abstract i

Acknowledgements ii

1.0 Introduction 1

2.0 Literature Review 3

2.1 Background - Current state of agriculture and related problems 3

2.2 Background – Controlled Environment Agriculture to mitigate current problems 7

2.3 Technological barriers with current Controlled Environmental Agriculture solutions 11

2.4 Monitoring and Sensors 16

2.5 System Integration 18

2.6 Actuation 22

2.7 Literature Review Summary 25

3.0 Methodology 27

3.1 System Overview 27

3.2 Mechanical Setup 28

3.3 Electronics Setup 38

3.4 Software Setup 41

3.5 PID Controller Development 49

3.6 System Integration 65

3.7 Final Experimental Methodology 72

4.0 Results 78

5.0 Conclusions & Discussion 97

5.1 Conclusion 97

5.2 Discussion 98

6.0 Future Work 99

7.0 References 102

8.0 Appendix 110

Table of Figures

Figure 1: Increase in organic agricultural land area between 2016 [5] and 2017 [6] 4
Figure 2: Aquaponics Process [16] 7
Figure 3: Diagram of NFT setup [19] 8
Figure 4: Diagram of Deep Water Culture setup [19] 8
Figure 5: Diagram of Ebb & Flow setup [19] 9
Figure 6: Diagram of Aeroponics setup [19] 9
Figure 7: Different Processes of Automated CEA [15] 11
Figure 8: Dual Lens Camera (stripped front to remove excess material) [34] 12
Figure 9: Smart Farming in Ireland - Implementations of Technology [87] 14
Figure 10: Current Level of Automation in vertical farming (dashed line represents current level) [88] 15
Figure 11: BH1750FVI Sensor [89] 16
Figure 12: DHT11 Sensor [90] 17
Figure 13: DS18B20 Waterproof Temperature Sensor [57] 18
Figure 14: Raspberry Pi 3 Model B Microcontroller [93] 19
Figure 15: Arduino Uno Microcontroller [94] 20
Figure 16: ESP8266 Microcontroller [95] 20
Figure 17: Example air conditioning unit used in large-scale CEA implementations [96] 23
Figure 18: 12 V, 150 W Electric Ceramic Thermostatic PTC Heating Element [97] 24
Figure 19: Similar element to 12 V, 1 A, DC Fan [98] 24
Figure 20: Vertical Farm and corresponding sub-systems 27
Figure 21: Egmont Hydro Kit [91] 28
Figure 22: Initial design of NFT Model 29
Figure 23: Solidworks CAD Model of Enclosure 30
Figure 24: Enclosure CAD Design and design features for corresponding constraints (numbers correlate with constraint numbering in table 3) 31
Figure 25: Removable panels - Top-most panel, tab and slot fit (Top left), Electronics Chamber side panel with four rotating lock tabs (Bottom left), Front window panel with a tight fit (Right) 33
Figure 26: Model components in final position. Top plant chamber inside view, Ultra-bright LEDs, Temperature Sensor, STEGO Heater, DIN Rail, DC Fan, Plant Holder – Numbers correlate part numbers in table 4 35
Figure 27: Model components in final position. PVC Tray, PVC Tray Cover, PVC Tap, Zinc Knob / Handle, Pump, Temperature Sensor (waterproof) – Numbers correlate part numbers in table 4 36
Figure 28: Final Model CAD Design (Left), Final Model after manufacturing and assembly (Right) 37
Figure 31: Final installation of the PCB (Top Right), Raspberry Pi (Bottom Left), Switch Mode Power Supply (Bottom Right) within enclosure 40
Figure 32: Final installation of electronics platform (spacer – top left, MDF base – top right, acrylic platform – centre bottom) within enclosure 40
Figure 33: Directory Structure 42
Figure 34: Publisher and Subscriber Files created. Subscriber Node (left), Publisher Node (Right) .. 43
Figure 35: Initial Temperature Reading Code 44
Figure 36: GPIO port and component on/off 45
Figure 37: Basic Implementation of Flask Web Application Framework [80] 45
Figure 38: First function (main web page) part of Flask Python code 46
Figure 39: Web Page display when first opened 47
Figure 40: Second function within Flask Python code, input temperature is assigned to 'setpoint' variable, and current temperature is displayed 47

Figure 41: User Input of set point on web page 48
Figure 42: Web Page display after temperature input has been received 48
Figure 43: Expected Transient Response from system and key parameters L, T, and K [81] 49
Figure 44: Transient Response of our system and the key parameters highlighted 50
Figure 45: Closed Loop system schematic for obtaining step response and tuning PID Controller 51
Figure 46: PID Controller block and initial values of P, I, and D 51
Figure 47: Ziegler-Nichols tuning formula to calculate the Kp, Ti, and Td parameters 51
Figure 48: Output of initial closed loop system and PID controller 52
Figure 49: Expected outputs and characteristics after increasing P, I, and D parameters [82] 53
Figure 50: Output observed after Kp decreased by 50 %, Ki decreased by 50% and Kd increased by 50 % 54
Figure 51: Output observed after Kp decreased by 50 %, Ki decreased by 50% and Kd increased by 50 % once more 55
Figure 52: Further refined output by decreasing only Kd by 50 % 56
Figure 53: Decreased Overshoot and smoother curve output after further reduction in Kd parameter 57
Figure 54: Final iteration of decreasing Kd to observe smoother curve output 58
Figure 55: Curve output after Kp and Ki decreased by 50 % to decrease settling time 59
Figure 56: Final curve output with final P, I, and D parameter values 60
Figure 57: Final curve output zoomed out view - curve stabilised at desired set point 60
Figure 58: Final parameter values using auto-tuning function in Simulink software 61
Figure 59: Output curve using auto-tuning function in Simulink software 61
Figure 60: Solver Configuration changed from 'variable step' to 'fixed step' with a step size of 0.1 .. 62
Figure 61: Summary of PID Parameter Values and Performance & Robustness Values 63
Figure 62: Change of controller to discrete time while PID values are kept same 63
Figure 63: Publisher and Subscriber Nodes merged. The topics were swapped in the other microcontroller 65
Figure 64: ROS Nodes and Topic Overall Structure 66
Figure 65: Transfer Function Simulink Modelling & Components 73
Figure 66: Step Input Block - final value represents set point; initial value accounts for offset to match ambient temperature in physical model 73
Figure 67: PID Controller Block - discrete time PID controller with PID parameters matching the parameters used in the physical model and obtained from calucations (section 3.5) 74
Figure 68: Transport Delay Block - provides the 400 second delay to replicate the time before temperature change in the physical model 74
Figure 69: 'House Heating System' Simulink model layout with key components highlighted 75
Figure 70: Threshold block and Parameters 76
Figure 71: Heater Block and Parameters 76
Figure 72: House Block and Parameters - all values were calculated using our physical models specifications 77
Figure 73: Visual Studio Code Terminal Outputs when initiated 78
Figure 74: Home page of Flask Web Application 78
Figure 75: Flask Web Application page after entering set point (30℃) 78
Figure 76: Terminal Outputs (one on left, other on right) after temperature was set, including change in set point 79
Figure 77: Flask Web Application Output after 'refreshing' the webpage 79
Figure 78: Iteration 1 - Physical Model Experimentation Result 80
Figure 79: Iteration 2 - Physical Model Experimentation Result 81
Figure 80: Iteration 3 - Physical Model Experimentation Result 82

Figure 81: Iteration 4 - Physical Model Experimentation Result 83
Figure 82: Iteration 5 - Physical Model Experimentation Result 84
Figure 83: Iteration 6 - Physical Model Experimentation Result 85
Figure 84: Iteration 7 - Physical Model Experimentation Result 86
Figure 85: Iteration 8 - Physical Model Experimentation Result 87
Figure 86: Iteration 9 - Physical Model Experimentation Result 88
Figure 87: Iteration 10 - Physical Model Experimentation Result 89
Figure 88: Iteration 11 - Final Physical Model Experimentation Result (with Flask Web App) 90
Figure 89: Simulink Transfer Function Model Experimentation Result 91
Figure 90: Simulink House Heating System Model Experimentation Result 92
Figure 30: Board view of Printed Circuit Board in Eagle PCB Design Software 110

Table of Tables

Table 1: Absolute change in Greenhouse Gas emissions between 1990 & 2016 in main agricultural sectors of the world [13] 6
Table 2: Advantages & Disadvantages of different types of Controlled Environment Agriculture 10
Table 3: Advantages and Disadvantages of different Microcontrollers 21
Table 4: General, Nutrient Solution Reservoir and Electronics Housing Constraints set for design of enclosure 29
Table 5: Full List of Mechanical Components used in model system 34
Table 6: Full List of Electronic Components used in model system 38
Table 7: Parameter Value comparison - Manual Tuning v Auto-Tuning 62
Table 8: Summary of all curves' characteristics 93

1.0 Introduction

Farming methods have evolved over time, with many countries shifting from traditional farming to organic farming recently. Traditional farming relies on the use of pesticides and other plant growing regulators. In contrast, organic farming involves biological pest control, compost, and crop rotation methods etc. Organic farming is better, as traditional farming depletes natural resources used for crop growing, consequently degrading crop quality [1]. Organic farming improves soil conditions; thus, yielding a better crop [2].

The increase of organic farming means that there is more land required. This increased land requirement is solved by deforestation; however, this increases negative environmental impacts, such as greenhouse gases. Moreover, the land available for organic farming is decreasing due to soil erosion and urbanisation. Therefore, climate change and loss of land are two major problems currently adversely affecting normal farming methods.

Controlled Environment Agriculture (CEA) is seen as a solution to this problem. CEA is the farming of crops in an environment wherein variables required for plant growth e.g., light, water etc, are monitored and controlled. Greenhouses and Vertical Farming are the two main types of CEA. Vertical Farming is a newer method of farming. Plants are provided with controlled nutrients while being suspended in a soilless medium. There are various Vertical Farming methods, of which, Nutrient Film Technique (NFT) is commonly used in the industry. NFT includes a thin layer of water streamed past each plant's roots. Vertical Farming is more advantageous, as it can be implemented anywhere, including urban city areas, as it incorporates vertical stacking. Greenhouses use a horizontal layout for stacking, thus requiring more space and still being affected by the problems applicable to organic farming.

Currently, while vertical farming has been implemented in small (research based) and large (industrial) forms; it has not been explored in great depth for a long time. Hence, due to less familiarity with implementation and technology, there are various gaps and barriers observed in the method. As each plant variable is monitored and controlled, the system uses sensors, actuators, microcontrollers, and communication methods which increases technological complexity. Due to this, farmers shifting to a more modern farming method is slower and less likely. There are five overall barriers observed in the transition to modern technological farming. They are: lack of knowledge, high implementation cost, cyber security risks, need for human intervention, and lack of temperature control implementation. These barriers and gaps are preventing the vertical farming industry to develop and become a more common method of farming.

Hence, the aims of this thesis are as follows:

- Investigate the current state of Controlled Environment Agriculture.
- Identify the main gaps and barriers which affect the development of this method.
- Develop a robust solution targeting the gaps and barriers.
- Test and evaluate the effectiveness of the solution in reducing the gaps and barriers.
- Discuss improvements to aid in mitigating gaps, and enhance upscaling possibilities.

The objectives are to perform a robust literature review on the current state of art. This would enable us to create a model of a CEA system, to develop a framework, that could be used to create a modular CEA unit and control the unit using modern technology. Examples of the modern technologies includes ROS (software libraries and tools to enable development of robot applications), website implementations, and various components, such as sensors, LEDs, fans, etc, all controlled using a microcontroller.

The scope of this work was not to develop a commercially viable system; rather, to design a small scale solution using cost-effective components and frameworks, such as ROS, which are easily accessible. The solution was to be designed to be modular, which would enable prospects of upscaling the system, i.e., enlarging all aspects that could transform it into a commercially viable system.

The contributions of this thesis include:

- A robust literature review into the current systems used for various farming methods.
- Developing a prototype CEA system, which uses technologically advanced methods, that are currently less employed in the industry.
- Including temperature monitoring and control, as well as automation i.e., lack of human intervention required; these can be considered as the novel aspects of this thesis.
- Highlighting potential improvements that can be incorporated in the future to enable upscaling of the system.

The remainder of this thesis is organised as follows: Section 2 – Literature Review, this includes a review of the current state of agriculture and related problems, the possible mitigations using CEA, the barriers in the solutions, and the components and frameworks used; Section 3 – Methodology, this includes the mechanical, electronics, and software setup of our solution, as well as PID controller development, system integration, and the final experimental methodology; Section 4 – Results, this presents all the temperature output graphs as well as the website implementation results, followed by a summary table of all key findings and discusses possible reasons and shortfalls; Section 5 – Conclusions & Discussion, this includes a final conclusion of this thesis and a summary discussion of the results and their relation to the gaps identified in the literature review; Section 6 – Future Work, this includes the

potential improvements required to progress and develop an upscaled version suitable for the current CEA industry.

2.0 Literature Review

This chapter presents and discusses a variety of globally published literature on the current state of agriculture, its related problems, and the mitigations that Controlled Environment Agriculture (CEA) methods can offer. It also discusses the technological barriers within CEA mitigations and describes currently used technologies that can serve as solutions to these barriers, such as components for monitoring, sensing, actuation, and full system integration. The information gathered and presented in this section builds the foundation for this thesis and project, and summarizes the barriers or gaps in the current CEA industry.

2.1 Background - Current state of agriculture and related problems

Many countries, such as Australia, the United States of America, and Germany started transitioning from traditional to organic farming in the early 1900s. Towards the end of the 20th century, organic agriculture was a more commonly used methodology than traditional methods. The primary reasons for this change was the increased use of pesticides and fertilizers and their negative effects on crop [3].

Organic agriculture is a combination of traditional and modern methodologies, including the use of modern equipment and technology. Traditional farming incorporates the use of pesticides and other "plant growing regulators", which make the soil acidic; hence, harmful for eating without appropriate treatment. In contrast, organic farming involves biological pest control, compost and crop rotation which ensures that eating any crop without treatment is safe [4]. Another study reached a similar conclusion and stated that rotating and varying crops, improving soil conditions, and controlling pests naturally are all key aspects of organic agriculture [2]. Traditional farming is seen to deplete natural resources used for crop growing, consequently degrading crop and environment quality. Organic farming "substitutes cultural and biological inputs for synthetically made fertilizers and chemicals for crop nutrition and pest management" [1].

There has been a consistent increase in organic agriculture, and, consequently, the land resources required. Globally available organic agricultural land area increased from 57.8 million hectares (578,000 km^2) in 2016 [5], to 69.8 million hectares (698,000 km^2) in 2017 [6]. A notable contributor to the increase in land area has been Australia, providing an increase of 5 million hectares (50,000 km^2) from 2015 to 2016 [5], and an increase of 8.5 million hectares (85,000 km^2) from 2016 to 2017 [6]. Deforestation is one of the sources of this increase in land area. It is worth noting here that although deforestation increases land area, it also causes a negative environmental impact. Emissions

such as greenhouse gases collect in the atmosphere and warm our planet. Trees reduce this motion by using the carbon dioxide; thus, deforestation is increasing greenhouse gas emissions [7].

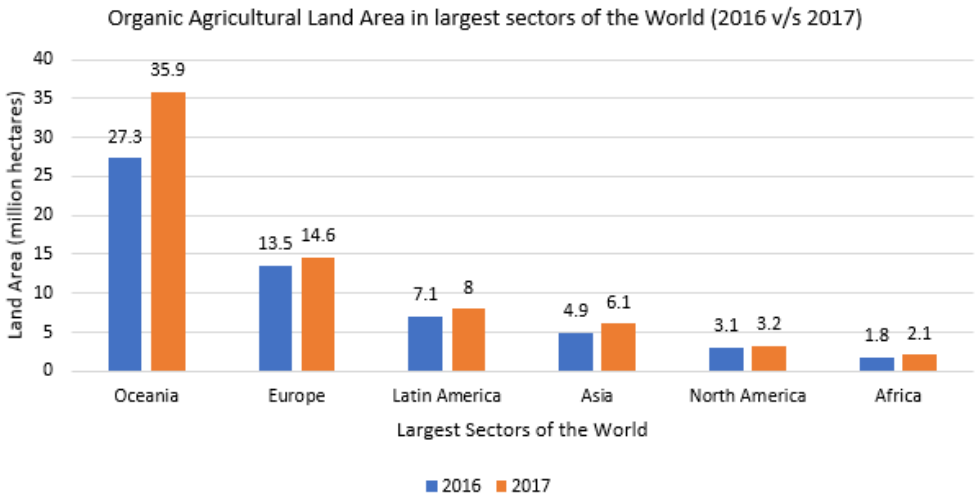


Figure 1: *Increase in organic agricultural land area between 2016 [5] and 2017 [6]*

Figure 1 illustrates the increase in organic agricultural land area in the larger sectors of the world between 2016 and 2017 [6]. Organic agricultural land area increased by 16.67 %, around the globe, between 2016 and 2017.

As of 2017, although there was an increasing trend of organic farming, only 1.4 % of farmland across the world was organic [6]. Hence, it would be appropriate to focus non-organic farming as part of our research.

While organic farmland area is predicted to increase, a large portion of the remaining farmland decreases either due to natural causes e.g., erosion or due to human influences e.g., urbanisation (highways, housing etc). Annually, 75 billion tonnes of soil is lost due to erosion, across the globe [8]. As we have analysed land area values thus far, the soil erosion equates to approximately 3 million hectares (30,000 km^2) of farmland per year [9]. Similarly, on average, 4 million hectares (40,000 km^2) of farmland is annually converted to use for housing, highways, and factories etc [9]. A recent study highlighted the average yearly increase in urban land in the past 30 years to have been 5.8 million hectares (58,000 km^2) in larger sectors such as India, China, and Africa, with North America being the largest [10].

Given the large annual increase in organic agricultural land area, the land area lost to soil erosion and urbanisation would seem minimal; however, in the last 30 years the amount of land lost equates to 140 million hectares (1,400,000 km^2) [9]. Using the data and trends of the last century, it is predicted that due to exponential growth, 152.7 million hectares (1,527,000 km^2) of farmland will be

lost to urbanisation from 2000 to 2030 [10]. This significant loss is nearly equivalent to the land area of Iran.

Although the effect of land loss is impactful, it is not the only component affecting agriculture or agricultural growth. The analysis thus far suggests that agriculture or agricultural techniques suffer from other problems; however, a different view also shows that agriculture or agricultural techniques themselves have negative impacts on the world.

This review focusses on the technological problems more than the problems for which the technology was introduced. Hence, rather than discussing any problem in depth; a range of problems have been discussed on a high level.

Along with loss of land, there is a large impact of emissions due to agriculture. As previously seen in this review, an increase in greenhouse gases was observed due to deforestation. Hence, we will have a high-level focus on Greenhouse Gases for this review. It can be argued that Greenhouse Gases have both advantages and disadvantages so they cannot be classified as extremely harmful. However, in this case, it is important to understand the negative effects, as they can be reduced, hence benefiting agriculture and even global climate.

Climate change is one of the largest effects that Greenhouse Gases impose. Although an advantage is that the gases only let sunlight through, the disadvantage is that the trapped heat is causing the earth to warm up [11]. Common inference suggests that the trapped heat is harming agriculture itself. The gases emitted by agricultural techniques / methods increase the trapped heat in the world, causing many problems such as drought and sea level rise. Briefly analysing the effects of these two problems, it is clear how a drought (lack of water) harms agricultural growth and the sea level rising causes more land (possibly farmland near the ocean) to be submerged under water due to flooding. In-land farms can be affected by flooding due to a rise in sea levels too. Although, this is a brief high-level interpretation of the possible problems, one of the root causes of Greenhouse Gas emissions is agriculture.

A study carried out by the World Resources Institute states that 6 billion tonnes of Greenhouse Gas emissions were recorded in 2011 which accounted for 13 % of the total global emissions. It was the world's second largest emitter [12]. The main reasons of agricultural emissions were noted to be soil carbon, soil respiration, fertilizer, and pesticide, among others. By 2030, it is predicted that the Greenhouse Gas emissions will increase by another 15 % and the only method to reduce this is to change farming practices [12].

A similar study presented data for all countries in the world. Maintaining the main sectors stated in **Figure 1**, **Error! Reference source not found.**, indicates the absolute change in Greenhouse Gas emissions in the main contributors of the world [13].

Table 1: *Absolute change in Greenhouse Gas emissions between 1990 & 2016 in main agricultural sectors of the world [13]*

Country	Greenhouse Gas Emissions (tonnes) (Absolute difference between 1990 and 2016)
Brazil	+162.37 million
China	+142.17 million
India	+137.52 million
United States	+22.66 million
New Zealand	+0.04 million
United Kingdom	-13.05 million
Australia	-63.20 million
World	+766.36 million

Over a span of 30 years, a large increase in Greenhouse Gases was seen in Brazil, China, and India. In contrast, there was a small increase in United States of America (USA) and New Zealand, while Australia and United Kingdom have reduced their emissions. Brazil, China, and India are known to be developing countries, while the others are developed countries. From this, although we cannot conclude that the reduction in emissions is solely due a change in farming methods; we can observe that a large reason behind this is modern farming methods. A study carried out in 2018 verifies this reasoning and summarises the reduction of emissions due to farming into two parts. Firstly, modern farming techniques such as vertical farming reduce the required land area. Hence, less land change results in lesser emissions. Secondly, modern farming techniques such as organic farming reduce the use of chemicals and fertilizers in soil resulting in fewer emissions. This is a significant contributor – "eighty percent or more of the emissions for agriculture happens on the farm" and "reductions will come from how we are managing our soils", crops on the land and chemicals used [14].

The problems highlighted in this review thus far clearly indicate how significant the problems are in the agricultural sector. A solution is needed to solve the problems and it is visible that some areas of the world have been successful in their attempts. The next section of this review explores the current methods that countries are using to mitigate the current agricultural problems.

2.2 Background – Controlled Environment Agriculture to mitigate current problems

In simple terms, Controlled Environment Agriculture (CEA) can be described as farming of crop in an environment wherein key variables required for plant growth, e.g., light, water, nutrients etc, are monitored and controlled, hence providing the plant the optimum growing conditions.

For the scope of this review, we will focus on two types of Controlled Environment Agriculture: vertical farming and greenhouses. Vertical farming is "based on multi-level factory design" [15] also known as stacked farming. Plants are provided with nutrients while being suspended in a soilless medium. As seen in **Section 2.1**, some countries had a decreasing greenhouse gas emissions rate in the last 30 years. A reason for this could be that these countries, e.g., USA, Singapore, Japan, and Korea, were the first countries to use vertical farming. Although being first doesn't justify the decreasing emissions rate, it validates countries getting more time, experience, and experimentation in improving the farming techniques, as compared to countries who implemented vertical farming much later [15]. In support of this conclusion, vertical farming was also described as "a movement in the society to reduce greenhouse gas emission" [15].

Hydroponics and Aeroponics are common vertical farming methods. Hydroponics methods include Aquaponics, Nutrient Film Technique (NFT), and Ebb & Flow techniques. Simple hydroponics can be used in vertical farming, wherein the roots of the plant are immersed in water and nutrients are added to the water after certain intervals. Aquaponics is an advanced method of hydroponics and can be defined as "recirculating aquaculture and hydroponics" [16], or in simple terms, it is the farming method in which water in a fish tank is mixed with nutrients and is circulated to the plant roots in a separate tank. The process is completed with recycling the used water back into the fish tank. **Figure 2**, below, illustrates the process of Aquaponics.

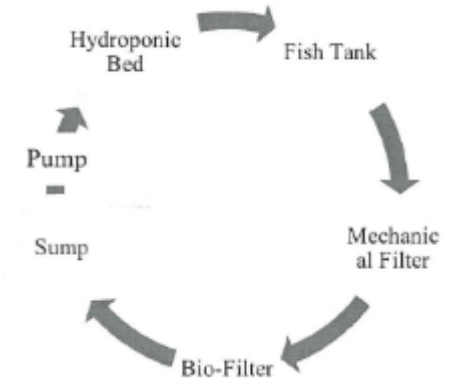


Figure 2: *Aquaponics Process [16]*

NFT can be used with Aquaponics too, hence building a more complex hydroponics method. In this method, the roots of the plant are partially immersed in narrow channels or paths and a thin layer water is streamed past each plant's roots. A calculated portion of nutrients that plants require, are dissolved, and mixed in this water [17]. **Figure 3**, below, shows an implementation of NFT.

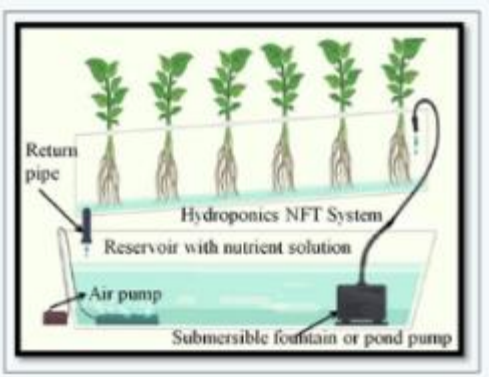


Figure 3: *Diagram of NFT setup [19]*

Deep Water Culture is another variant in hydroponics. In this, water, that contains nutrient, is supplied to the roots. These roots are "always submerged in water and oxygen." This method requires less maintenance and less fertilizer as the system is always heavily enriched with oxygen [18]. **Figure 4**, below, shows an illustration of an implementation of Deep Water Culture.

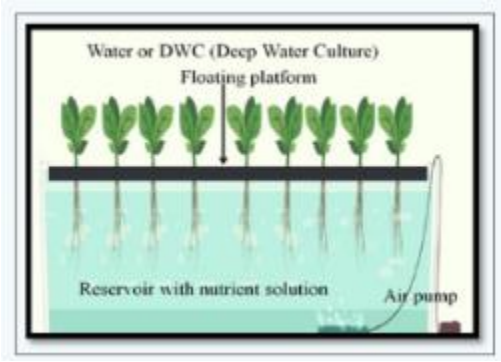


Figure 4: *Diagram of Deep Water Culture setup [19]*

Ebb & Flow is also a type of hydroponic methodology that is inexpensive to setup. Two containers are used wherein one contains the crop, while the other container is filled with nutrient solution and is placed directly beneath the first container. A pump is placed inside the bottom container which intermittently pumps the nutrient solution into the top container. The solution in the top container is "gradually siphoned out via the pump line" when the pump is switched off. The time required in siphoning the solution out allows oxygen to reach roots. The timing of the pump can vary. For example, in some cases the pump is switched on for 30 minutes and then switched off for 15 minutes and then the cycle is repeated [19]. **Figure 5**, below, shows an illustration of an implementation of an Ebb & Flow technique.

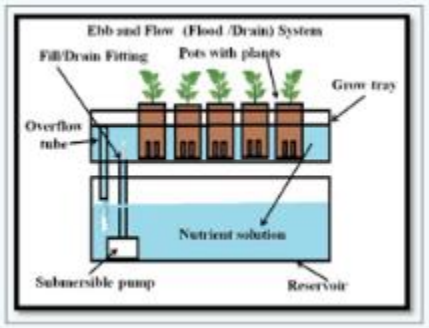


Figure 5: *Diagram of Ebb & Flow setup [19]*

Aeroponics is a similar method to hydroponics; however, rather than roots being submerged in water, the roots of the plants are sprayed with "atomised nutrient solution" [20], i.e., a mist of nutrient enriched water. Other technological aspects are like that of hydroponics i.e., controlled and monitored environment. However, studies have indicated that aeroponics may be able to better "sustain hyper growth" [21] as the roots are sprayed water using "pressure nozzles or foggers" [21] rather than being immersed in water for long periods. **Figure 6**, shows an illustration of an implementation of Aeroponics.

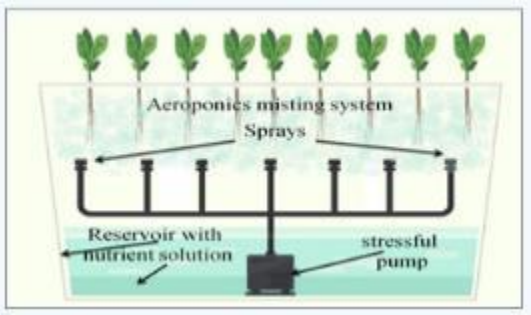


Figure 6: *Diagram of Aeroponics setup [19]*

Greenhouse agriculture, like vertical farming, is also a common CEA method. Although, it is indoor farming, the plants are horizontally laid out rather than vertically stacked. Hence, space required is the clear difference between both types of controlled agriculture methods. While vertical farming can focus on the use of artificial lighting, greenhouse farming is based on the use of natural lighting, as the roofs of the indoor farms are commonly seen to be transparent [22]. Greenhouses can have various control methodologies implemented for different environmental conditions. For example, in colder regions, an "active climate control", high technology system can be implemented i.e., a system with stronger monitoring and control systems in place. This presents higher yields and good quality all year, however, consequently, it endures higher costs [22]. In warmer regions, a "passive climate control", low-medium technology is implemented, which gives good quality yields in limited periods i.e., irregular production, however, it has lower running costs [22].

Both vertical farming and greenhouse agriculture have their advantages and disadvantages, some of which are summarised in Table 2, below. Although both methods of Controlled Environment Agriculture are beneficial, given the statistics of urbanisation in **Section 2.1**, it may be argued that there will be demand for vertical farming more than greenhouses due to its space saving nature. This will also prove to be beneficial considering high population growth. Recent data and predictions state that in the next 30 years, the population will grow by 2 billion people and by 2100 we could peak at a population of 11 billion. That would be approximately four billion more than the current population [23]. This also further validates the need in the future for vertical farming more than greenhouses.

Table 2: *Advantages & Disadvantages of different types of Controlled Environment Agriculture*

Type of Controlled Environment Agriculture	Advantages	Disadvantages
Greenhouses	• Longer growing seasons due to controlled climate [24]. • Crops can be grown out of season [24]. • Variety of crops can be grown [24]. • Faster growth and higher yields obtained [24].	• Larger area required, hence, not ideal for urban areas. • Costlier to set up [25]. • Increased running costs e.g., water / electricity [25].
Vertical Farming [26]	• Increased production & availability in crops. • Conservation & appropriate recycling of water etc. • Less land use, i.e., environmentally friendly.	• Non-uniform farming practices due to different weather patterns over the world. • Lack of crop varieties suitable for vertical farming. • Lack of knowledge and skills required for vertical farming practices.

Conventional farming requires the use of large quantities of fresh water and fertilisers with marginal returns, while Controlled Environment Agriculture (CEA) enables higher yields with reduced water usage (water is recycled) [17]. However, it is clear to state that the reduction in conventional farming methodologies affects naturally produced crops (e.g., using natural sunlight) as well as biodiversity (animal and plant living in a natural way) [27]. While CEA cannot support a variety crop growth as compared to conventional farming, it still provides longer seasons and increased production, while also enabling farming in urbanised areas. In these regards, it is fair to conclude that CEA is helping save global agriculture.

In the future, using artificial resources and controlled environmental agriculture, crops may be tested and grown in space, however, Controlled Environment Agriculture solutions have only recently been introduced and consequently, there are aspects that can be developed to enhance the performance and outputs. A study carried out in 2016 used a sample crop, Lettuce, and compared vertical farming output (Controlled Environment Agriculture) against conventional horizontal farming outputs. The results indicated that vertical farming had a greater output / efficiency as it yielded "13.8 times more crop" as compared to conventional farming [28]. This included a ratio of yield to "occupied growing floor area" too. Hence, this research also proved the importance and benefits of vertical farming. On this basis, the following section reviews the current technological / engineering barriers with controlled environmental agriculture solutions.

2.3 Technological barriers with current Controlled Environmental Agriculture solutions

'Smart Farming' is another commonly used term for automated controlled environmental agriculture. A smart farm system would be made of many sub-systems. For example, a farm that is cultivating lettuce will use one of the vertical farm methods as a basic setup of the crop. For now, let us consider the Deep Water Culture setup. The roots of the lettuce would be fully submerged in water that contains nutrients. The water level and temperature, nutrient levels etc. are continuously monitored using sensors such as temperature sensor etc. The crop setup is given the required lighting using artificial lights such as LEDs.

These sensors are all connected to a computer system that analyses and calculates data using the readings. Consequently, these results and pre-set values are used to actuate other hardware such as a pump to pump water into the tank etc. An example of a control system to be used for this is PLC (Programmable Logic Control). There are more sub-systems to the smart farm and to better understand these, we will split smart farming into three different processes, see **Figure 7**, below.

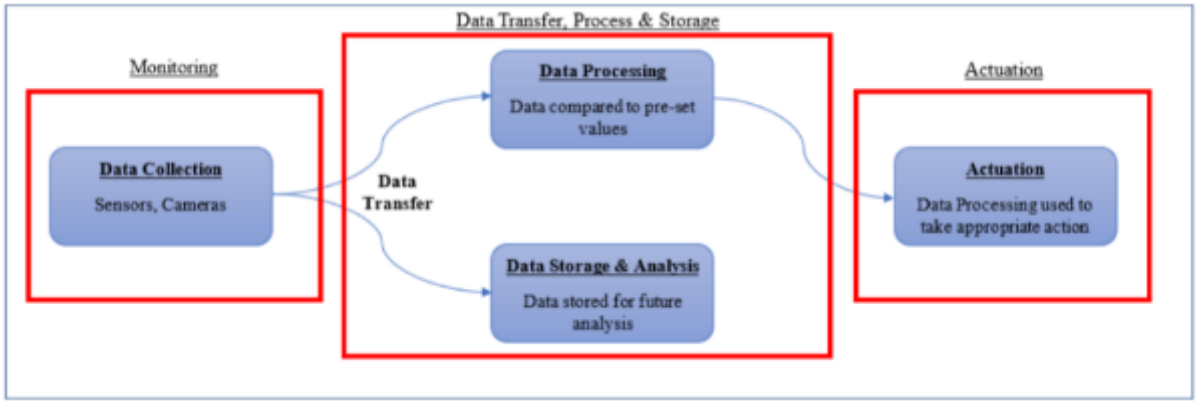


Figure 7: *Different Processes of Automated CEA [15]*

The first process is monitoring of the plant and its environment. This includes monitoring of key plant growth variables such as temperature, humidity, light, carbon dioxide and water quantity [29]. Plant (leaf) colour has also been a monitored variable in some studies.

Wireless and wired sensors are equally used in the monitoring of these variables. As the technology improves and more features are available in vertical farming, using a completely wireless system has been an increasing trend [30]. Large companies, that use vertical farming methodologies, such as 'AeroFarms' (founded in 2004) have implemented this monitoring system using IoT (Internet of Things) integration [31], while others such as 'Gotham Greens' (founded in 2008) [32], and 'Plenty' (founded in 2014) [33] have preferred partial integration of IoT elements.

Along with sensors, computer vision has been used to monitor and identify crops that are well grown or dead. A study carried out in Florida used a camera system to detect if crops "are ready to be harvested or removed" [34]. Their webcam had two camera lenses that captured one image each, simultaneously. Any crop between the common portion of the two images was captured by both camera lenses, hence providing a clearer image of the crop in sight. **Figure 8**, below, is an image of the camera with a stripped front to remove excess material.

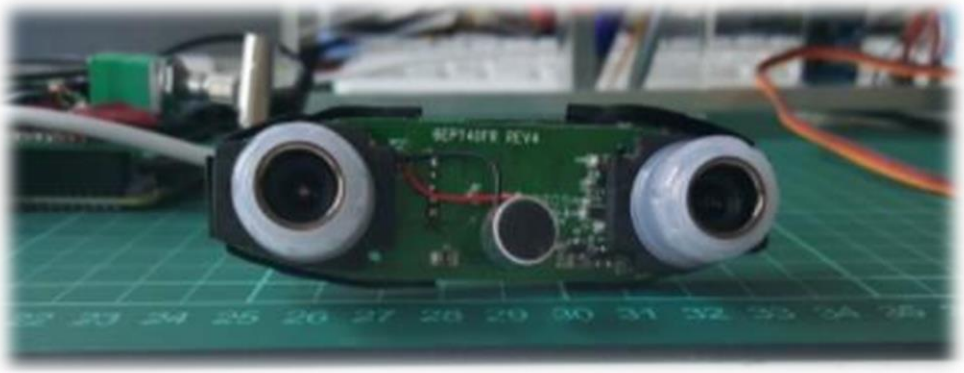


Figure 8: *Dual Lens Camera (stripped front to remove excess material) [34]*

The second process in the system is Data transfer, data process, and data storage (this can be noted as a group process as part of system integration i.e., the integration of multiple data and information). Large and modern smart farming has mostly been implemented using wireless systems. Data is transferred from wireless sensors to a cloud-based system. A conference carried out in 2019 showed that the most common methodology for data transfer was uploading data to a 'ThingSpeak' cloud using a wireless module such as the 'Intel Edison' [35]. This statement matched a conclusion that a study in 2017 had stated. 'ThingSpeak' was used as the cloud during that study too and the communication tools were a wireless module and a microcontroller [36]. The study further concluded that although large-scale farms were wireless, the prototypes were always wired. A cloud medium for storage, and data transfer using a wireless module and microcontroller are the frequently used methodologies of data transfer and storage.

For data transfer and module communications, various types of networks are implemented to make data transfer a faster process. “Low-Power Wide Area Networks (LP-WANs) are used for “long range communication at low data range”, while “Ultra-Low-Power Personal Area Networks (ULP-PANs) are used for “short range communications at fairly high data rate” [37].

For processing data, several methodologies have been adopted. Web user interfaces have been developed for the worker to analyse data and use the interface to appropriately change modes of control signals [36]. Android applications have also been adopted by vertical farming companies, wherein the user can monitor, analyse, and send control signals using the Android application’s user interface [38].

Whichever technology or methodology is used, an alarm notification is always sent to the user through SMS, web, or phone applications when the monitored data values exceed or drop below the pre-set parameters. This is one of the most essential and commonly used process.

The final process in the whole system is the actuation of hardware depending on the monitored data. Three different aspects have been trialled. Hardware such LED lights, fans etc, are activated by a user after he/she manually analyses the monitored data results – this is the first aspect and is very commonly used. The second aspect is partially automated wherein, the user activates the hardware and then the hardware continues at a regular interval. For example, in an Aeroponic system, a pump is used to spray nutrient solution at high pressure. Once the user activates this pump, the nutrient solution is pumped at regular intervals for a certain period using a digital timer [39]. The third aspect, used mainly for large scale vertical farming, incorporates many racks of crops that are monitored and controlled by “robot appliances” – i.e., the data from the sensor is monitored and analysed by a “robotic appliance” and consequently the robot activates the relevant hardware to bring readings back to pre-set values [40].

The highly complex and technical features of smart farming consequently bring barriers and issues that are a hindrance to advancements made in this field. Considering the complexity, the second process that we have highlighted – transferring, storing, and processing data – “comes with inherent security risks and vulnerabilities” [41]. The use of IoT is one of the main reasons that the agricultural sector gets exposed to cyber security threats. Risks of these threats can be large scale such as datasets being used to harm the agricultural industry and hence the economy of a whole country. Similarly, on a smaller scale, the datasets can be manipulated, consequently affecting real time decisions of automated portions of farms, which in turn can ruin batches of crops [42].

Processes 1 and 3, that we have defined above, also have problems and issues; however, the problems have not been identified, understood, and resolved in great depth. This is primarily because these processes have not been extensively tested. There are two major reasons behind the lack of testing: implementation & maintenance cost and lack of knowledge. A smart farming related study carried out in 2017 stated that most challenges can be solved, if “enough business opportunities … in smart farms can be created” – i.e., smart farms are not explored in enough detail [43].

As there are many hardware and software parts to a smart farm, the implementation and maintenance cost is very high. This cost is higher when smart farming is implemented using IoT. Consequently, farmers avoid smart farming. Some conferences and studies have similarly concluded that the sensing & monitoring process is detailed and convoluted, hence having more technical problems; however, a larger problem is the costs related to it [43]. A study completed in 2018 concluded that while labour costs have reduced, not all farmers implement this system due to the high setup and ongoing costs [44]. From this, a clear divide is visible between small scale farmers, who are unable to afford the initial and ongoing expenses of a smart farm, and large smart farming companies, as named previously, who have made many advancements in this field.

Along with costs, the other reason many farmers avoid the technologies in smart farming is their lack of knowledge of how the technology works or in some cases their lack of knowledge of the technology itself. Setting up everything with monitoring facilities requires "software expertise" which is where many farmers lack [45]. A study carried out in precision agriculture stated that farmers in rural areas are not familiar with new technologies and consequently, a slight hardware issue that cannot be fixed quickly, thus resulting in catastrophic damage to the plant and potentially the whole production [46]. Therefore, the growth "has not been as robust as was expected earlier" [46]. Another study carried out focussing on IoT also reached a similar conclusion and stated that by "continuing with traditional and orthodox methods" they are limiting their ability in upskilling and upgrading their farms [47].

Therefore, smart farming technology is expensive and farmers with smaller farms cannot use it due to high implementation cost. At the same time, "limited knowledge and skills" limits the farmers' abilities in adopting smart farming technologies [48]. A survey carried out in Ireland in 2019 supported the conclusions drawn above. **Figure 9**, below, highlights the details.

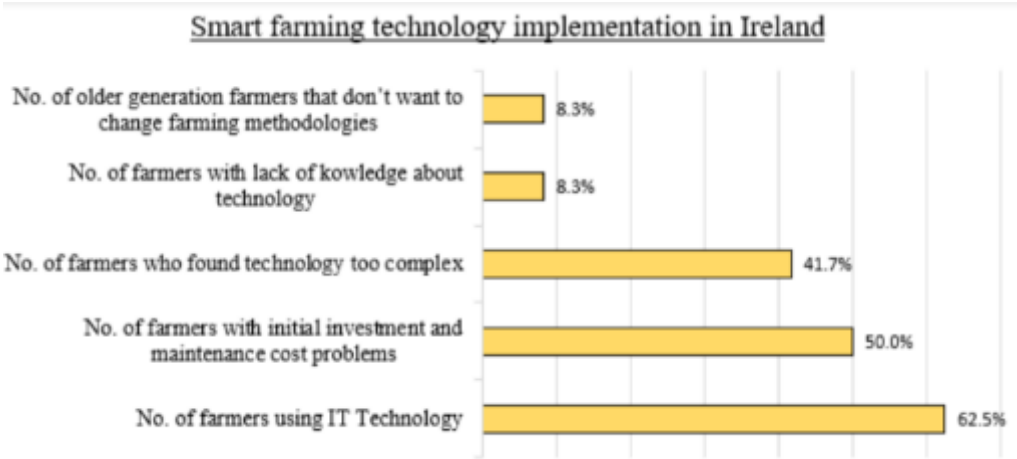


Figure 9: *Smart Farming in Ireland - Implementations of Technology [87]*

Although Ireland is a small sample size compared to other large countries such as the United States of America and Australia etc, the results of the survey give a clear depiction of the trends of smart technology implementation. The low number of farmers using IT technology (62.5 %) represents the key problem, while the rest of the categories represent the reasons as to why this is a problem. This data, although small in sample, gives the impression that technology used in smart farming is only made for farmers who are based in well developed areas.

Therefore, one of the technological barriers with smart farming is cyber security, which will provide major issues if it is not researched and resolved in greater depth. Although there are problems with sensing, monitoring, and actuating, those problems are unknown to some extent. This is due to the largest problem with smart farming technology: the lack of its implementation. High implementation and maintenance costs and farmers' lack of knowledge are key reasons to the limited use of this technology. This has meant that automation in vertical farming has not grown as was expected. **Figure 10**, below, details the level of automation currently in vertical farming.

Figure 10: *Current Level of Automation in vertical farming (dashed line represents current level) [88]*

Due to the slow growth and hence lack of implementation of technology, the barriers related to sensing, monitoring, and actuating are discovered to a limited extent. **This is a gap in knowledge of smart farming.** It is believed that more testing and usage of this technology, at small or large scale, would aid in making advancements in this field.

2.4 Monitoring and Sensors

Process one, as described in **Figure 7**, involves monitoring plant growth. To effectively ensure that a crop is stored in a supportive environment, all necessary variables are continuously monitored using sensors. Although there are different methods of smart farming, the variables that are monitored are constant.

For ideal plant growth, it is necessary to monitor temperature, humidity, light, water quantity, moisture etc. A study carried out prior to the introduction of smart farming, plant growth under controlled in 1944, stated the use of similar factors such as temperature and water level etc [49]. These factors have remained consistent over the last 75 years. A study using a microcontroller based vertical farming automation system, carried out in 2018, also supported the use of the same monitored factors [50].

As plant growth is heavily dependent on light, the BH1750FVI sensor is an example of a light intensity sensor used in monitoring of light intensity. An OP-AMP and Photo Diode integrated in this sensor record values of current and convert to voltage [51]. An ADC (Analogue to Digital Converter) is used to produce digital values which are processed through a Logic+I2C unit where "illuminance values are converted to Lux and I2C communication" for data transfer takes place. The sensor, illustrated in **Figure 11** below, can be operated in three modes [51]:

- H-Resolution Mode2 which has a 0.5 lx resolution and requires 120 milliseconds for measurement.
- H-Resolution Mode which has a 1.0 lx resolution and requires 120 milliseconds for measurement. This is the best mode as it is effective in the dark and can easily reject noise.
- L-Resolution has a 4.0 lx resolution and requires 16 milliseconds for measurement.

Figure 11: *BH1750FVI Sensor [89]*

Industrial computerised vertical farming companies have verified the use of this sensor and a study carried out in 2019 emphasised similar advantages of using this sensor as it is insensitive to background light [52]. Other light sensors such as the VEML7700 have also been used for smart farming however, it is a low-cost sensor, consequently not having highly accurate results [53].

Temperature is also important to plant growth. This is split into two parts. The ambient temperature of the crop, and the temperature of the water in which the plant is submerged (partially or fully).

The DHT11 sensor, part of the DHTXX series of sensors, is a humidity and temperature sensor and typically used to monitor ambient temperature. This sensor unit is made up of three main components. It has a resistive type of humidity sensor, a NTC (negative temperature coefficient) thermistor is used to measure temperature and an 8-bit ADC is used for data type conversion [54].

Figure 12, below, illustrates a DHT11 sensor. Using this sensor, the temperature of the surroundings of the crop can be monitored and controlled. Using the results from the sensor, electric fans / heaters can be used to cool / warm up the plant and ensure crop growth is constant and ideal.

Another smart farming study in conducted Malaysia have come to similar conclusions regarding this sensor and have also recommended this sensor for use with IoT for better results [55].

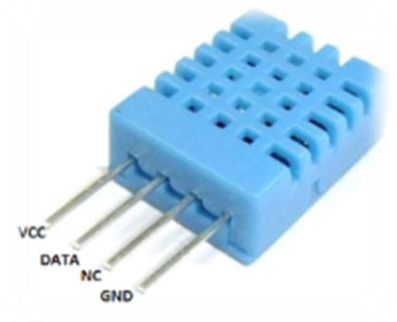


Figure 12: *DHT11 Sensor [90]*

The DS18B20 temperature sensor has two variants. The first is a "bare sensor in a standard transistor casing", while the other is a "probe with the sensor encased in a waterproof metal sleeve" [56]. The waterproof nature of this sensor results in the advantage of using it to monitor the water temperature in which the crop is submerged. This temperature needs to be ideal as the nutrient solution is also mixed with the water. Using the data from the sensor, cooler or hotter water can be regulated into the tank. This process is sometimes carried out at the same time as refilling nutrient solution. These sensors are "1-wire sensors" that have an error range of $\pm 0.5°C$. The sensor is known to have an ideal sensing range up to 100 $°C$ but can be used sparingly up to 120 $°C$ [57]. **Figure 13**, below, is an example of the DS18B20 sensor.

Figure 13: *DS18B20 Waterproof Temperature Sensor [57]*

Other weather/ humidity sensors such as the BME280 have also been used for smart farming however, it is low-cost sensor, consequently not having highly accurate results [53].

As some smart farming solutions involve the method of submerging plant roots in a tank full of water, Ultrasonic sensors, such as the HC-SR04 sensor, are used to monitor the water level [53]. The sensor is situated at the top of the tank and can measure its distance from the water surface. Given a pre-known distance between the sensor and the bottom of the tank, it is easy to calculate the water level.

2.5 System Integration

Process two in the system, as described in **Figure 7**, involves data processing, data storage and data transfer – i.e., the system integration. To effectively ensure that a crop is stored in a supportive environment, all necessary variables are continuously monitored using sensors and controlled using actuation components. Currently in the Controlled Environment Agriculture (CEA) industry, the actuation is primarily done manually requiring human intervention. Therefore, any results or data obtained from the sensors e.g., temperature, moisture, humidity etc, are either manually collated and analysed or in recent times, vertical farms have been implemented in a 'smart' form wherein the results or data is collated using software programming and presented to the user on websites, mobile applications in the form of graphs and tables. The users then use the data to manually carry out actuation tasks, e.g., turning lights on/off.

As part of this, microcontrollers are used to receive information from and send signals to components. The microcontrollers are connected to other modules e.g., Wi-Fi modules, or directly to a Wi-Fi cloud to form the means of data transfer and data storage. The microcontrollers require programming as well as the website or mobile application. Together, this forms the framework of the system. In recent case studies, various microcontrollers, applications, and programming languages have been used to form the frameworks.

In a CEA water level monitoring study carried out in 2020, the experimented methodology involved using a Raspberry Pi 3 Model B microcontroller, see **Figure 14**. All sensors and components were connected to the Raspberry Pi microcontroller. This was a low complexity study, controlling the water level based on the sensor readings obtained. Therefore, instead of implementing a website or application to allow users to analyse data, an automated approach was used wherein the microcontroller was programmed to use the sensor readings to control a motor that was connected to a lever that allowed water to flow in and out [58], hence controlling water level.

Figure 14: *Raspberry Pi 3 Model B Microcontroller [93]*

A similar study was carried out in 2022 [59], with an increased scope of developing an IoT based vertical farming solution. The project had a similar set up in which they used a Raspberry Pi 3 microcontroller, as it was a "low-cost implementation", which was programmed using Python programming language. The data received from the sensors was stored in an SQL database in a cloud storage medium and was accessed manually by the users and consequently any required actuation tasks were carried out. Another project completed for automated field-based tent systems for CEA & experimentation also used a Raspberry Pi microcontroller, albeit an upgraded Raspberry Pi 4 model B [60]. The Raspberry Pi microcontroller is a "cost-effective, efficient, versatile hardware solution" and "each model has improved memory and processing capabilities from the previous generation."

A study in 2022, proposed a combined approach in their model in which they used an IoT server that was the base controller of the system i.e., receiving and storing data while also allowing the user to analyse the data. A Raspberry Pi 4 microcontroller was used to connect the components and transfer data to the IoT server. The microcontroller was described as very suitable for the project, as it "is a low-cost, low-power Wi-Fi microchip with full TCP/IP stack and microcontroller capabilities" [61].

While Raspberry Pi microcontrollers are commonly used in the CEA industry, Arduino microcontrollers are equivalently commonly used. An automated IoT enabled vertical farming study carried out in 2023 explained their use of an Arduino microcontroller, see **Figure 15**, to "log the data coming from the sensors" [62]. The data was then transferred and stored in a database and the whole system was set up on a website that allowed the user to remotely monitor the data readings online.

Figure 15: *Arduino Uno Microcontroller [94]*

In an alternate study carried out in 2020, a combination of an ESP8266 microcontroller (for connecting components e.g., sensors, LEDs etc,) and a Raspberry Pi microcontroller (for connecting to the cloud for data storage and allowing user control with web dashboard) were used [63]. It was noted that this project was more dependent on the ESP8266 microcontroller, see below, as it had more functionality than the Raspberry Pi microcontroller, which was idle, waiting for new data, once the user control action was complete [63].

Figure 16: *ESP8266 Microcontroller [95]*

From the findings detailed above, it was found that the Raspberry Pi microcontroller is more commonly used in the CEA industry. The Arduino Uno and ESP8266 microcontrollers are also suitable for CEA and at times the framework implemented has included multiple microcontrollers to undertake two different functionalities. The table below, highlights the advantages and disadvantages of the three microcontrollers to better illustrate which is most suitable for any similar set up.

Table 3: *Advantages and Disadvantages of different Microcontrollers*

Microcontroller	Advantages	Disadvantages
Raspberry Pi	• Supports an operating system – i.e., can work in the form of a small-scaled computer [64]. • Large number of GPIO Pins [64]. • As it has a CPU, it supports Ethernet and Wi-Fi connections [64].	• Its "single-board computer (SBC) cannot be customised. [64]. • Requires a micro-SD card as it has no built-in storage. [64]. • Can overheat when carrying out large, complex tasks. [64].
Arduino Uno	• Hardware and software are open-source i.e., online ready-to-use code files can be directly implemented [64]. • Less expensive than Raspberry Pi [64].	• Arduino does not have built-in "internet support" [64]. • "Very less processing power" than Raspberry Pi [64]. • Cannot complete complex tasks using Arduino [64].
ESP8266	• Can be programmed using the Arduino IDE programming environment [65]. • Built-in Wi-Fi module [65]. • Less expensive than Raspberry Pi [65].	• Less GPIO pins than Raspberry Pi [65]. • Lower processing power than Raspberry Pi. • Limited memory – complex tasks can be difficult to complete.

Although all microcontrollers are fit for purpose, from the table above, it is evident that the Raspberry Pi microcontroller is most suitable for a CEA set up and experimentation. Two microcontrollers have been used in previous studies to individually replicate one user end and one as a controller for components. While one Raspberry Pi microcontroller would be sufficient to carry out all necessary tasks, there may be scope during the experimentation phase of this project to explore the option of implementing two Raspberry Pi microcontrollers to replicate the same functionality as mentioned above.

From this research, it is also evident that most CEA industry practices include a phase of human intervention, either in analysing the data or in the implementation and actuation functionality while the plant is running. While few projects have attempted to solve this, it is not common and can be seen as

a missing element in the current CEA practices and implementation techniques. This can be linked to the technological barriers or gaps previously discussed in this thesis.

2.6 Actuation

Process three in the system, as described in **Figure 7** previously, involves the actuation phase. To effectively ensure that a crop is stored in a supportive environment, all necessary variables are continuously monitored as mentioned above; however, to maintain the thresholds required for any variables, components need to be actuated. Currently in the Controlled Environment Agriculture (CEA) industry, the actuation is primarily done manually requiring human intervention. The main components that are used in this third process, currently implemented in the industry, are heating and cooling elements (for temperature control) and lighting elements (for lighting simulation and growth of crop).

For the research aspect of this project, the industrial CEA study can be split into large scale and small implementations. A study carried out in 2022 to improve the environmental performance of urban vertical farming detailed a large-scale implementation of a vertical farm which included temperature and lighting control. "The air in the facility was considered to be constantly conditioned … the temperature is kept at 25 ℃" [66]. The air temperature and quality was maintained using an air handling unit through which air was passed using fans and the "heat recovery" was done using a "water-cooled chiller" [66]. The lighting was cost-effective and based on the average light intensity plants receive ($216\ \frac{\mu mol}{m^2} - s$ for 16 hours), "light emitting diodes were used to provide adequate photosynthetically active radiation for plant growth according to the plant's growth stage".

Another study researching large-scale implementations confirmed the use of large heating and cooling elements to control variables for full plant systems. Electric heaters are commonly used in the industry and are "operated via a thermostat or an automatic timer" to heat the environment and maintain the required temperature level [67]. Other studies have confirmed the use of Heating, Ventilation, and Air Conditioning (HVAC) systems for temperature control in their large-scale controlled environment along with LEDs as their lighting source instead of sunlight [68]. **Figure 17** below, is an example of a large air conditioning unit used in Controlled Environment Agriculture to control and maintain temperature and carry out humidification and dehumidification functions in a large facility .

Figure 17: *Example air conditioning unit used in large-scale CEA implementations [96]*

Lighting in the CEA is very important for various reasons including the sole purpose of illumination as well as an additional factor in plant growth with the use of Ultra-violet (UV) LEDs. A study carried out in 2016 about LED lighting for urban agriculture confirms the shift in the CEA industry with even large-scale manufacturers moving towards the use of LEDs for lighting. Previously, "commonly used artificial lighting sources included High Pressure Sodium Lamps and Fluorescent Light Tubes". "In recent years, LEDs have gained significant amount of attention in the research community, CEA industry, and lighting device manufacturers" [69]. An example of this shift, confirmed in the study, in Japan most CEA plant factories were initially (since 1990) equipped with fluorescent lighting and "now, as of 2013, nearly half are using LEDS" – approximately 20 % are also noted to be using LED grow lights [69].

For the basis of our project, small scale implementations and their methodologies are more applicable. A project carried out for a new device for CEA in 2017 detailed the components for heating which included a 12 V, 50 W, Electric Ceramic Thermostatic PTC Heating Element – like the element shown in **Figure 18**. The study also included a 12 V, 1 A, DC Blower Fan (like the element shown in **Figure 19**) and included LED grow light modules for lighting. These components were most suitable as the controlled environment was manufactured in the project to be operatable "on a desk in an indoor research environment (e.g., classroom)" etc [70]. This was considered a similar sized study to this project and several other similar projects have also used similar ceramic heating elements.

Figure 18: *12 V, 150 W Electric Ceramic Thermostatic PTC Heating Element [97]*

Figure 19: *Similar element to 12 V, 1 A, DC Fan [98]*

Although large-scale implementations were studied in this section to understand current industry equipment; for our project, like the figures above, smaller components will replicate the same functionality as the large equipment. For small scale CEA implementations, it is evident that the heating and cooling elements like above, and LEDs for lighting are the most suitable.

During the research of current implementations and studies, it was found that most of the small-scale studies and implementations focused on humidity, light intensity, and plant growth as their key elements i.e., their main objective was observed to be more towards monitoring variables and only in some cases towards controlling variables. A study carried out in 2019 focused on monitoring variables such as humidity, light, and water level [71]. The study also monitored temperature; however, temperature control was not part of the scope. An IoT implementation in 2015 was also focussed on monitoring humidity, carbon dioxide, and temperature – once again the scope did not involve temperature control [72]. These are a few examples of the current industry – many other studies have similar scopes.

From this research, it is evident that most small-scale CEA studies lack information and data around the implementation and results of temperature control. Hence, linking this to the barriers and gaps discussed above, we can also conclude that, as part of the actuation process, temperature control is a gap or an element of CEA that has not been explored and implemented in great depth.

2.7 Literature Review Summary

Farming shifted from traditional farming towards organic farming resulting in requiring land resources which were obtained from deforestation – causing greater emissions such as greenhouse gases etc. Emissions are negatively affecting the climate. To mitigate this, Controlled Environment Agriculture, described as farming of crop in an environment wherein key variables required for plant growth are monitored and controlled, was introduced. Vertical Farming and Greenhouses are two main types of CEA implemented worldwide. Hydroponics and Aeroponics are common vertical farming methods. There are various types of Hydroponics methods used in the industry. As the name suggests, the focus of vertical farming is more towards farming 'upwards' i.e., vertically stacking in comparison to greenhouses where plants are laid out horizontally. Both methods present advantages and disadvantages, discussed in previous sections; however, vertical farming may be a more optimum solution considering the space saving advantage that counters the problems caused by urbanisation.

'Smart Farming' is a commonly used term for automated controlled environmental agriculture. The framework of smart farming can be split into three processes which were detailed in the sections above. Noting the complexity of any 'smart' or technological farm implementation, barriers and issues are a hindrance to advancements in this field. The slow shift towards smart farming means there is a lack of trialling and testing, and this is due to two reasons – implementation & maintenance cost and lack of knowledge. The high cost creates a divide between small scale farmers and large smart farming companies. Along with this, once farms are 'smart' they are vulnerable to security and cyber threats which can lead to many harmful results for the agricultural industry. **Therefore, one of the technological barriers with smart farming is cyber security. While high implementation & maintenance costs and lack of knowledge are not directly technological barriers, they are barriers caused due to technological implementations.**

For plant growth, monitoring key variables is very necessary including temperature and light, among others. Various types of sensors are noted to be implemented in current industry smart farming projects and studies. The key sensors found were the BH1750FVI sensor for luminance and the DS18B20 temperature sensor which has waterproof (for water temperature) and non-waterproof (for ambient temperature) versions.

The smart farming system is integrated together using microcontrollers and different methods of implementation. While a web or mobile application is a common means of presenting data for the user, cloud or database storage is a common method for data storage. Currently in the CEA industry, the data is mostly transmitted / transferred using an internet connection. While many studies have implemented multiple microcontrollers, some have implemented only one microcontroller with IoT support used on the other end. After analysing the advantages and disadvantages of microcontrollers, the Raspberry Pi is most advantageous as it can even work in the form of a small-scaled computer, thus

allowing it to perform tasks that are more complex than other microcontrollers. While one Raspberry Pi microcontroller would be sufficient to carry out all necessary tasks, there may be scope during the experimentation phase of this project to explore the option of implementing two Raspberry Pi microcontrollers to replicate the same functionality as having a user end and the smart farm end. **From the research, it is evident that most CEA industry practices include a phase of human intervention, either in analysing the data or in the implementation and actuation functionality while the plant is running. While few projects have attempted to solve this, it is not common and can be seen as a missing element or a gap in the current CEA practices and implementation techniques.**

The final process in a smart farming system is using actuators to maintain thresholds for any variables. The main elements that were researched were lighting, heating, and cooling. While large-scale projects were researched, for the scope of our project, small-scale projects were more relevant. LEDs, DC cooling fans and small-sized heating elements like ceramic heaters were most appropriate for projects like this. From the research carried out, it was evident that most small-scale CEA studies lack information and data around the implementation and results of temperature control. **Hence, as part of the actuation process, temperature control is a gap or an element of CEA that has not been explored and implemented in great depth.**

3.0 Methodology

The experiment design was intended to address the technological barriers discussed earlier. This chapter describes the various subsystems in the final solution used for experimentation. It also details the mechanical, electrical, and software setup, as well as the development of the PID controller. The total system integration and the methodology used for the final experimentation are also discussed.

3.1 System Overview

A controlled environment was designed and developed, and an experimental setup was prepared to compare and evaluate the practical solution with model predicted results. This provided the confidence and further established the possibility of implementing the system as a potential solution to the discussed technological barriers in the Controlled Environment Agriculture sector. The solution had several sub-systems, illustrated in the **Figure 20** below:

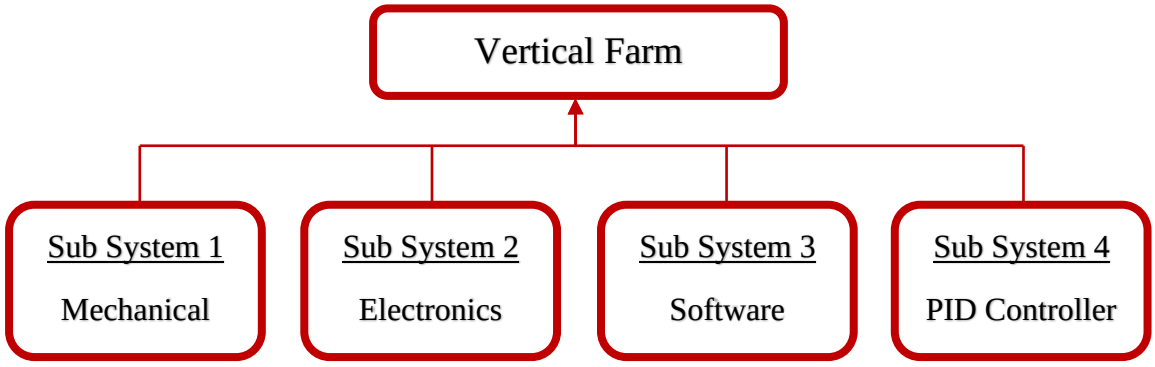


Figure 20: *Vertical Farm and corresponding sub-systems*

For the basis of the experimentation, a NFT (Nutrient Film Technique) was implemented as it is a common Hydroponic technique used in industry (explained in **Section 2.2**).

The system was designed to incorporate an existing 'small-scale' Hydroponics system along with heating, cooling, and sensing elements (**Mechanical**). These components were integrated and connected through a printed circuit board to a micro-controller (**Electrical**) and a PID controller was developed to continuously modulate and control the relevant variables (**PID Controller**).

While the software in the model used the PID controller's outputs to actuate the elements; on the user end, a light website was implemented which wirelessly communicated the user-input to the model (**Software**). This was integrated to build the controlled environment. The sub-systems and their detailed methodologies as well as the final integration are further explained in the following sub-sections of this thesis.

3.2 Mechanical Setup

NFT (Nutrient Film Technique) is a common Hydroponic technique used in the Controlled Environment Agriculture sector, as discussed in **Section 2.2**. Hence, to maintain a connection between the model and commonly used industry techniques, an 'off-the-shelf' DIY NFT system was purchased and used from a local trades store (Bunnings Warehouse), see **Figure 21**.

Figure 21: *Egmont Hydro Kit [91]*

The Hydroponic Kit included several sub-components for use such as:

- PVC Tray and Cap with 7 equally spaced slots for plant holders.
- 7 x plastic plant holders.
- 1 x Wired Pump.
- Plastic Pipe and Elbow fittings for Pump connection.

The sub-components in the kit, their fitting recommendations, and the basic principles of NFT were used as the design constraints for designing the layout of our NFT model. The two main NFT design constraints were:

1. Tray to be slightly tilted to allow for flow of water through the partially immersed plant roots.
2. A reservoir for nutrient solution and water mixture (to be pumped out using submersible pump).

Figure 22, below, is a hand-drawn illustration of the initial design of the model. Both main design constraints were satisfied in this design. It is notable that our model was designed with the pump pipe entering the tray in the same location as the fall of the nutrient solution back into the reservoir. In **Section 2.2**, Figure 3 shows a different design wherein the pipe is shown to enter the system on the opposite end of the tray. This was a small difference between the industry method and our design as we were constrained by the sub-components provided in the kit. However, the flow of water in both cases is the same and hence, the difference in design has a very negligible effect, if any, in the overall mechanics of our system.

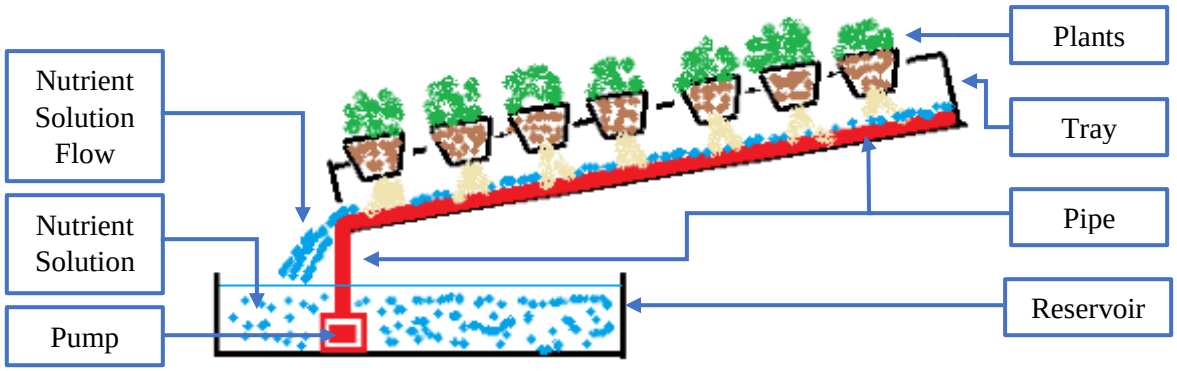


Figure 22: *Initial design of NFT Model*

Using this initial model, design constraints were developed for the enclosure of the system and split into three categories as shown in the table below:

Table 4: *General, Nutrient Solution Reservoir and Electronics Housing Constraints set for design of enclosure*

General Constraints	
1.	Tray to be slightly (~5 degrees) tilted to allow flow of water (gravity)
2.	Removable window to allow removal of Tray Cap
3.	Removable top cover to allow easy access into main chamber
4.	Hole for Heating fixture mounting
5.	Holes for Lighting fixtures mounting
6.	Holes for Cooling fixtures mounting
7.	Holes for Sensing fixtures mounting

Nutrient Solution Reservoir Constraints	
8.	Reservoir underneath Tray to allow for design shown in **Figure 22**
9.	Hole for Tap fitting to allow for emptying of reservoir
10.	Hole for pump pipe and Nutrient solution to flow from tray into reservoir
11.	Hole for Nutrient Solution overflow (when filling reservoir)

Electronics Housing Constraints	
12.	Electronics Housing Accessibility feature e.g., removable windows
13.	Hole for wires and cables to be passed out of enclosure

To design the enclosure, Solidworks CAD software was used, and all parts were separately designed and then assembled virtually, to form the final enclosure design. The enclosure was designed with dimensions that matched the main model tray i.e., the length, width and height of the main chamber was designed to be marginally larger than the tray, to reduce any wasted material or space. This top chamber was extended underneath and split into two chambers – one for the nutrient solution reservoir and one for the electronics housing. **Figure 23**, below, shows the Solidworks CAD model of the enclosure. **Figure 24** illustrates the design features that satisfy all the constraints listed in table 3. The numbers in **Figure 24** directly correlate with the constraint numbers in table 3.

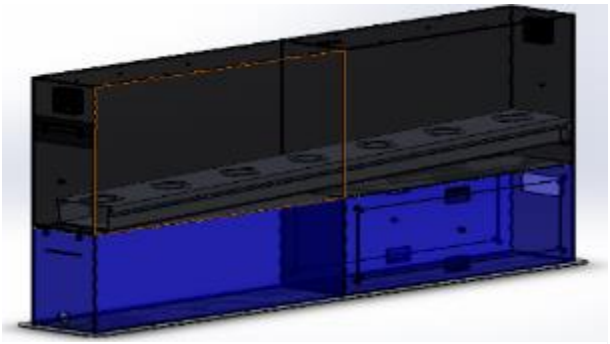


Figure 23: *Solidworks CAD Model of Enclosure*

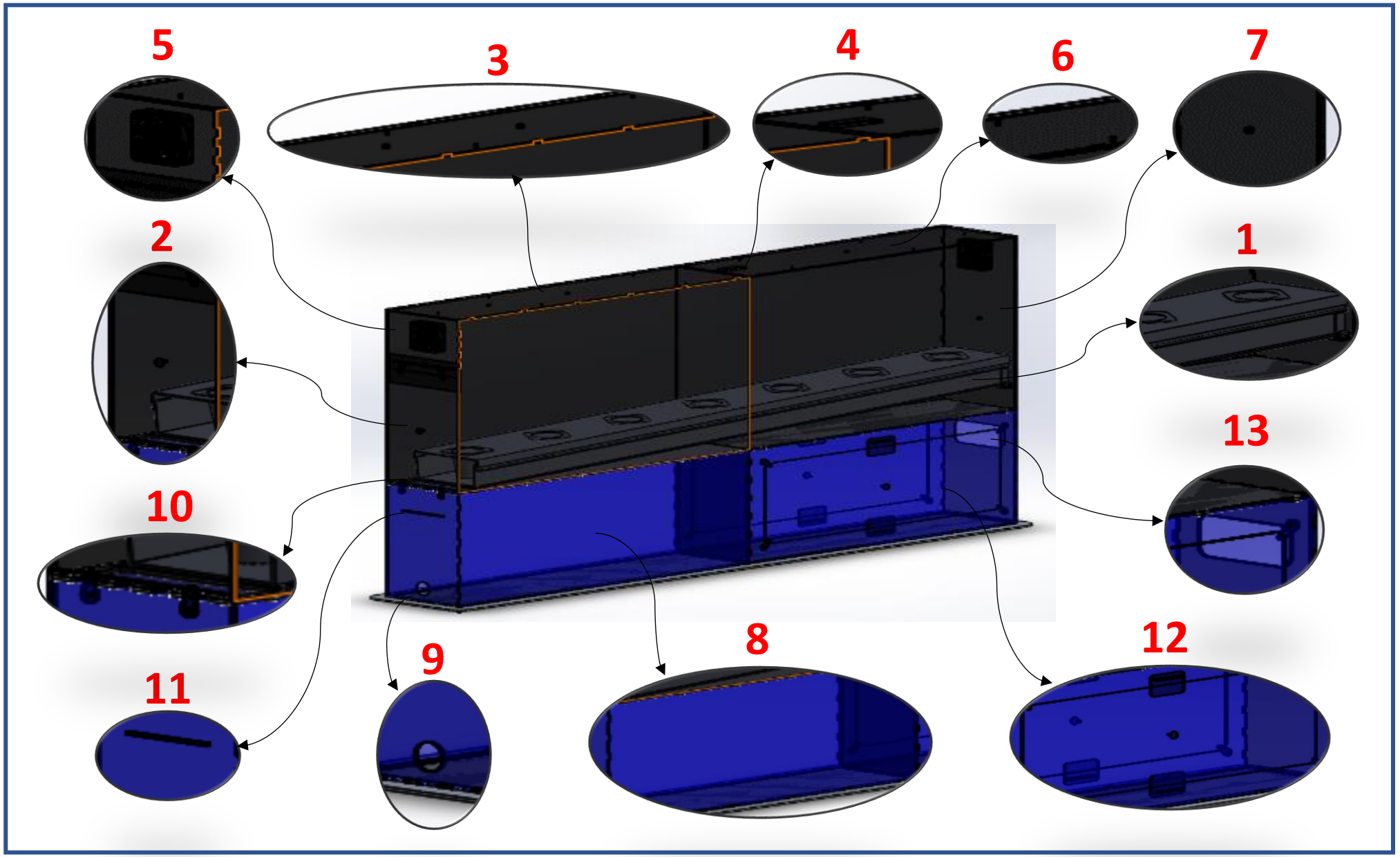


Figure 24: *Enclosure CAD Design and design features for corresponding constraints (numbers correlate with constraint numbering in table 3)*

All parts were designed as separate components; thus, each part was fabricated individually. As a safety feature, it was necessary to have visibility inside the electronics housing and the nutrient solution reservoir. Hence, the enclosure's bottom chambers were fabricated using blue Acrylic. All acrylic parts were cut to the required CAD model specifications using a Laser Cutter machine. To provide some structural rigidity, the entire enclosure's bottom base was manufactured and assembled on a single-piece Aluminium base. This aluminium base was cut to the required CAD model specifications using a Water-Jet Cutter machine. The top chamber, which included the plant tray, heating, cooling, lighting, and sensing components was also made of Acrylic to keep consistency with the bottom chambers. However, it is common industry practice to make the plant enclosure fully enclosed and with dark walls so that no natural lighting can enter the enclosure thus fulfilling the main aspect of this project – 'Controlled Environment Agriculture'. Hence, the top chamber was assembled with opaque, dark black Acrylic.

As the reservoir was to be filled with a mixture of water and nutrient solution, and given the flow of water through the system, it was vital that all chambers were fully sealed. So, to assemble all the chambers together, the acrylic panels were glued together and sealed using Sealant. Thus, ensuring that there was no risk of leakage. The system was tested several times by turning the pump on to ensure no water leaked. This was especially important in the electronics chamber given the potential risk of water entering and damaging any electronic component. Each acrylic panel was designed with a 'tab and slot' to ensure ease of assembly.

Three panels were designed to be removable. Firstly, the front window panel (10 in **Figure 24**) was designed to be removable to provide the capability of sliding out the tray top so that all plants can be removed, and pots replaced. Secondly, the top-most panel (3 in **Figure 24**) was also removable to allow access to the main chamber and heating, cooling, or sensing components. Finally, both side panels of the electronics chamber (12 in **Figure 24**), were removable to allow ease of access of the electronic components inside the chamber. Each of these removable panels were fitted with small steel knobs to act as handles. The top-most removable panel was aligned in place using the tab and slot mechanism but as it was flat on top, no other securing mechanism was required. The front panel was a tight fit panel, hence did not require any locking or securing mechanism. However, the two electronics chamber side panels were secured using four rotating lock tabs to prevent the panel from falling. All securing mechanisms and handles or knob mechanisms on the removable panels are shown in **Figure 25** below.

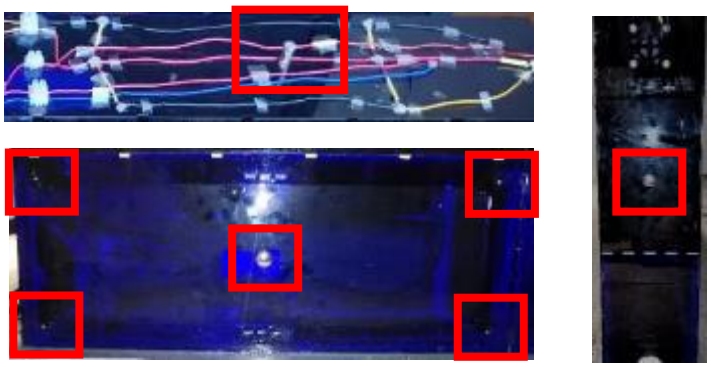


Figure 25: *Removable panels - Top-most panel, tab and slot fit (Top left), Electronics Chamber side panel with four rotating lock tabs (Bottom left), Front window panel with a tight fit (Right)*

The full system included various components that were used in the NFT model, and they are all listed in table 4, below. This included the heating, cooling, sensing, and lighting elements that were used. For our experiment it was important that appropriate lighting was used to replicate the natural lighting that normal plants would receive and replicate the current industry artificial lighting methodologies. For this purpose, ultra-bright LEDs were used. Similarly, the sensors that were used for the temperature readings were the exact same sensors that are common in the industry as discussed in **Section 2.4**. Two non-waterproof sensors were used to measure ambient temperature in the main plant chamber and one waterproof sensor was used to measure the nutrient solution temperature. The heater and fans were an exception to the use of components like the industry as our enclosure's size and the materials used limited the types of components that could be used.

For example, before the final heating element was selected for experiment, multiple heating elements were tested to understand the rate of heating of the air inside the enclosure. For this purpose, multiple types of ceramic heaters were used but a common problem observed was that the heating element heated up very quickly and thus started affecting the acrylic enclosure around the plant tray. Also, although the heater's rate of heating was quick, the air temperature inside the enclosure did not rise at a similar rate. If the heater was left on for a longer period to test the air temperature, there was risk in the enclosure melting. After this testing, a suitable heating element was selected.

Table 5: *Full List of Mechanical Components used in model system*

Component List			
Part No.	**Part Name / Description**	**Qty**	**Supplier**
1.	DC, 12V FANS	2	JAYCAR
2.	DS18B20 Temperature Sensor (Non-Waterproof)	2	SURPLUSTRONICS
3.	DS18B20 Temperature Sensor Probe (Waterproof)	1	
4.	Ultra-Bright, 3mm, LEDs	10	
5.	STEGO, 50W, Touch-safe Heater	1	RS COMPONENTS
6.	Top Hat punched DIN Rail, 0.5 metres	1	
7.	AC 240 V, Pump (part of original hydroponic kit)	1	BUNNINGS WAREHOUSE
8.	Plastic Tap	1	
9.	PVC Tray (part of original hydroponic kit)	1	
10.	PVC Tray Cover (part of original hydroponic kit)	1	
11.	Plastic Plant Holders (part of original hydroponic kit)	7	
12.	Zinc Knobs / Handles	5	

Figure 26 and **Figure 27**, on next pages, shows images of the final model, highlighting all the components listed in table 4, in their final positions after assembly. Following this, **Figure 28**, shows the final model CAD design and an image of the final model after manufacturing and assembly.

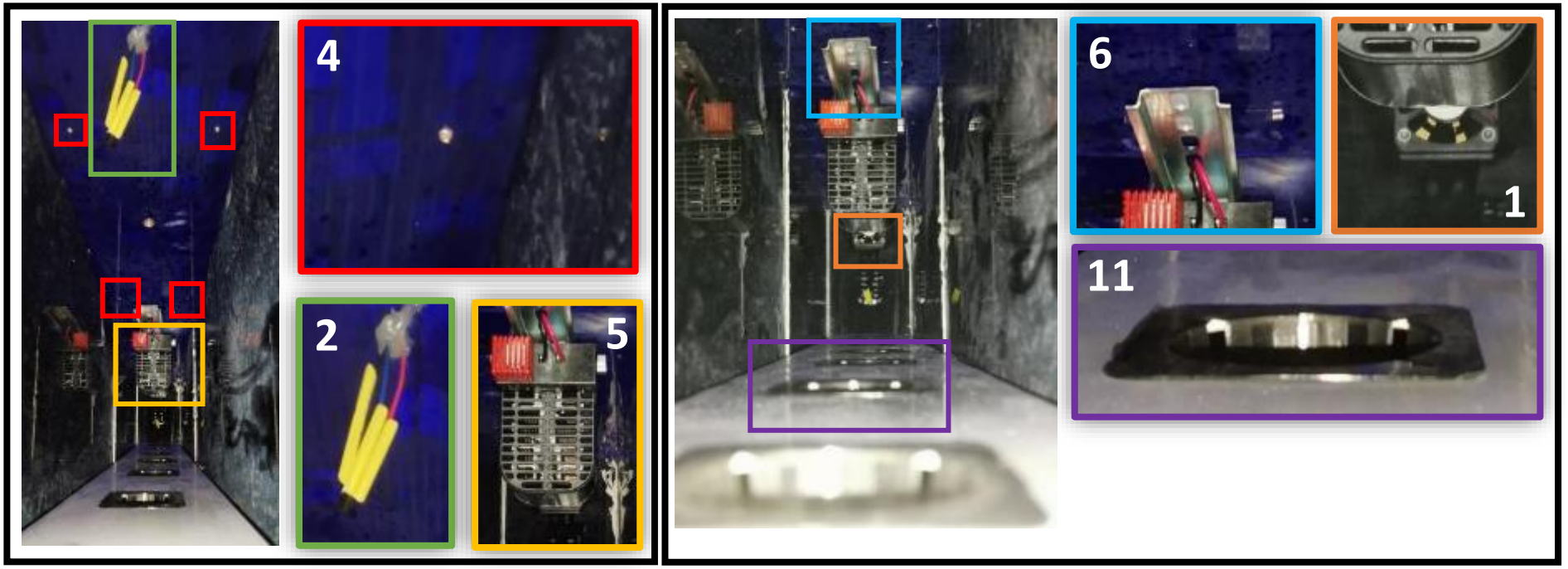


Figure 26: *Model components in final position. Top plant chamber inside view, Ultra-bright LEDs, Temperature Sensor, STEGO Heater, DIN Rail, DC Fan, Plant Holder – Numbers correlate part numbers in table 4*

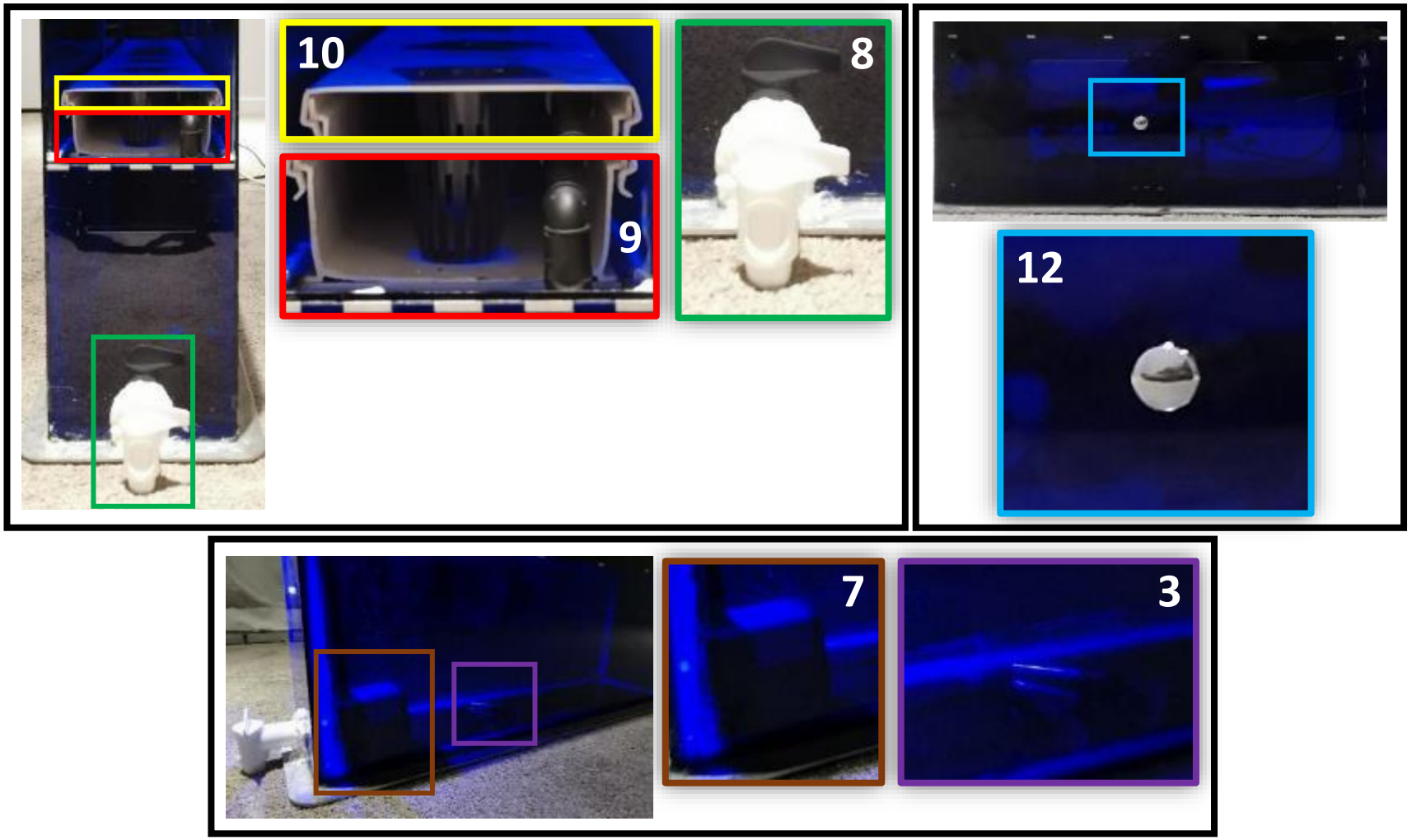


Figure 27: *Model components in final position. PVC Tray, PVC Tray Cover, PVC Tap, Zinc Knob / Handle, Pump, Temperature Sensor (waterproof) – Numbers correlate part numbers in table 4*

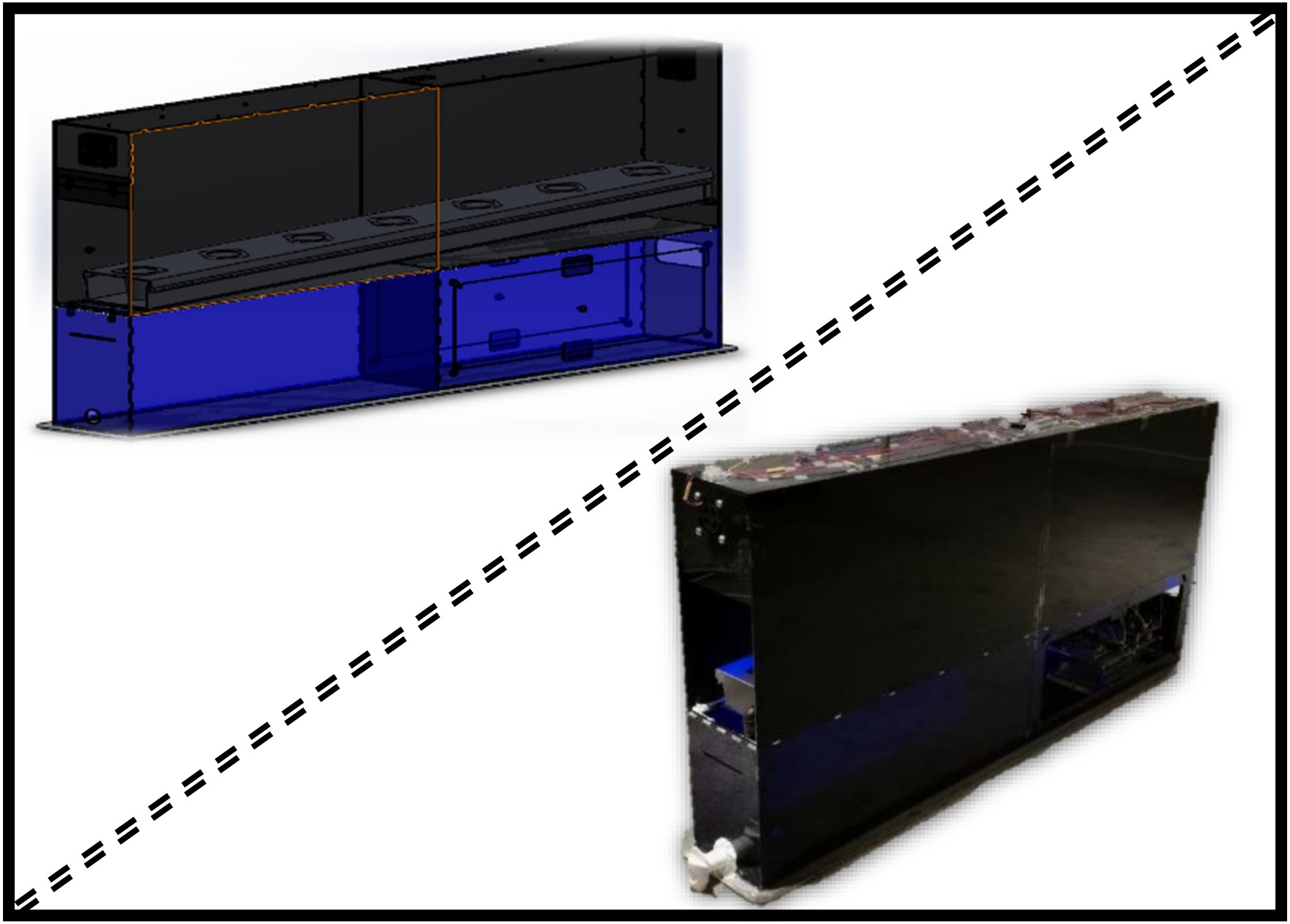


Figure 28: *Final Model CAD Design (Left), Final Model after manufacturing and assembly (Right)*

3.3 Electronics Setup

Our experimental set up was formed on a basic process of reading the current ambient temperature and within an embedded system, using the reading and a set of rules in a controller to actuate a component to heat or cool the environment. The major electronics components required were a temperature sensing element, heating element, cooling element, power source and controller. As we were controlling an environment, lighting components were also required.

Each of the components had a sub-system i.e., the 'cooling' sub-system required a switch (Relay) and a Fan. When the switch was turned on using a signal from the micro-controller, the power supply was connected to the cooling element. The remaining sub-systems: heating, temperature reading, lighting, and controller also had their respective components. The 'power' sub-system included the Switch Mode Power supply and two DC voltage regulators. Switch Mode power supply converted 240V AC to 24V DC and then the two voltage regulators converted 24V DC to 12V DC and 5V DC. Rather than reducing the output of the Switch Mode Power Supply directly to 12V DC, the 24V DC supply was used as an additional step to ensure power capabilities were maintained for any future improvements. Table 5 shows a full list of the components used in our model.

Table 6: *Full List of Electronic Components used in model system*

Component List			
Part No.	**Part Name / Description**	**Qty**	**Supplier**
1.	Ultra-Bright, 3mm, LEDs	10	SURPLUSTRONICS
2.	DS18B20 Temperature Sensor (Non-Waterproof)	2	
3.	DS18B20 Temperature Sensor Probe (Waterproof)	1	
4.	DC, 12V FANS	2	JAYCAR
5.	DC Voltage Regulator	2	Massey University Electronics
6.	Resistor 4700 Ω	3	
7.	Resistor 10,000 Ω	2	
8.	Capacitor 10 µF	1	
9.	Capacitor 100 nF	1	
10.	BC337 NPN Transistor	2	
11.	Diode	2	
12.	5V Relay Switch	1	RS COMPONENTS
13.	12V AC Relay Switch	1	
14.	Switch Mode Power Supply	1	
15.	Raspberry Pi 4	2	

A through-hole PCB (printed circuit board) was designed using Eagle PCB Design Software. A schematic and board layout was designed and then developed to optimise the layout to reduce wastage of any material. The PCB was fabricated using a PCB printing machine at Massey University. Figures in the Appendix, detail the schematic and board of the PCB respectively.

Once fabricated, the PCB, Raspberry Pi microcontroller and Switch Mode Power Supply were mounted on a 2-layer platform which was installed in the Electronics chamber in the enclosure. The platform was manufactured using clear acrylic and MDF (scrap pieces were used to save material) and was cut to size using a laser cutting machine. The spacers between both layers were fabricated using a 3D printing machine. All these parts were designed in Solidworks 3D CAD software. All the pins and ports were connected using jumper wires. **Figure 31**, below, shows the final installation of the PCB (red square), Raspberry Pi (yellow square, Switch Mode Power Supply (green square). **Figure 32** shows the platform used – acrylic (orange square), MDF (green square) and spacer (yellow square).

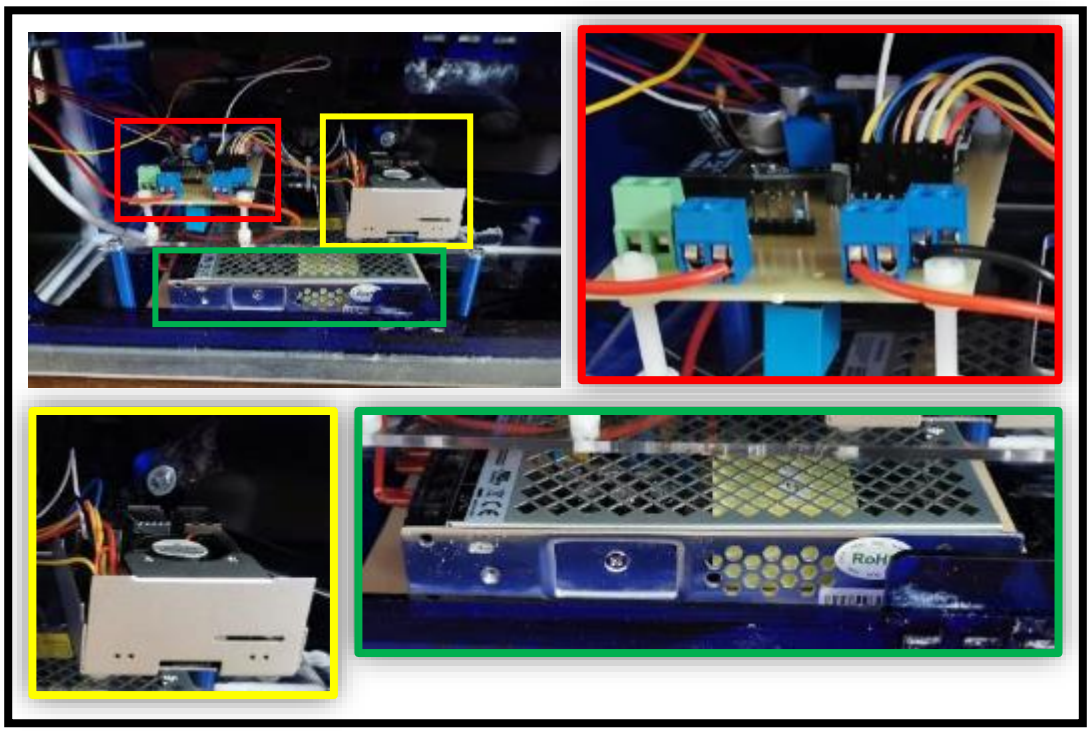


Figure 29: *Final installation of the PCB (Top Right), Raspberry Pi (Bottom Left), Switch Mode Power Supply (Bottom Right) within enclosure*

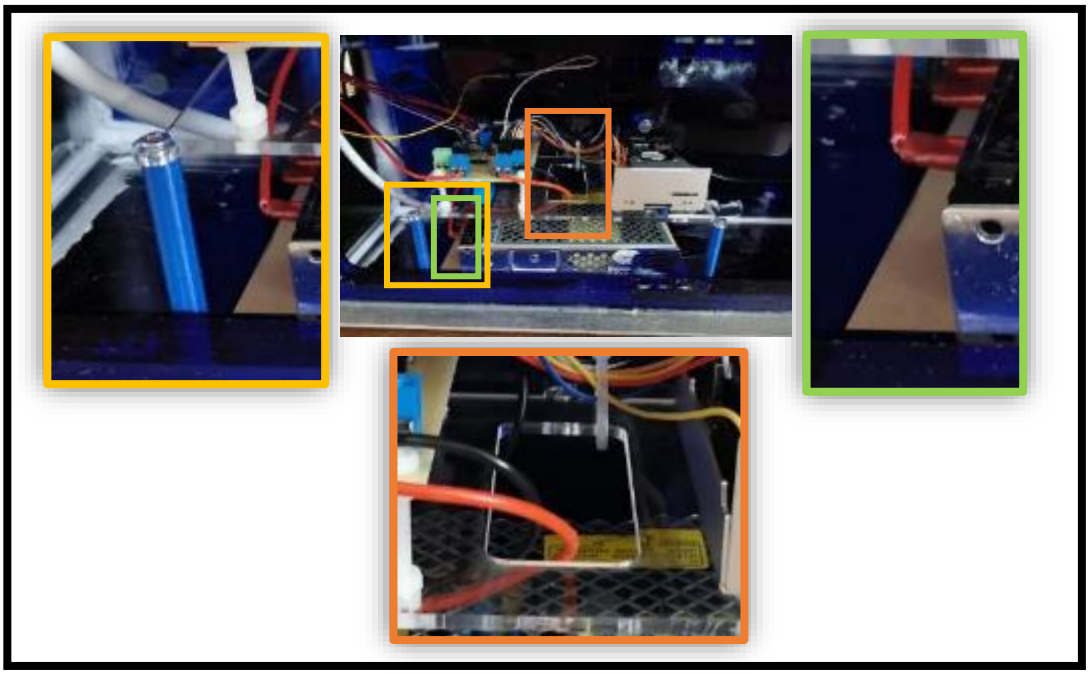


Figure 30: *Final installation of electronics platform (spacer – top left, MDF base – top right, acrylic platform – centre bottom) within enclosure*

3.4 Software Setup

The software setup of our model was based on the main experimental methodology of connecting one Raspberry Pi microcontroller to the Hydroponics system and controlling the components (i.e., the client end), while the other Raspberry Pi would be used to simulate the website and user input or control (i.e., the server end). The two Raspberry Pi microcontrollers would communicate wirelessly through SSH, and the code files would be accessed and edited using Visual Studio Code on a separate Windows device. Hence, to begin, both Raspberry Pi's had SD cards connected and the SD cards were flashed with Ubuntu Server 20.04 using a Raspberry Pi imager tool [73].

Following this, the following ROS 2 packages and Python tools were installed on both Raspberry Pi microcontrollers:

- ROS 2 Foxy Distribution was installed using the instructions provided:
  - http://docs.ros.org/en/foxy/Installation/Ubuntu-Install-Debians.html [74]
- ROS 2 Colcon Package was installed using the instructions provided:
  - http://colcon.readthedocs.io/en/released/user/installation.html [75]
- Python tools were installed using the following commands:

```
sudo apt install python3
sudo apt install pip3
```

OpenSSH client and server application was used for wireless communication between both microcontrollers via the Windows device. To install and start the OpenSSH client and server applications on the Ubuntu systems, the following commands were used in terminal on the microcontroller:

```
sudo apt install openssh-client
sudo apt install openssh-server
```

Following the installation, the configuration file was edited to ensure authentication for connection i.e., username and password login. Both details were set in this file and then the OpenSSH service was restarted using the following command:

```
sudo systemctl restart sshd.service
```

The two Raspberry Pi microcontrollers were now ready for communication with each other via the Windows device. Once both microcontrollers were prepared with ROS, Visual Studio Code was installed on the Windows device from https://code.visualstudio.com/. Once completing the installation, the following remote development extensions were installed in Visual Studio Code to ensure SSH connection can be made, and code files can be edited [76]:

- C/C++ and Python, Remote – Containers, and Remote – SSH

Once the Visual Studio Code setup was complete, in two separate Visual Studio Code terminals, both microcontrollers were connected using the SSH connections. In both microcontrollers, a workspace, and corresponding directories, were created. Following this, a publisher and subscriber package was also created. The following commands were used [77]:

```
mkdir -p ~/workspace/src          # creating the /workspace/src directory

ros2 pkg create –build-type ament_python py_pubsub    # creating a new python package named py_pubsub
```

Within this new package, new Python files were created and named as 'pub' (publisher) and 'sub' (subscriber). Code was written in both files to create nodes that published and subscribed to a topic, and sent & received information to each other. This was obtained from some previously written publisher and subscriber lecture slides [77]. **Figure 31**, shows the directory structure that was created.

```
/Ubuntu
    /workspace
        /src
            /py_pubsub
                /launch
                /py_pubsub
                    /__init__.py
                    /pub.py
                    /sub.py
                /package.xml
                /setup.cfg
                /setup.py
```

Figure 31: *Directory Structure*

The Ros topic was kept as 'topic' to begin. In the publisher file, the node was initiated and was 'talking' on the topic, 'topic'. A timer call back was used which called the 'timer_callback' function repeatedly at a set interval. The function outputted a set message. The subscriber node was similar – the node was initiated and set to 'listen' to the topic 'topic'. Similarly, a call back was used to call the listener function which outputted the message that had been received.

The setup file was then edited to create 'entry's for both publisher and subscriber nodes. To test if both microcontrollers are communicating with each other, the publisher topic on one Raspberry Pi was matched with the subscriber of the other and vice-versa. The workspace was built and both files were run to ensure the connection was working. As both microcontrollers were sending and receiving different information, a successful build was confirmed. **Figure 34**, on the next page, shows the publisher and subscriber files that were initially created in both Ubuntu systems on the microcontrollers.

workspace > src > py_pubsub > py_pubsub > sub.py > ...

```python
import rclpy
from rclpy.node import Node
from std_msgs.msg import String


class Sub(Node):

    def __init__(self):
        super().__init__('sub')
        self.subscription = self.create_subscription(
            String,
            'topic',
            self.listener_callback,
            10)
        self.subscription

    def listener_callback(self, msg):
        self.get_logger().info('I heard: "%s"' % msg.data)


def main(args=None):
    rclpy.init(args=args)

    sub = Sub()

    rclpy.spin(sub)

    sub.destroy_node()
    rclpy.shutdown()


if __name__ == '__main__':
    main()
```

workspace > src > py_pubsub > py_pubsub > pub.py > ...

```python
import rclpy
from rclpy.node import Node
from std_msgs.msg import String


class Pub(Node):

    def __init__(self):
        super().__init__('pub')
        self.publisher_ = self.create_publisher(
            String,
            'topic',
            10)
        timer_period = 0.5
        self.timer = self.create_timer(timer_period, self.timer_callback)
        self.i = 0

    def timer_callback(self):
        msg = String()
        msg.data = 'Hello World: %d' % self.i
        self.publisher_.publish(msg)
        self.get_logger().info('Publishing: "%s"' % msg.data)
        self.i += 1


def main(args=None):
    rclpy.init(args=args)

    pub = Pub()

    rclpy.spin(pub)

    pub.destroy_node()
    rclpy.shutdown()


if __name__ == '__main__':
    main()
```

Figure 32: *Publisher and Subscriber Files created. Subscriber Node (left), Publisher Node (Right)*

Once this initial set up was established, the temperature sensor was connected to the Raspberry Pi and code was added to the above 'Publisher' file to obtain temperature readings. The temperature sensor that we used communicates with a 'one wire' communication protocol i.e., only one wire is used to send readings to the Raspberry Pi. Using this, the one wire interface was enabled in Python code and the system files is edited to ensure the port is set as in 'slave' mode to receive readings. Using the available resources online, code was obtained for temperature readings [78]. **Figure 35**, below, shows the initial temperature reading code. The reading is received as text in a file from the sensor. The file is read and the temperature reading portion, originally in degrees Fahrenheit, is converted to degrees Celsius and then output.

```
import rclpy
import sys
import os
import glob
import time
from datetime import datetime
os.system('modprobe w1-gpio')
os.system('modprobe w1-therm')
base_dir = '/sys/bus/w1/devices/'
device_folder = glob.glob(base_dir + '*24c5')[0]
device_file = device_folder + '/w1_slave'
from rclpy.node import Node
from std_msgs.msg import String

# Below: Temperature Sensor (ambient)

def read_temp_raw():
    f = open(device_file, 'r')
    lines = f.readlines()
    f.close()
    return lines
def read_temp():
    lines = read_temp_raw()
    while lines[0].strip()[-3:] != 'YES':
        time.sleep(0.2)
        lines = read_temp_raw()
    equals_pos = lines[1].find('t=')
    if equals_pos != -1:
        temp_string = lines[1][equals_pos+2]
        temp_c = float(temp_string) / 1000.0
        temp_c = round(temp_c, 1)
    return temp_c2
```

Figure 33: *Initial Temperature Reading Code*

The temperature reading code was then merged with the initial publisher message. Hence, like the message was being transmitted previously, the temperature reading was now being transmitted from one Raspberry Pi to the other. The Raspberry Pi input and output ports were then imported in the Python code and was implemented to turn the Fan and Heater on by sending a 'HIGH' signal to the GPIO (General Purpose Input Output) pins (consequently a 'LOW' signal to turn them off). **Figure 36**, below, shows the GPIO port initiation and the code that turns the components on or off. This was an initial set up and the heater and fan component 'on /off' functionality was moved to another function which is described in later sections.

```
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO.setup(26, GPIO.OUT) # LEDs
GPIO.setup(27, GPIO.OUT) # Heater
GPIO.setup(22, GPIO.OUT) # Fans
from rclpy.node import Node
from std_msgs.msg import String

def LEDs():
   GPIO.output(26, GPIO.HIGH) # LEDs On
   GPIO.output(27, GPIO.HIGH) # Heater On
   GPIO.output(22, GPIO.HIGH) # Fan On
```

Figure 34: *GPIO port and component on/off*

While the sensing, cooling, heating components were programmed on one Raspberry Pi microcontroller, the final part of the software set up was installing Flask, a web application framework [79], and was done on the second Raspberry Pi microcontroller. This would provide the interface to the user for inputting the temperature set point and the web application would also display the current temperature. The Flask framework was installed using the following command [80]:

python -m pip install –upgrade flask *# this installs the framework*

A basic code file was created and **Figure 37**, below, shows the code that was written [80]. This was a basic implementation to output the text 'Hello World!' on the web page when executed.

```
from flask import Flask

app = Flask(__name__)

@app.route("/")
def hello_world():
    return "<p>Hello World!</p>"

if __name__ == "__main__":

    app.run()
```

Figure 35: *Basic Implementation of Flask Web Application Framework [80]*

The function within the Flask code was then edited to include a more enhanced display on the web page and display the temperature reading. The formatting was done using HTML code and label & input fields were also implemented to get the set point entries. Due to some technical issues in setting up an HTML program file and merging with our publisher and subscriber Python File, the program was written in Python with HTML code used directly as a 'text' version in the 'return' line. The code was split into two functions. The first function , shown in **Figure 38** below, was the main function that was initiated when the web page was opened. Once the user entered a set point, the second function, **Figure 39** on next page, was called and the set point was received & assigned to a variable, and the current temperature was displayed back on the web page. **Figure 40**, on the following page, shows the web page output when first opened – with a user input field. **Figure 41** and **Figure 42**, on the following page, show the web page displays after the user has entered the initial required temperature set point.

The Python code for the controller set up is described in **Section 3.5** (PID Controller Development) and the integration of the code with the publisher and subscriber is described in **Section 3.6** (System Integration).

```
@app.route("/")
def Current_temp():
    return f'''
    <!DOCTYPE html>
    <html lang="en">
        <head>
            <meta charset="UTF-8">
            <title>Vertical Farm Temperature Control</title>
        </head>
        <body>
            <h1>Vertical Farm Temperature Control</h1>
            <form method = "GET"
                  action="Set Temperature">
                <label for="Set Temperature">Enter Temperature</label>
                <input type="text" name="Set Temperature" value="" required>
                <input type="Submit" value="Set">
            </form>
        </body>
    </html>
    '''
```

Figure 36: *First function (main web page) part of Flask Python code*

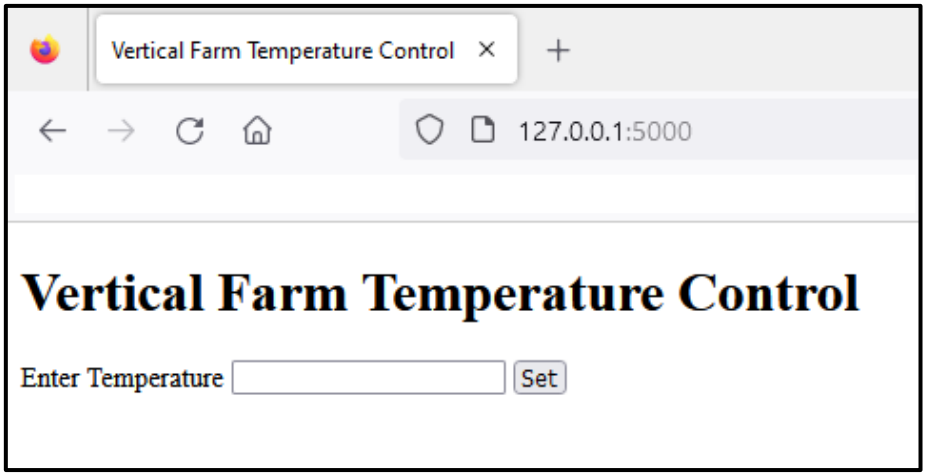


Figure 37: *Web Page display when first opened*

```
@app.route("/Set Temperature")
def current_temp():
    global setpoint, curr_temp
    if request.args.get("Set Temperature") is not None:
        setpoint = request.args.get("Set Temperature")
    return f'''
    <!DOCTYPE html>
    <html lang="en">
        <head>
            <meta charset="UTF-8">
            <title>Vertical Farm Temperature Control</title>
        </head>
        <body>
            <h1>Vertical Farm Temperature Control</h1>
            <form action="Set Temperature" style='float:left;'>
                <label for="Set Temperature">Enter Temperature</label>
                <input type="number" name="Set Temperature" value="" required>
                <input type="Submit" value="Set">
            </form>
            <form action="Set Temperature">
                <input type="Submit" value="Refresh">
            </form>
            <br>
            <h3>The temperature has been set to {setpoint}.</h3>
            <br>
            <h3>The current temperature is: {curr_temp} &#8451</h3>
        </body>
    </html> '''
```

Figure 38: *Second function within Flask Python code, input temperature is assigned to 'setpoint' variable, and current temperature is displayed*

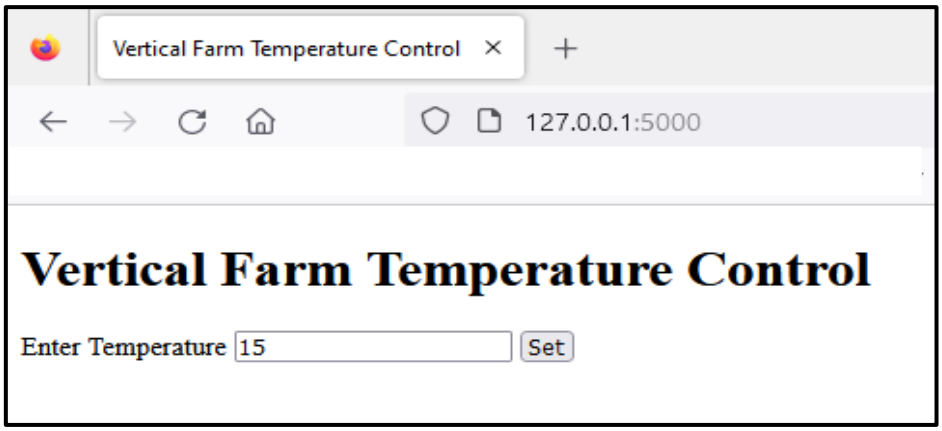


Figure 39: *User Input of set point on web page*

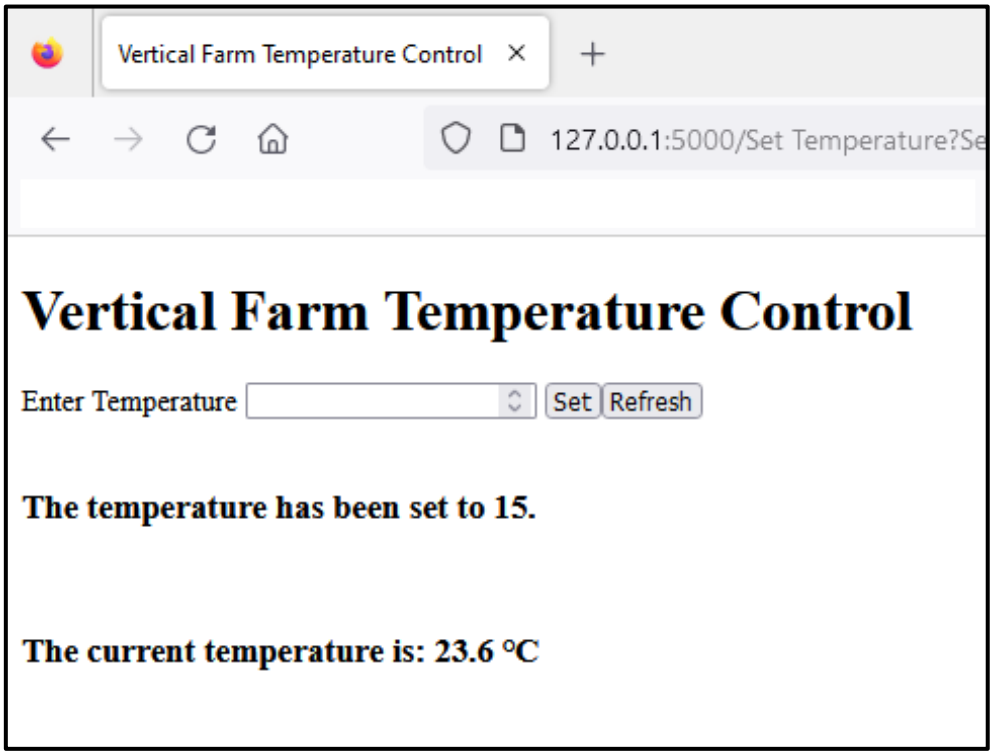


Figure 40: *Web Page display after temperature input has been received*

3.5 PID Controller Development

Monitoring the environment of a plant is one of the key processes in Controlled Environment Agriculture, as discussed in **Section 2.3**, and for our experiment this includes temperature monitoring and control. Monitoring is through temperature sensor, discussed in the previous section. A PID (Proportional, Integral, Derivative) Controller was developed for controlling the ambient temperature. The physical model was all assembled, including all hardware such as temperature sensor, fan, heater, and lights, as explained in **Section 3.2** (Mechanical Set Up). The Electronics Set Up (as explained in **Section 3.3**) was also assembled with the PCB, microcontroller, and Power Supply. Both elements were then combined with all components connected inside the enclosure ready for experimentation. The basic experiment for developing a PID Controller was to run our system i.e., the heater and temperature sensor and to record the readings to obtain a response curve which would be further used to develop our controller. Our experiment was set up as a closed loop system and the parameters of the PID Controller had to be determined and then the controller had to be tuned to ensure that the parameters are optimal. The parameters include K_p (Proportional Gain), K_i (Integral Gain) and K_d (Derivative Gain). To determine these parameters, the Ziegler-Nichols First Method for tuning was used for this [81]. The first step in this method was to obtain a transient response of the system and obtain key parameters from the graph. **Figure 43**, below, shows the response expected from the system and the parameters obtained. The parameters include the delay time (L), time constant (T) and the y-axis intersection point of when the curve stabilises (K). **Figure 44** shows our obtained response and the key parameter values.

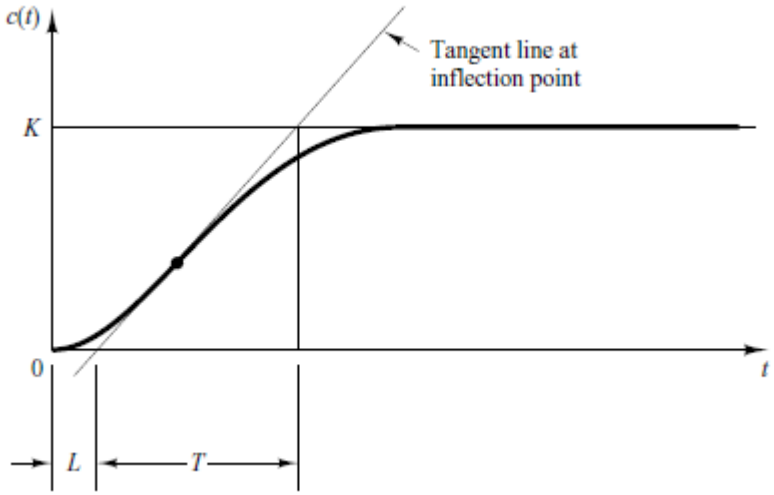


Figure 41: *Expected Transient Response from system and key parameters L, T, and K [81]*

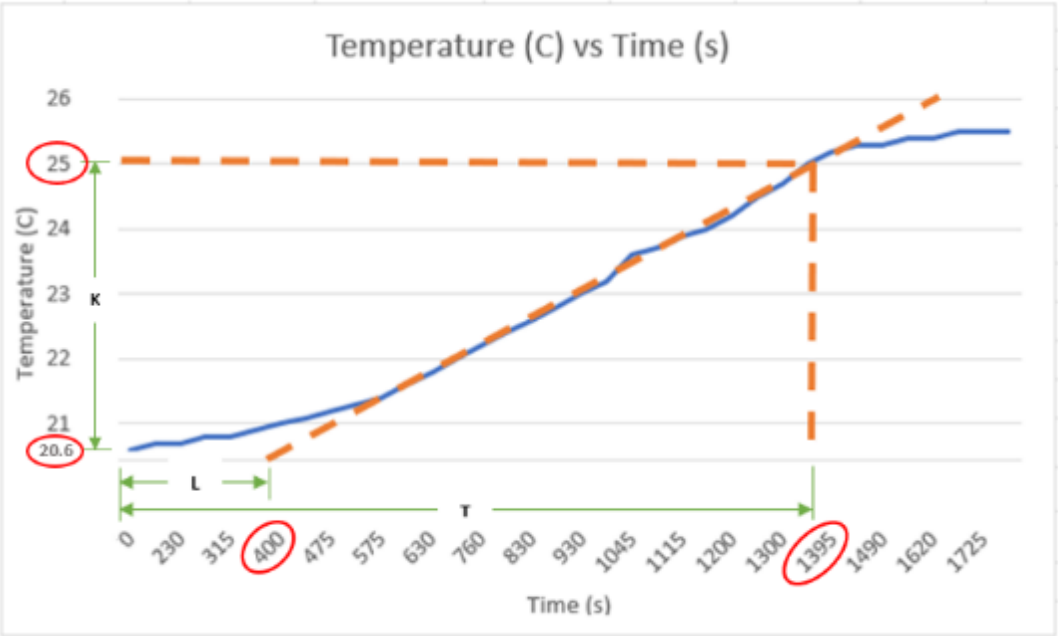


Figure 42: *Transient Response of our system and the key parameters highlighted*

When comparing the response in **Figure 43** to our response in **Figure 44**, one of the differences was our temperature (y-axis) not starting at zero. Hence, the L and T parameters could be obtained by reading the values from the graph; however, a small calculation needed to be done to obtain K, as below:

- $L = 400 - 0 = 400\ s$
- $T = 1395 - 0 = 1395\ s$
- $K = 25 - 20.6 = 4.4\ ℃$

Using these values, we obtained the transfer function of the first-order system using the equation below [81]:

$$\frac{C(S)}{U(S)} = \frac{Ke^{-Ls}}{Ts + 1} = \frac{4.4e^{-400s}}{1395s + 1} \qquad (1)$$

A model of our closed loop system was then created in MATLAB & SIMULINK software to simulate the behaviour of our system. **Figure 45**, below, shows our closed loop system with a step input to obtain a step response and set up with our transfer function and a 'transport delay' of 400s (obtained from our transfer function). A PID controller was added to aid the next process of tuning our controller.

Figure 43: *Closed Loop system schematic for obtaining step response and tuning PID Controller*

When the PID controller module was opened, the initial PID controller parameters K_p (Proportional Gain), K_i (Integral Gain) and K_d (Derivative Gain) were automatically set as shown in **Figure 46** below.

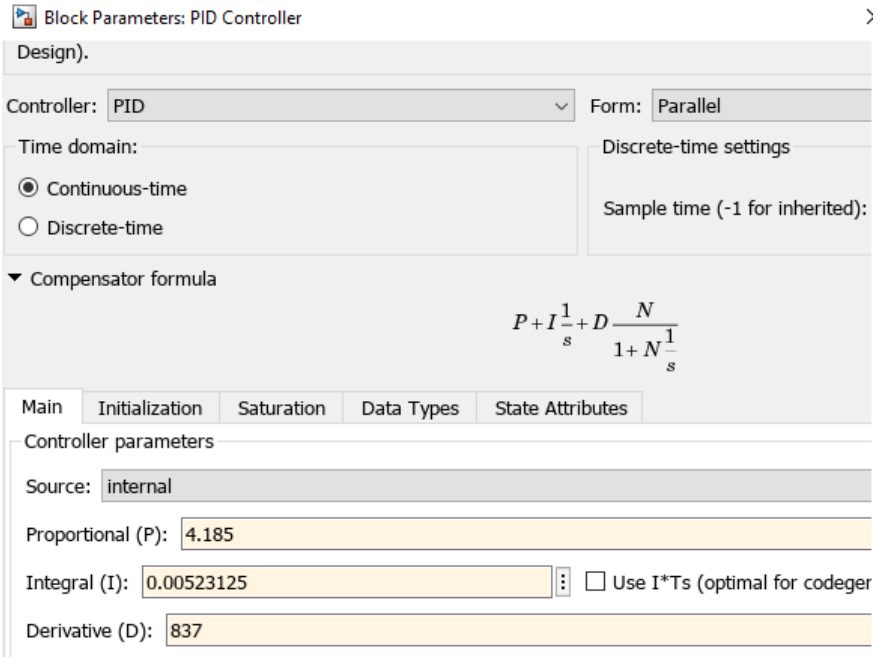


Figure 44: *PID Controller block and initial values of P, I, and D*

To ensure that these parameters were correct and matched our system, a small verification was carried out. **Figure 47**, below, shows the Ziegler-Nichols tuning formulas [81] for obtaining the Kp, Ti, and Td parameters. As we are designing a PID controller, the third row of formulae in the figure below are relevant.

Type of Controller	K_p	T_i	T_d
P	$\frac{T}{L}$	∞	0
PI	$0.9\frac{T}{L}$	$\frac{L}{0.3}$	0
PID	$1.2\frac{T}{L}$	$2L$	$0.5L$

Figure 45: *Ziegler-Nichols tuning formula to calculate the Kp, Ti, and Td parameters*

Using the formulas above and our values for L, T, and K, the parameters were calculated as:

$$K_p = 1.2\frac{T}{L} = 1.2\frac{1395}{400} = 4.185 \quad (2)$$

$$T_i = 2L = 2 \times 400 = 800 \quad (3)$$

$$T_d = 0.5L = 0.5 \times 400 = 200 \quad (4)$$

$$K_i = \frac{K_p}{T_i} = \frac{4.185}{800} = 0.00523125 \quad (5)$$

$$K_d = K_p T_d = 4.185 \times 200 = 837 \quad (6)$$

As these Proportional, Integral and Derivative gain values matched the Simulink P, I, and D values, we could confirm that our Simulink model was working as expected. The output, when the model was run, is shown in **Figure 48**.

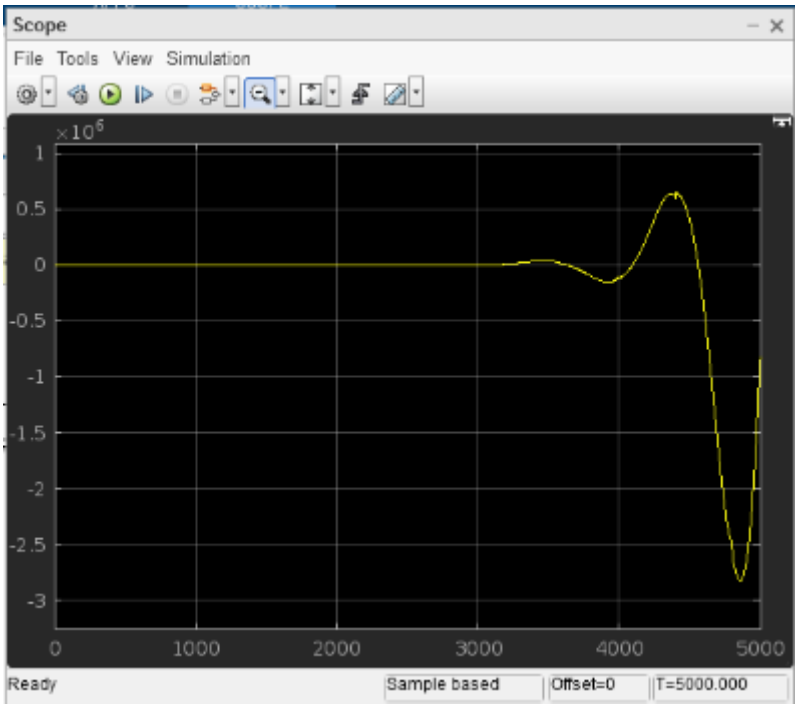


Figure 46: *Output of initial closed loop system and PID controller*

To tune the closed loop systems, the characteristics of each of the P, I, and D terms were used. **Figure 49** [82], below, shows the theoretical output expected upon increasing any of the P, I or D terms.

CL RESPONSE	RISE TIME	OVERSHOOT	SETTLING TIME	S-S ERROR
Kp	Decrease	Increase	Small Change	Decrease
Ki	Decrease	Increase	Increase	Decrease
Kd	Small Change	Decrease	Decrease	No Change

Figure 47: *Expected outputs and characteristics after increasing P, I, and D parameters [82]*

Using the expected outputs from **Figure 49**, the PID controller values were altered and tuned to achieve a better curve output, i.e., a curve output that would rise to the step input and stabilise. The iterative process, to achieve this end output, with each iteration goal and result is shown below. The sample Time for each iteration was 5000 seconds.

Iteration 1, Goal 1: Decrease Overshoot		
Parameter	**Old Value**	**New Value**
Kp (decrease by 50 %)	4.185	2.0925
Ki (decrease by 50 %)	0.00523125	0.002616
Kd (increase by 50 %)	837	1255.5

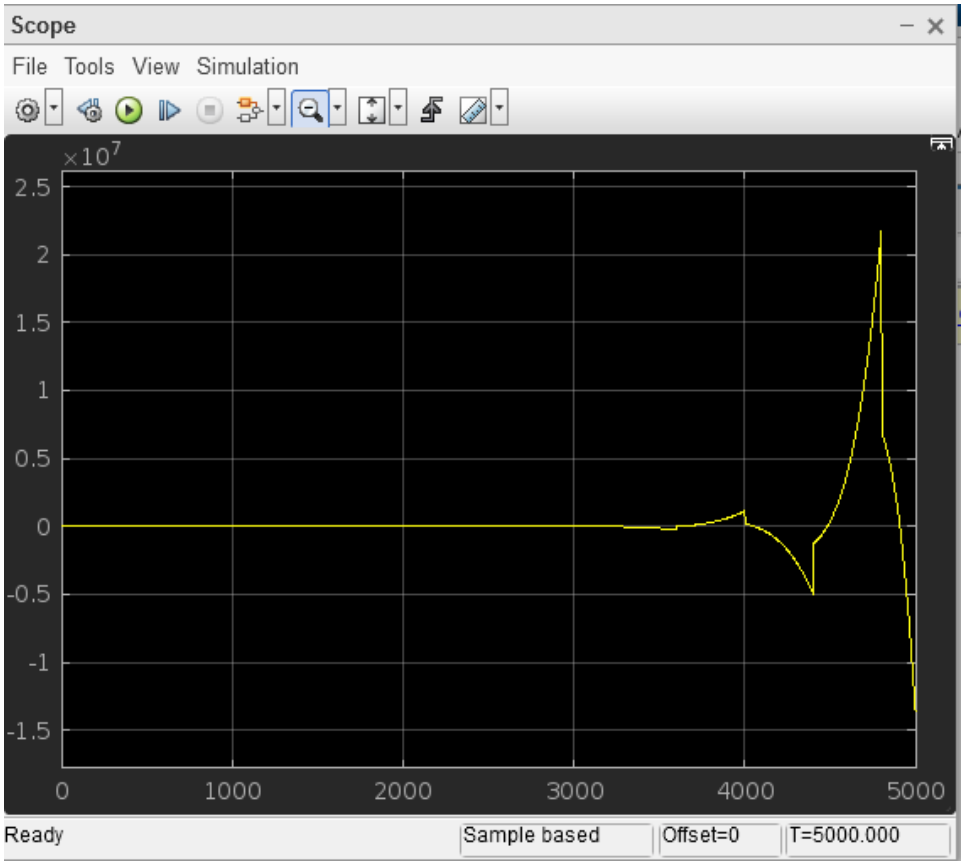


Figure 48: *Output observed after Kp decreased by 50 %, Ki decreased by 50% and Kd increased by 50 %*

Iteration 2, Goal 2: Repeat Decrease Overshoot		
Parameter	**Old Value**	**New Value**
Kp (decrease by 50 %)	2.0925	1.04625
Ki (decrease by 50 %)	0.002616	0.001308
Kd (increase by 50 %)	1255.5	1883.25

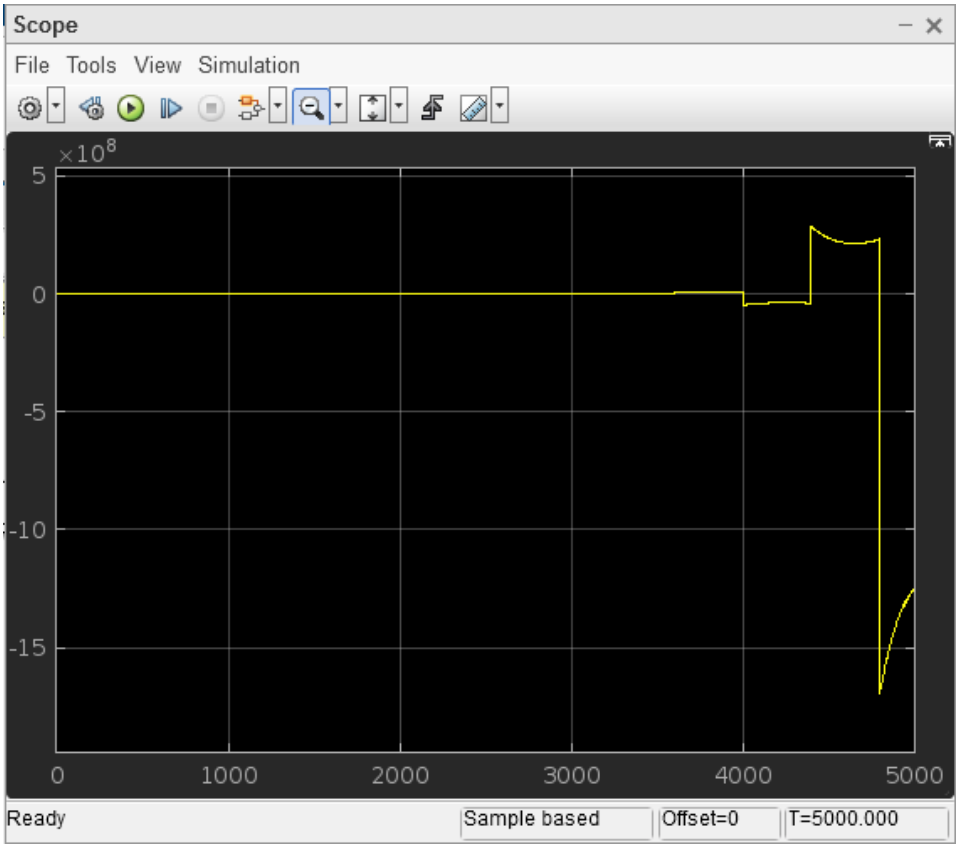


Figure 49: *Output observed after Kp decreased by 50 %, Ki decreased by 50% and Kd increased by 50 % once more*

It can be seen from the figures above, that the output curve, started to form a wave pattern after Kp and Ki were decreased by 50 % and Kd was increased by 50 %.

While undertaking a test by randomly changing the parameters, it was observed that decreasing Kd resulted in a smoother waveform and contradictory to the characteristics above, reducing the derivative gain yielded in a reduced overshoot. To test this further, in iteration 3, a test was done to observe a change in overshoot with only decreasing Kd by 50 %. A significant change could be seen in the output, when compared to the previous result, and a curve-like pattern was further observed, as seen in **Figure 52** below. Hence, to test the output of a significant change in the Kd parameter, the Kd parameter was decreased by a further 75 % and the output curve, in **Figure 53**, was observed to be transforming towards the final desired result.

Iteration 3, Goal 3: Test curve output with only Kd		
Parameter	**Old Value**	**New Value**
Kp (no change)	1.04625	
Ki (no change)	0.001308	
Kd (decrease by 50 %)	1883.25	941.625

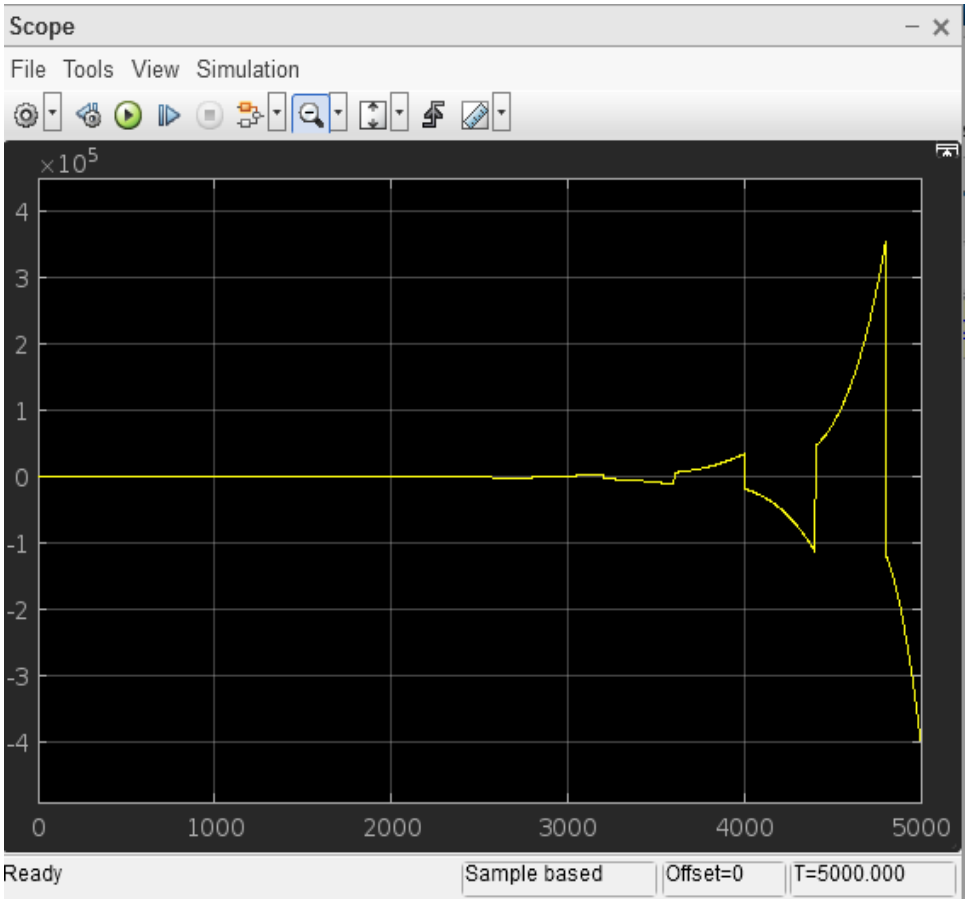


Figure 50: *Further refined output by decreasing only Kd by 50 %*

Iteration 4, Goal 4: Test overshoot and test curve output with only Kd		
Parameter	**Old Value**	**New Value**
Kp (no change)	1.04625	
Ki (no change)	0.001308	
Kd (decrease by 75 %)	941.625	235.4063

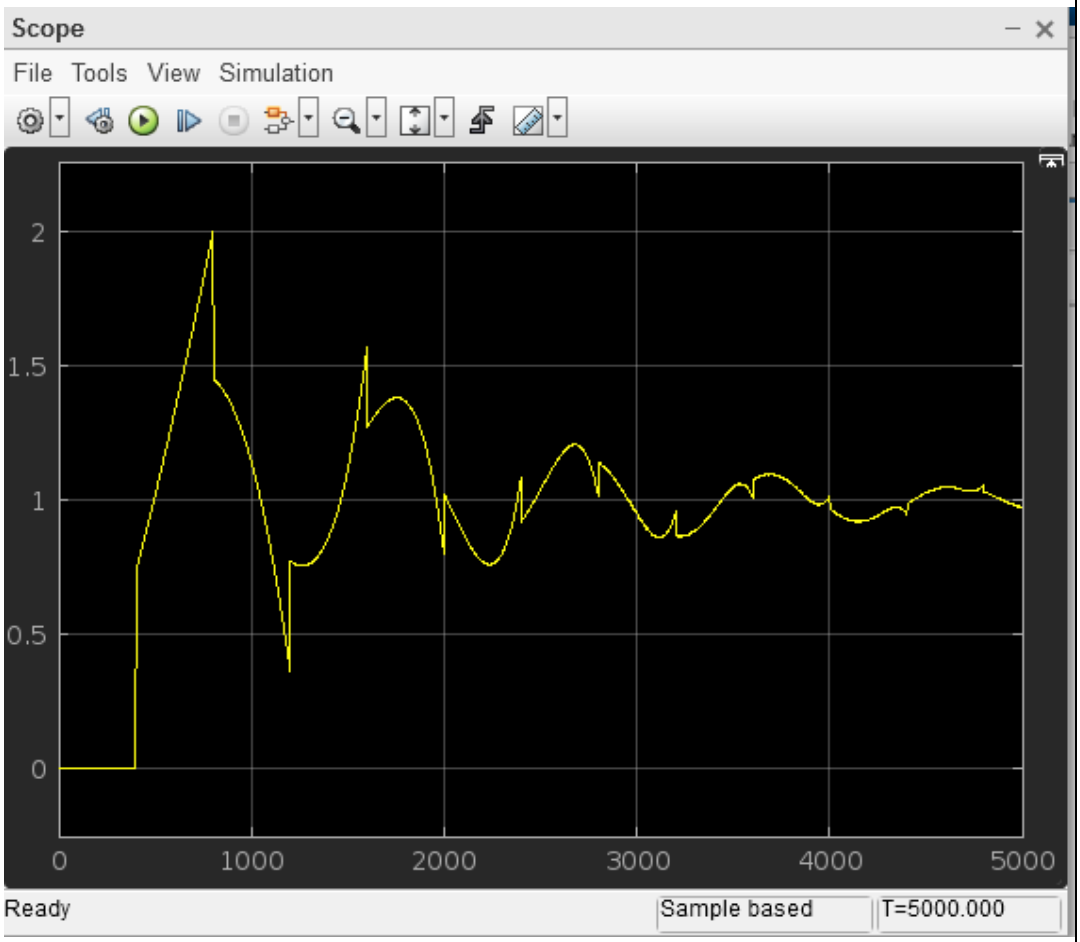


Figure 51: *Decreased Overshoot and smoother curve output after further reduction in Kd parameter*

The same method, of decreasing Kd was tested for a few more iterations and a smooth curve was obtained in iteration 6, as seen in **Figure 54**, below. Following this, the new goal was to decrease settling time, as seen in iteration 7 – **Figure 55** shows the output.

Iteration 6, Goal 6: Test curve output with only Kd		
Parameter	**Old Value**	**New Value**
Kp (no change)	1.04625	
Ki (no change)	0.001308	
Kd (decrease by 50 %)	117.7032	58.8516

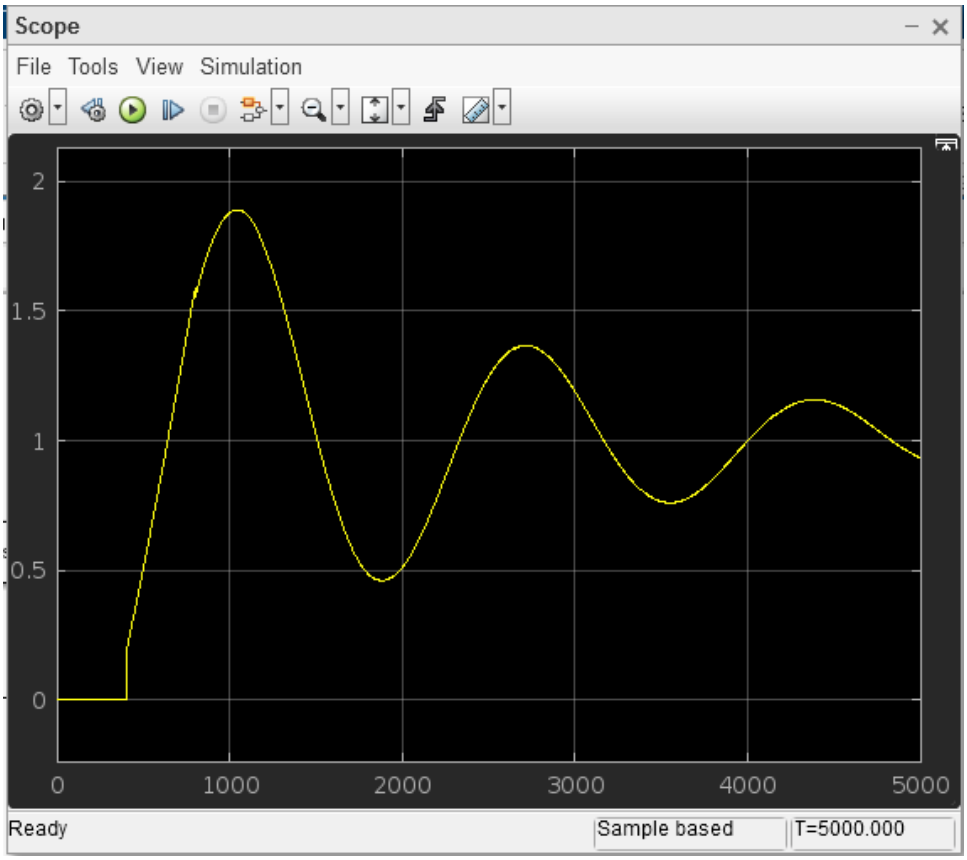


Figure 52: *Final iteration of decreasing Kd to observe smoother curve output*

Iteration 7, Goal 7: Decrease Settling Time		
Parameter	**Old Value**	**New Value**
Kp (decrease by 50 %)	1.04625	0.523125
Ki (decrease by 50 %)	0.001308	0.000654
Kd (no change)	58.8516	58.8516

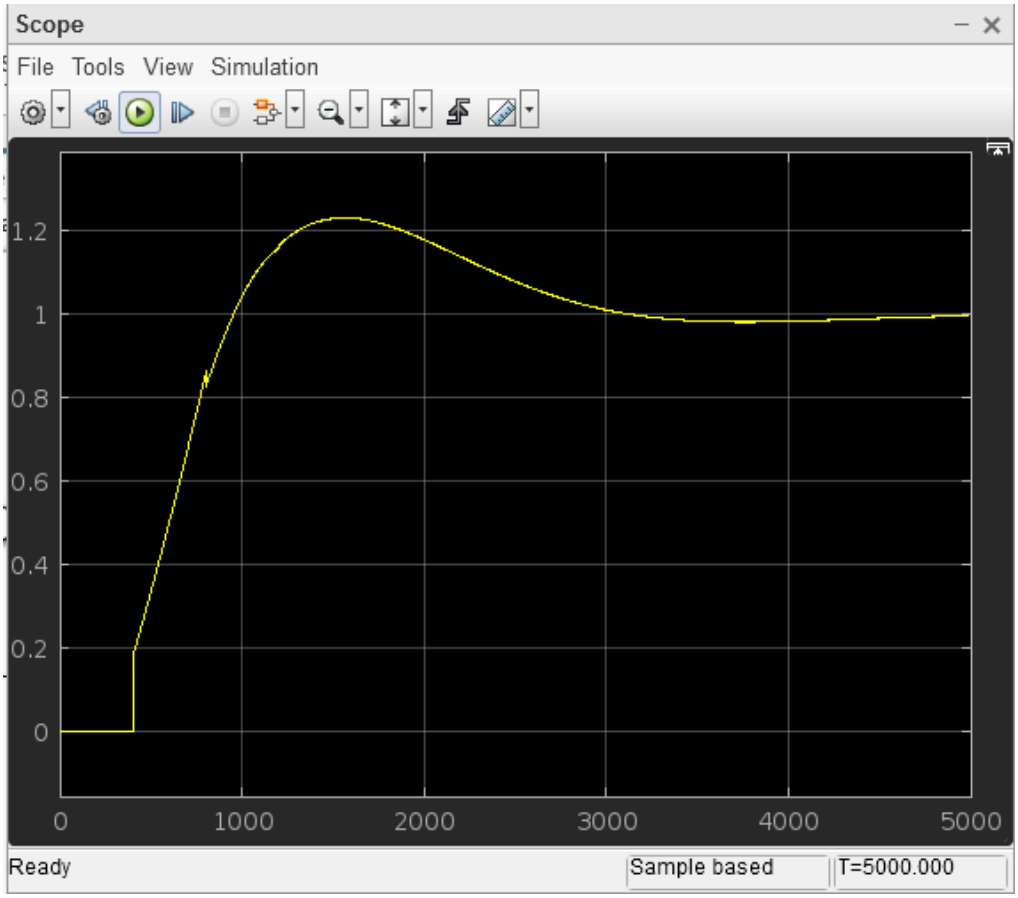


Figure 53: *Curve output after Kp and Ki decreased by 50 % to decrease settling time*

After numerous iterations with different goals of reducing settling time, decreasing rise time, and decreasing overshoot, the final curve output, **Figure 56** below, and corresponding P, I, and D parameters were obtained. It is observed that the 400 second delay that was observed in the initial temperature readings is still maintained in this curve, as expected. **Figure 57** shows a zoomed out view of the same curve output to confirm the stabilisation of the curve around the desired step input value.

Final values after all iterations	
Parameter	**Final Value**
Kp	0.262946
Ki	0.000318
Kd	37.66502

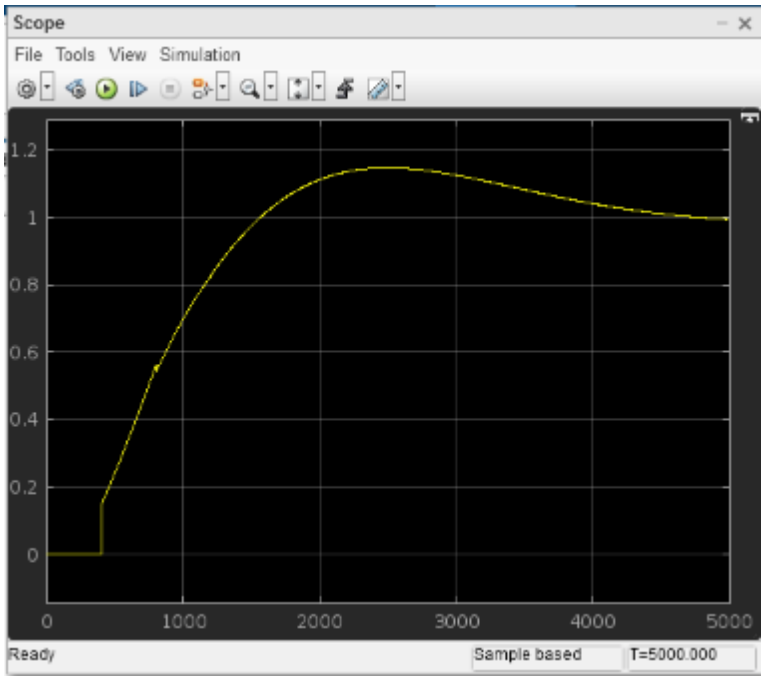


Figure 54: *Final curve output with final P, I, and D parameter values*

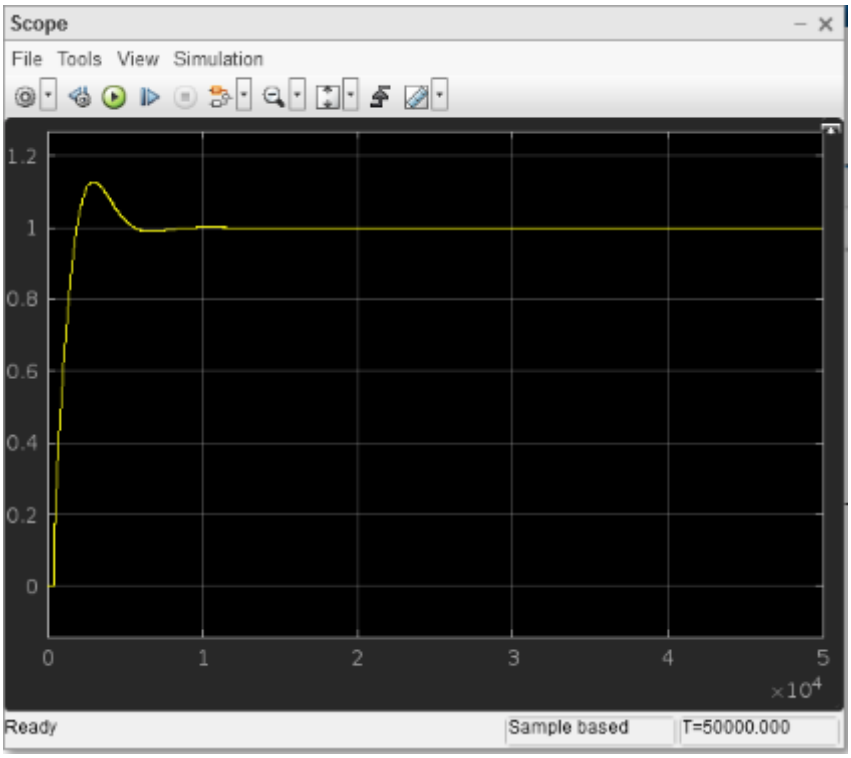


Figure 55: *Final curve output zoomed out view - curve stabilised at desired set point*

To verify our tuning results and parameter values, the Simulink auto-tuning function was used and compared with our results. **Figure 58** shows the final values after using the auto-tuning function. Our tuning methods were proved to be correct, and results were very close to the auto-tuning results. **Figure 59** shows the output curve of the auto-tuning function.

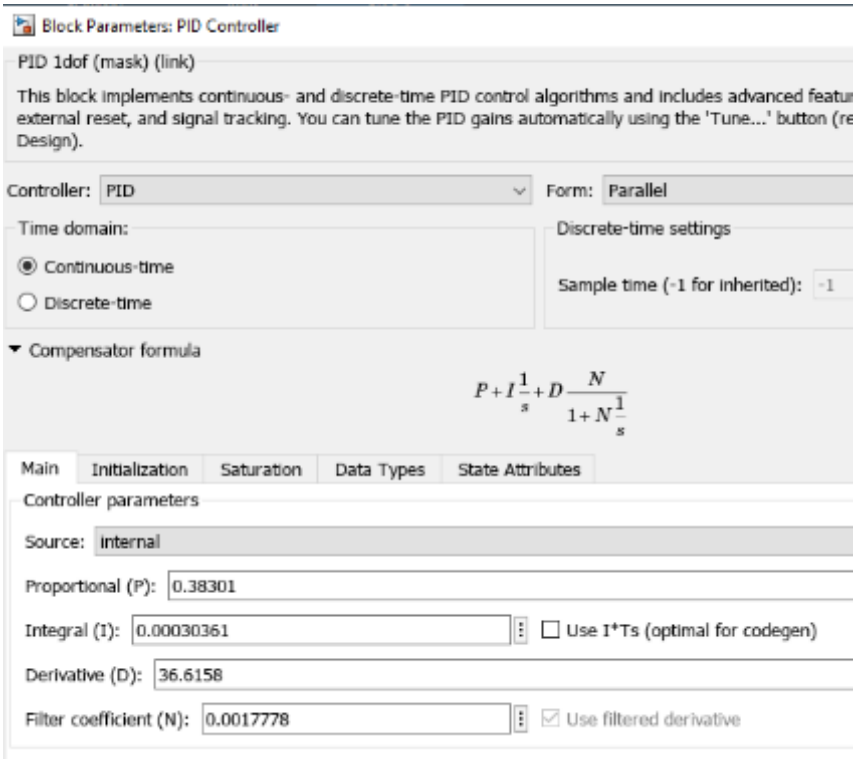


Figure 56: *Final parameter values using auto-tuning function in Simulink software*

Figure 57: *Output curve using auto-tuning function in Simulink software*

Table 6, below, shows the comparison of the parameter values from our manual tuning and the auto-tuning results. Both sets of values were in proximity of each other. When comparing the two curve outputs from these results, the manual tuning curve output (**Figure 56** and **Figure 57**) had an extra dip below the desired set point before stabilising while the auto-tuning curve output (**Figure 59**) was smoother in settling directly towards the desired set point. Hence, for the next steps of developing the PID controller, the auto-tuning parameter values (highlighted yellow in table 6) were used.

Table 7: *Parameter Value comparison - Manual Tuning v Auto-Tuning*

Parameter	Manual Tuning Values	Auto-Tuning Values
K_P	0.2629	0.3830
K_I	0.000318	0.000304
K_D	37.67	36.62

Once the auto-tuning values were selected, a small change was tested to see if the results observed until now could be further refined. To do this, the configuration of the solver used in auto-tuning was changed from ‘variable step’ to ‘fixed step’ with a step size of 0.1 as seen in **Figure 60**, below. However, this did not change anything in the output curve or the parameter values. The tuning function was then reverted to ‘variable step’ for continuity. Consequently, the first iteration of auto-tuning results i.e., auto-tuning parameters in Table 6 were confirmed to be used in the next process of controller development which was discretisation of the controller.

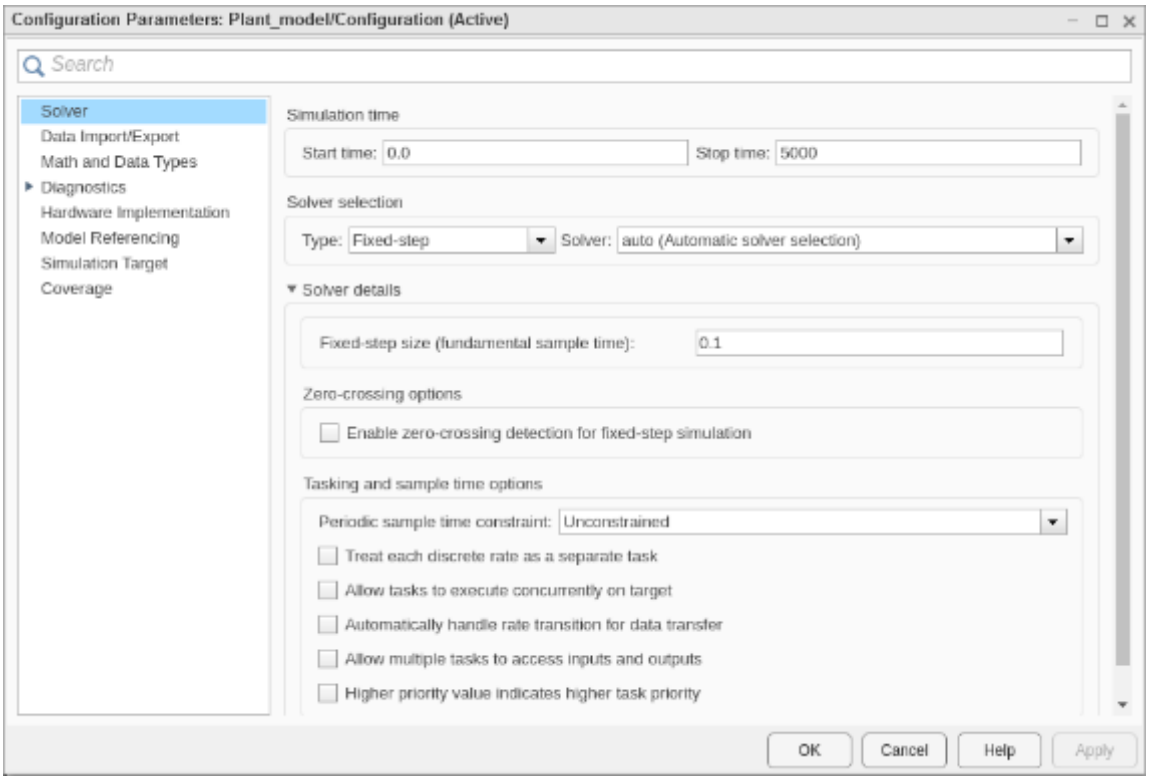


Figure 58: *Solver Configuration changed from 'variable step' to 'fixed step' with a step size of 0.1*

Figure 61, below, shows a summary of the PID parameter values and the performance specifications of the final output curve. In particular, the Rise Time was 722 seconds, Settling Time was 1860 seconds, the output curve had an Overshoot of 2.03 % and the curve was a Stable Closed-loop system. 'N' was the Filter Coefficient of the PID controller.

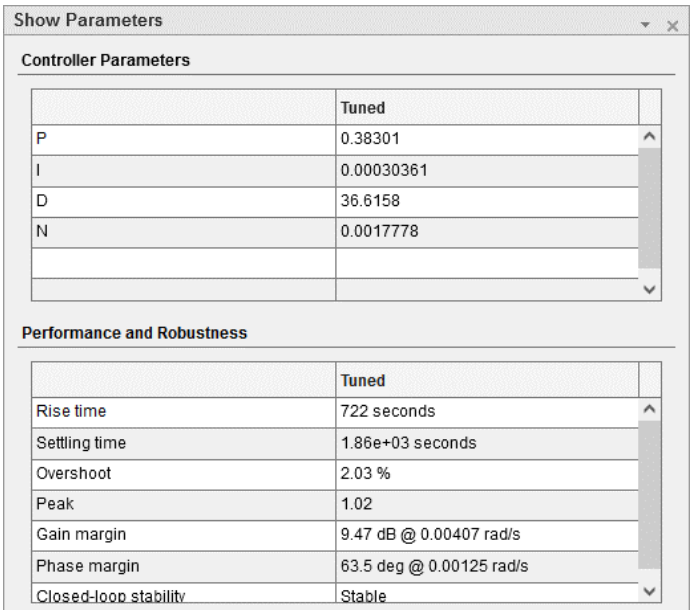


Figure 59: *Summary of PID Parameter Values and Performance & Robustness Values*

To discretise the PID Controller, an option in the MATLAB & Simulink software was used and the compensator formula was converted into a discrete model formula. **Figure 62**, below, shows the change in the controller to 'discrete time' (red circle 1) with the parameters kept the same (red circle 2) along with the compensator formula (yellow box).

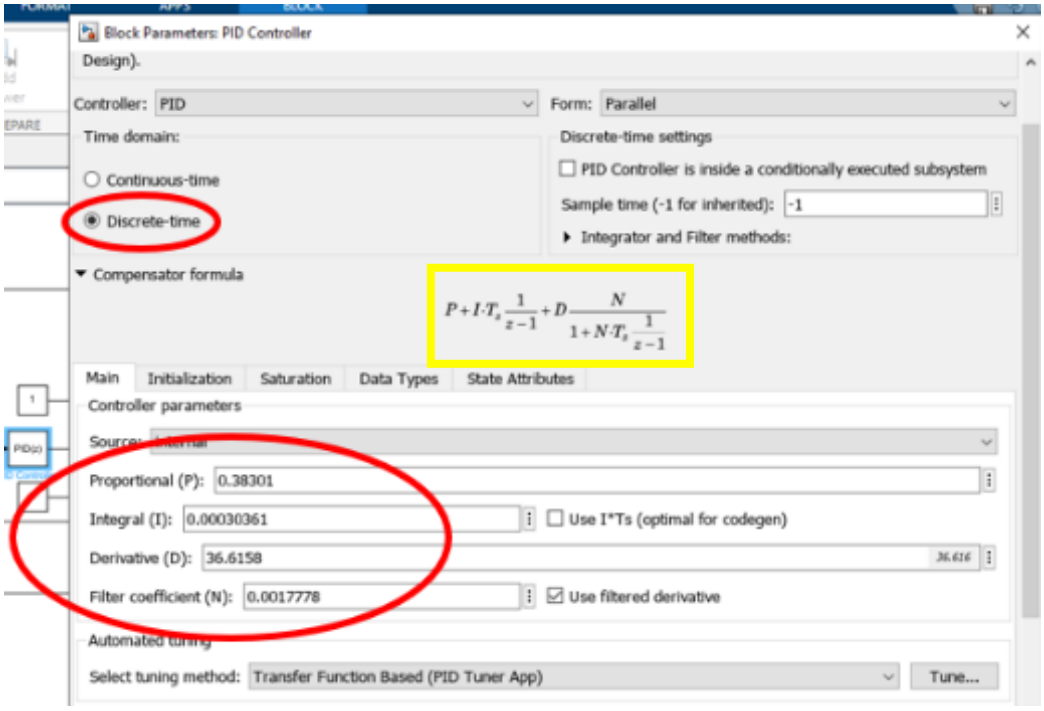


Figure 60: *Change of controller to discrete time while PID values are kept same*

The last part of designing the PID Controller involved converting the discrete compensator formula (shown in yellow box – **Figure 62**) to a difference equation that can be implemented in software coding. The steps from here onwards were adapted from online tutorial resources [83] [84].

Our parameters are:

$$P\ (K_P) = 0.38301 \quad I\ (K_I) = 0.00030361 \quad D\ (K_D) = 36.6158 \quad N = 0.0017778$$

The compensator equation can be simplified to the following:

$$G(S) = \frac{U(S)}{E(S)} = K_P + K_I \frac{1}{s} + \frac{K_D s}{\tau s + 1}$$

$$= 0.38301 + 0.00030361 \frac{1}{s} + \frac{36.6158 s}{562.4929688 s + 1} \qquad (7)$$

where τ (filter coefficient) $= \frac{1}{N} = \frac{1}{0.0017778} = 562.4929688$

The Tustin Transform is a common method to get the z-transform i.e., transforming the equation from s-domain to z-domain. The Tustin Transform involves substituting all the 'S's with the following term:

$$S = \frac{2}{T} \frac{z - 1}{z + 1} \qquad (8)$$

Note: T is the sample time; also on a theoretical basis, multiplying by z is equivalent to a time shift.

After carrying out all algebraic simplification, substitution, and rearranging, we can split the P, I and D portions of the difference equation and obtain the following equations:

$$p[n] = K_p \times e[n] \quad \textit{Proportional Term} \qquad (9)$$

$$i[n] = K_i \times \frac{T}{2} (e[n] + e[n-1]) + i[n-1] \quad \textit{Integral Term} \qquad (10)$$

$$d[n] = \frac{2 \times K_d}{(2 \times \tau) + T} (e[n] - e[n-1]) + \left(\frac{(2 \times \tau) - T}{(2 \times \tau) + T} \times d[n-1] \right) \qquad (11)$$

$$\textit{Derivative Term}$$

Note: e[n] is the error i.e., difference between set point and current temperature e[n-1] is the error of the previous iteration, [i-1] is the previous integral term, d[n-1] is the previous derivative term.

The final PID controller value is the sum of the three difference equations:

$$PID = p[n] + i[n] + d[n] \qquad (12)$$

This was then implemented in Python coding. The process of integrating all the aspects is described in **Section 3.6** (System Integration).

3.6 System Integration

As mentioned in sections 3.5 (PID Controller Development), the mechanical and electronic aspects were already integrated into the enclosure with the model ready for further controller and software integration. It is important to note that the software integration was for both microcontrollers files, while the controller integration was for the model microcontroller only.

The first integration was the process of combining the 'publisher' and 'subscriber' nodes into one node so that only one node needed to be started in each file (i.e., one node for each microcontroller). This reduced time and implementation effort as only one node needed to be started each rather than two. **Figure 63**, below, shows the code block of the node once 'publisher' and 'subscriber' were integrated in the model microcontroller. This is the same code as shown in **Section 3.4** (Software Set Up), with only an integration and a different topic name for 'publishing' – 'Client_PUB' and 'subscribing' – 'Server_PUB'. In the coding files for the user microcontroller, the topics were interchanged i.e., 'publishing' – 'Server_PUB' and 'subscribing' – 'Client_PUB'.

```
# ------------------------------------------------------
# Below: Publisher and Subscriber
class Pub(Node):
    def __init__(self):
        super().__init__('Data')
        self.publisher_ = self.create_publisher(
            String,
            'Client_PUB',
            0)
        timer_period = 0.5
        self.timer = self.create_timer(timer_period, self.timer_callback)
        self.i = 0

        self.subscription = self.create_subscription(
                String,
                'Server_PUB',
                self.listener_callback,
                10)
        self.subscription

    def listener_callback(self, msg):
        global set_point
        self.get_logger().info("{}".format(
            msg.data))
        set_point = msg.data

    def timer_callback(self):
        global set_point
        msg = String()
        msg.data = start(int(set_point))
        self.publisher_.publish(msg)
# ------------------------------------------------------
```

Figure 61: *Publisher and Subscriber Nodes merged. The topics were swapped in the other microcontroller*

The ‘publisher’ and ‘subscriber’ on both ends of the system were now integrated and communicating with each other. **Figure 64**, below, shows a diagram of the Ros Nodes and Ros Topics structure that was set up here. The user inputs a set point which is published to the physical model using the ‘Server_PUB’ topic, while the physical model publishes the current temperature using the ‘Client_PUB’ topic and that is then displayed on the website.

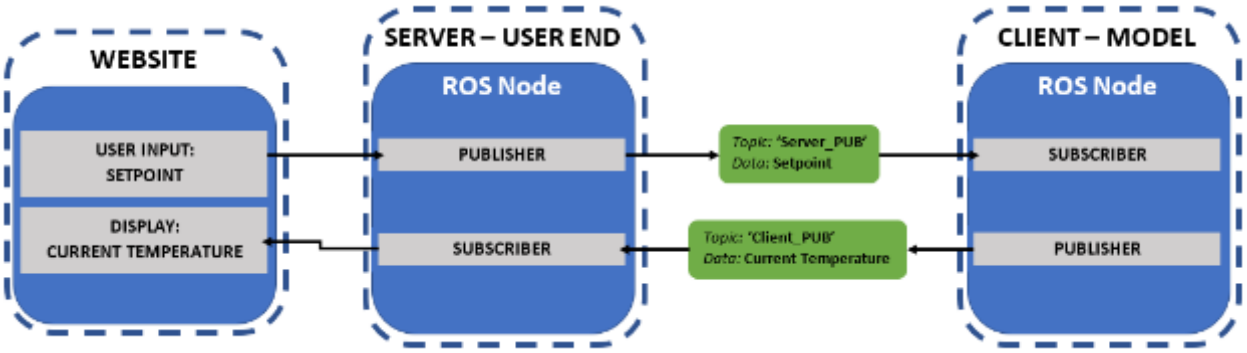


Figure 62: *ROS Nodes and Topic Overall Structure*

The second part of the integration process was integrating the Flask web application code, discussed in **Section 3.4**, with the publisher and subscriber code. This was required on the user end (Server) only. The integration was straightforward as the functions were inserted and were only required to be called from the main function. Using global variables, the set point and current temperature were passed to the publisher and received from the subscriber. As the Flask web application and Publisher & Subscriber were required to run simultaneously and continuously, a ‘thread’ was used so that the one of the threads, in our case it was the Publisher & Subscriber, could work in the background while the main thread, Flask web application, was running. Before the main thread was terminated, the two threads were joined. This way the tasks performed concurrently and saved run-time. The full code for the user end i.e., the server end, is shown below:

```
import rclpy
from rclpy.node import Node
from std_msgs.msg import String
import threading
from flask import Flask, render_template, flash, request
app = Flask(__name__)
curr_temp = 0
setpoint = 0

class Pub(Node):
    def __init__(self):
        super().__init__('pub')
        self.publisher_ = self.create_publisher(
            String,
            'Server_PUB',
            10)
        timer_period = 0.5
        self.timer = self.create_timer(timer_period, self.timer_callback)
```

```
        self.i = 0

        self.subscription = self.create_subscription(
            String,
            'Client_PUB',
            self.listener_callback,
            10)
        self.subscription
    def listener_callback(self, msg):
        global curr_temp
        self.get_logger().info("{}".format(
            msg.data))
        curr_temp = msg.data
    def timer_callback(self):
        global setpoint
        msg = String()
        msg.data = str(setpoint)
        self.publisher_.publish(msg)
@app.route("/", methods = ['GET'])
def Current_temp():
    return f'''
    <!DOCTYPE html>
    <html lang="en">
        <head>
            <meta charset="UTF-8">
            <title>Vertical Farm Temperature Control</title>
        </head>
        <body>
            <h1>Vertical Farm Temperature Control</h1>
            <form method = "GET"
                action="Set Temperature">
                <label for="Set Temperature">Enter Temperature</label>
                <input type="text" name="Set Temperature" value="" required>
                <input type="Submit" value="Set">
            </form>
        </body>
    </html>'''

@app.route("/Set Temperature")
def current_temp():
    global setpoint, curr_temp
    if request.args.get("Set Temperature") is not None:
        setpoint = request.args.get("Set Temperature")
    return f'''
    <!DOCTYPE html>
    <html lang="en">
        <head>
            <meta charset="UTF-8">
```

```
        <title>Vertical Farm Temperature Control</title>
    </head>
    <body>
        <h1>Vertical Farm Temperature Control</h1>
        <form action="Set Temperature" style='float:left;'>
            <label for="Set Temperature">Enter Temperature</label>
            <input type="number" name="Set Temperature" value="" required>
            <input type="Submit" value="Set">
        </form>
        <form action="Set Temperature">
            <input type="Submit" value="Refresh">
        </form>
        <br>
        <h3>The temperature has been set to {setpoint}.</h3>
        <br>
        <h3>The current temperature is: {curr_temp} &#8451</h3>
    </body>
</html>'''

def main(args=None):
    rclpy.init(args=args)
    pub = Pub()
    x = threading.Thread(target=rclpy.spin, args=[pub], daemon=True)
    x.start()
    app.run()
    pub.destroy_node()
    rclpy.shutdown()
    x.join()

if __name__ == '__main__':
    main()
```

The third integration process was to combine the Publisher & Subscriber code on the model end i.e., client end, with the PID Controller code (discussed in **Section 3.5** – PID Controller Development). This required some rearranging; hence, separate functions were created for turning on LEDs, reading Temperature and the PID Controller. These were separate to the Publisher & Subscriber Node function. The controller code matches the code discussed in **Section 3.5**. To simulate daylight and night-time, a 12 hour period was used between 6am and 6pm to turn the lights on i.e., daytime, and the lights were turned off for the remaining time to simulate night-time. The full code for the model end i.e., the client end, is shown below. Like the user end code, global variables were set up to define the main controller parameters e.g., current, and previous values of K_P, K_I, and K_D, as well as the other controller variables such as the Sample Time, PID Threshold etc. The PID Threshold is the threshold value that was used to decide based on the controller output whether to turn the components on or off e.g., if the controller output was greater than the PID Threshold, then the heating element would turn on otherwise it would be turned off. This replicates the MATLAB & Simulink set up for experimentation which is discussed in **Section 3.7** (Final Experimental Methodology).

```
import rclpy, os, sys, glob, time
import RPi.GPIO as GPIO
from datetime import datetime

set_point = 0

GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO.setup(26, GPIO.OUT) # LEDs
GPIO.setup(27, GPIO.OUT) # Heater
GPIO.setup(22, GPIO.OUT) # Fans

os.system('modprobe w1-gpio')
os.system('modprobe w1-therm')
base_dir = '/sys/bus/w1/devices/'
device_folder = glob.glob(base_dir + '*24c5')[0]
device_file = device_folder + '/w1_slave'

from rclpy.node import Node
from std_msgs.msg import String

# ------------------------------------------------------
# Below: PID Global Variables

LED_Status = False
Kp = 0.38301
Ki = 0.00030361
Kd = 36.6158
prev_proportional = 0
prev_integral = 0
prev_derivative = 0
Sample_Time = 1
Components_on = False
Tau = 562.4929688
PID_Threshold = 0
prev_Error = 0
prev_Temperature = 0

# ------------------------------------------------------
# Below: Temperature Sensor (ambient)

def read_temp_raw():
    f = open(device_file, 'r')
    lines = f.readlines()
    f.close()
    return lines

def read_temp():
```

```
    lines = read_temp_raw()

    while lines[0].strip()[-3:] != 'YES':
        time.sleep(0.2)
        lines = read_temp_raw()
    equals_pos = lines[1].find('t=')

    if equals_pos != -1:
        temp_string = lines[1][equals_pos+2:]
        temp_c = float(temp_string) / 1000.0
        temp_c = round(temp_c, 1)
    return temp_c2


def LEDs():
    global LED_Status
    now = datetime.now()
    current_time_total = now.strftime("%H: %M")
    current_time_hour = now.strftime("%H")
    current_time_minute = now.strftime("%M")
    if int(current_time_hour) >=18 or int(current_time_hour) < 6:
        if LED_Status == False:
            GPIO.output(26, GPIO.HIGH)      #LED HIGH
            LED_Status = True
    else:
        if LED_Status == True:
            GPIO.output(26, GPIO.LOW)                   #LED LOW
            LED_Status = False
    return

def PID_Controller(setpoint):
    global Kp, Ki, Kd, prev_proportional, prev_integral, prev_derivative,
           Sample_Time, Components_on, Tau, PID_Threshold, prev_Error,
           prev_Temperature
    current_Temperature = read_temp()
    Error = setpoint - current_Temperature
    proportional = Kp * Error
    integral = (Ki * Sample_Time * 0.5 * (Error + prev_Error)) + prev_integral
    derivative = ((((2 * Kd) / ((2*Tau) + Sample_Time)) * (Error –
    prev_Error)) +
    ((((2*Tau) - Sample_Time) / ((2*Tau) + Sample_Time)) * prev_derivative))
    PID_output = proportional + integral + derivative
    if PID_output >= 0:
        if Components_on == False:
            GPIO.output(27, GPIO.HIGH) #Heater
            GPIO.output(22, GPIO.HIGH) #Fan
            Components_on = True
```

```
    else:
        if Components_on == True:
            GPIO.output(27, GPIO.LOW) #Heater
            GPIO.output(22, GPIO.LOW) #Fan
            Components_on = False

    prev_Error = Error
    prev_Temperature = current_Temperature
    prev_proportional = proportional
    prev_integral = integral
    prev_derivative = derivative

    temp_final = str(current_Temperature)
    return temp_final
def start(setting_temperature):
    LEDs()
    return(PID_Controller(setting_temperature))
# ------------------------------------------------------
# Below: Publisher and Subscriber
class Pub(Node):
    def __init__(self):
        super().__init__('Data')
        self.publisher_ = self.create_publisher(
            String,
            'Client_PUB',
            0)
        timer_period = 0.5
        self.timer = self.create_timer(timer_period, self.timer_callback)
        self.i = 0

        self.subscription = self.create_subscription(
                String,
                'Server_PUB',
                self.listener_callback,
                10)
        self.subscription
    def listener_callback(self, msg):
        global set_point
        self.get_logger().info("{}".format(
            msg.data))
        set_point = msg.data

    def timer_callback(self):
        global set_point
        msg = String()
        msg.data = start(int(set_point))
        self.publisher_.publish(msg)
def main(args=None):
```

```
    rclpy.init(args=args)
    pub = Pub()
    rclpy.spin(pub)
    pub.destroy_node()
    rclpy.shutdown()

if __name__ == '__main__':
    main()

# ----------------------------------------------
```

3.7 Final Experimental Methodology

The methodology in our experiment was split into three parts. The aim was to obtain three different output curves of the temperature readings after the user enters a set point. This would test the PID Controller and the overall model including the enclosure. The first curve was the temperature readings of our physical model. The second curve was our transfer function using MATLAB & Simulink software when combined with our PID Controller and its parameters. The third curve was an output from a MathWorks 'House Heating System' [85]. The 'House Heating System' example is a model template created using Simulink software with a heating element, switch, thermostat, and a house network that stores key house details e.g., wall and roof thickness, wall and roof area, convection and conduction values etc. We treated our physical model as a 'house' and modified the Simulink model to match our physical system. This resulted in an output curve. The three output curves were then compared using details such as rise time, overshoot, settling time etc.

For the first curve, our physical model was run using the PID Controller within our software for 10 iterations to obtain and ensure consistent results. Due to ambient temperature being around 26℃, the set point was set as 30℃. During these 10 iterations, the Flask web application was yet to be developed; hence, the set point was directly entered into the user end code file. The rest of the process i.e., publishing the set point and receiving the current temperature as well as the PID control was already implemented and functioning correctly. The 11th iteration, which was run after integrating the Flask web application, was used to confirm that the full system including the Flask application is functioning correctly and the final iteration was also used as a validation that the final result was consistent to the previous 10 iterations.

For the second curve, our transfer function, described in **Section 3.5** (PID Controller Development), was set up along with our PID Controller block which included the same parameters that were used in our physical model. **Figure 65** shows the layout of the transfer function Simulink model. Each of the components used are highlighted – PID Controller block (Red), Transfer Function (Yellow), Transport Delay block (Purple), Constant (Blue), Step Input (Green). This simulation model assumes that the starting point is zero.

Figure 63: *Transfer Function Simulink Modelling & Components*

Our ambient temperature was around 26℃ (as mentioned previously); hence, an offset was added to the simulation to account for the ambient temperature difference. The Constant (Blue) and the summing junction in **Figure 65** achieves this off set. As our physical model noted a delay of 400 seconds before any temperature change, this was replicated in the simulation using a Transport Delay Block (Purple).

The figures below, show the values and parameters used for the main blocks part of the Transfer Function Simulink model shown in **Figure 65**. The box colours around each figure correlate with the boxes in **Figure 65**.

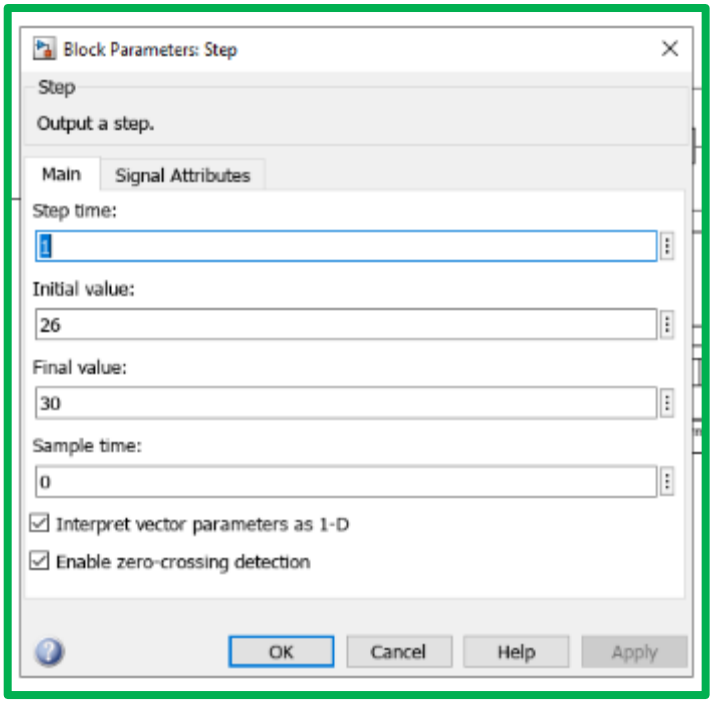


Figure 64: *Step Input Block - final value represents set point; initial value accounts for offset to match ambient temperature in physical model*

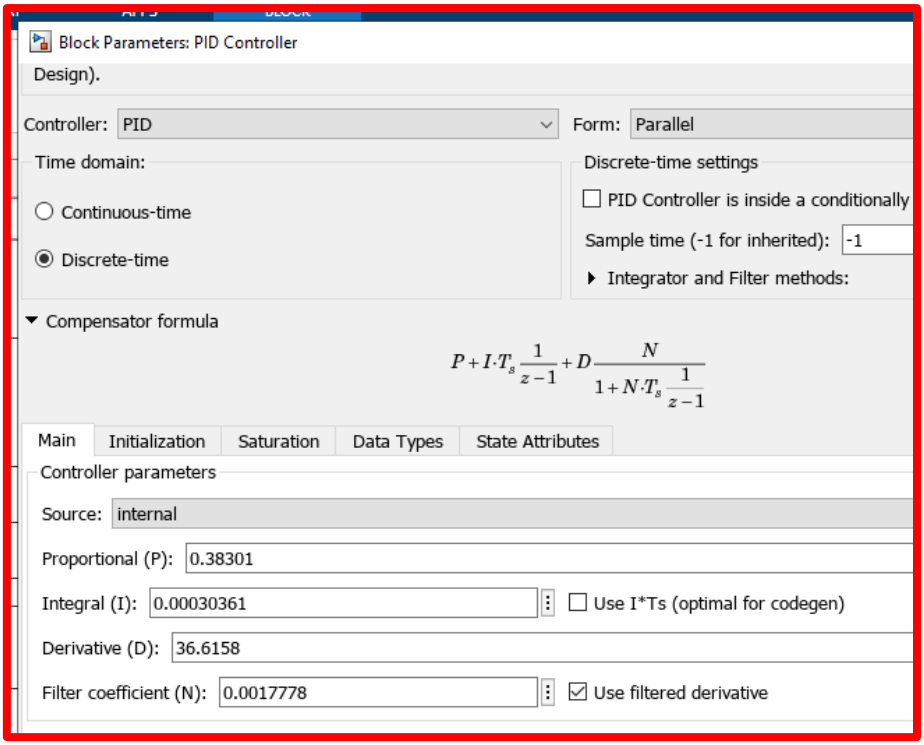


Figure 65: *PID Controller Block - discrete time PID controller with PID parameters matching the parameters used in the physical model and obtained from calucations (section 3.5)*

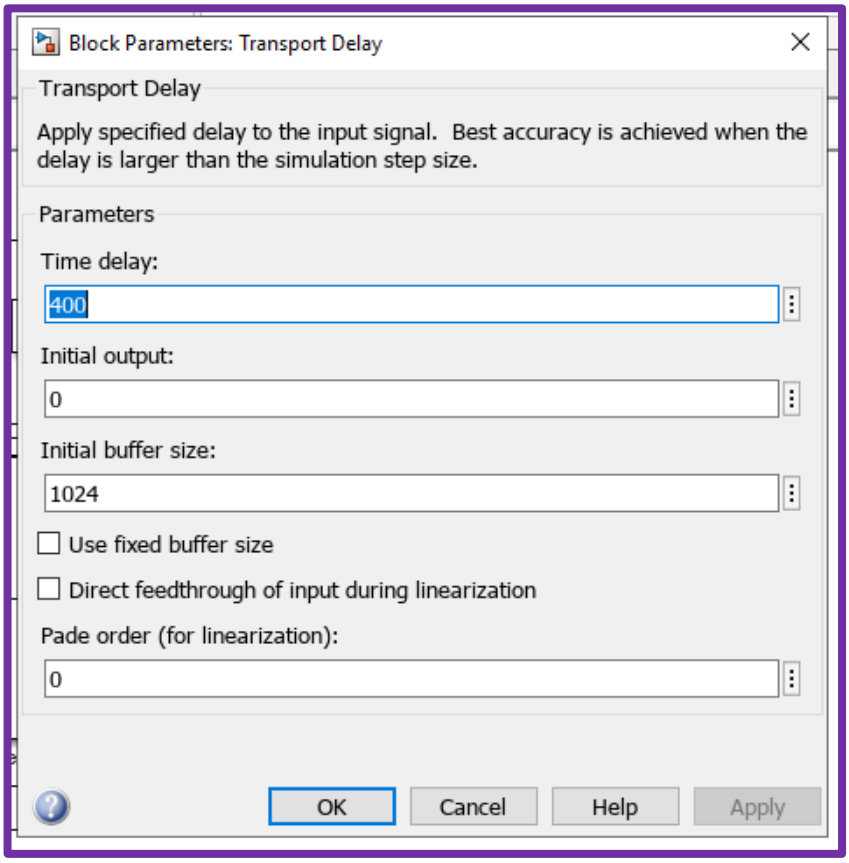


Figure 66: *Transport Delay Block - provides the 400 second delay to replicate the time before temperature change in the physical model*

With the set up complete, the Simulink model of the Transfer Function was run for one iteration (as this was a model simulation, the output was going to be the same in every iteration; hence, it was not run for multiple iterations like the physical model.

For the third curve, as mentioned above, a MathWorks 'House Heating System' [85] template was used as a model. The 'House Heating System' example is a model template created using Simulink software with a heating element, switch, thermostat, and a house network that stores key house details e.g., wall and roof thickness, wall and roof area, convection and conduction values etc. We treated our physical model as a 'house' and modified the Simulink model to match our physical system. **Figure 69**, below, shows the layout of the model blocks with each key component highlighted and explained further below. Key components: Step Input (Green), PID Controller block (Red), Threshold (Yellow), Heater (Orange), House (Blue), Constant 1 (Black), Constant 2 (Brown).

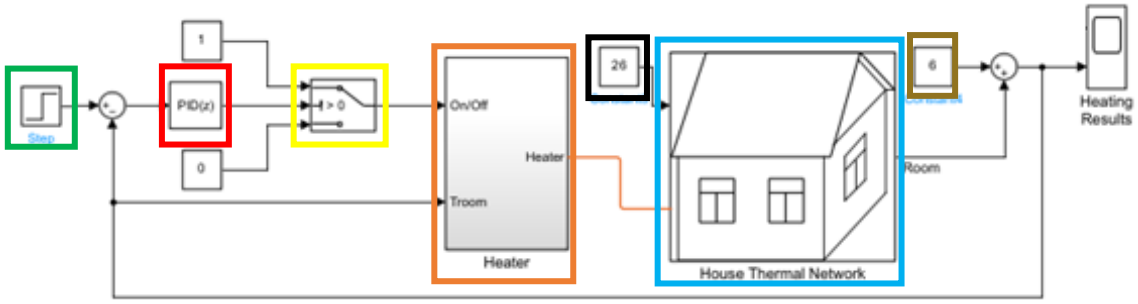


Figure 67: *'House Heating System' Simulink model layout with key components highlighted*

The Step Input block shown in **Figure 69** (green), had the same parameters set as shown in **Figure 66**, above. Similarly, the PID Controller block had the same parameters as shown in **Figure 67**, above. The Threshold block's (yellow in **Figure 69**) parameters are shown below in **Figure 70**. The block carried out the task of a switch. If the PID Controller's output was greater than zero, the heater was sent a '1' i.e., 'turn on' signal and sent a '0' i.e., 'turn off' if the PID Controller's output was lesser than zero. This replicates the Threshold value of zero in our software code shown and described in **Section 3.6** (System Integration).

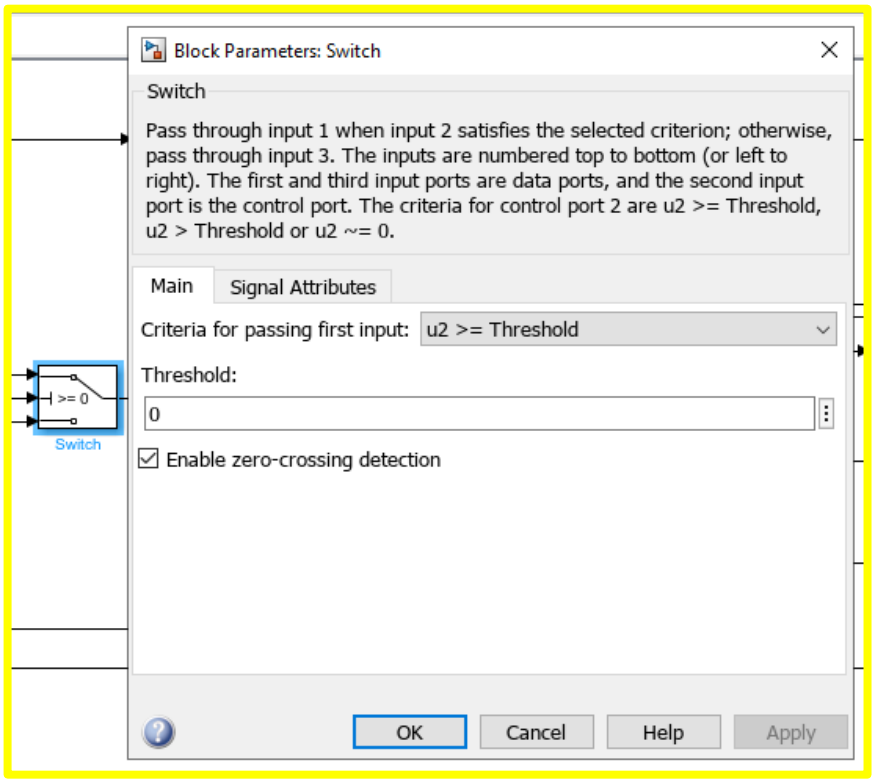


Figure 68: *Threshold block and Parameters*

The heater block (orange in **Figure 69**) sets out the transfer function of the system and the heater's air temperature. The heater air temperature was set to equal the air temperature of the heater used in our physical model – in our case this was 30℃. The transfer function matches our transfer function described earlier in this section. **Figure 71**, below, shows the heater specifications in Simulink software.

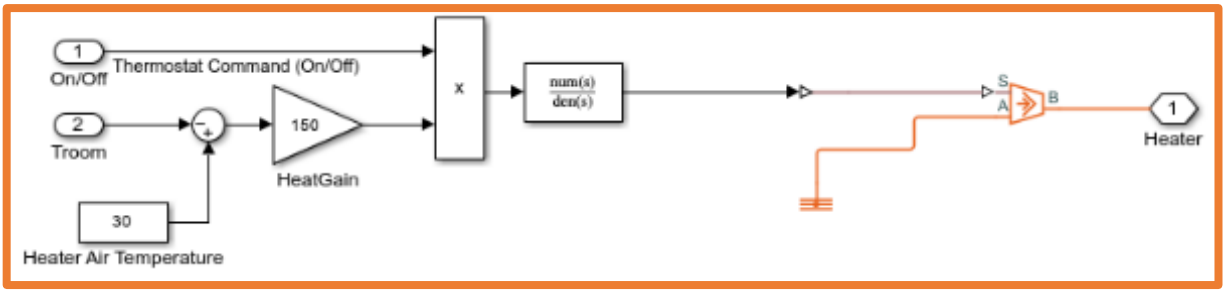


Figure 69: *Heater Block and Parameters*

The first constant (black box in **Figure 69**) was used as the offset discussed earlier to ensure the start point matches our physical model with a starting temperature of 26℃. While the parameters of the 'House Heating System' could be altered to match our system, there was still another offset which couldn't be tracked in the framework of the 'off-the-shelf' Simulink model. Hence, to combat this, another constant (brown box in **Figure 69**) was added into the structure. After trialling and testing, it was observed that this offset was corrected when a constant of '6' was added into the system.

The biggest component to be adjusted in this model was the 'House' component as the specifications needed to be matched to our physical model. **Figure 72**, below, shows the parameters that were required. Specifications from our physical model were used to calculate all the parameters. The specifications required included length, width, and height to calculate roof and wall area, roof, and wall thickness etc. Using our models material i.e., Acrylic, material properties were also researched and calculated e.g., thermal conductivity, thermal mass etc.

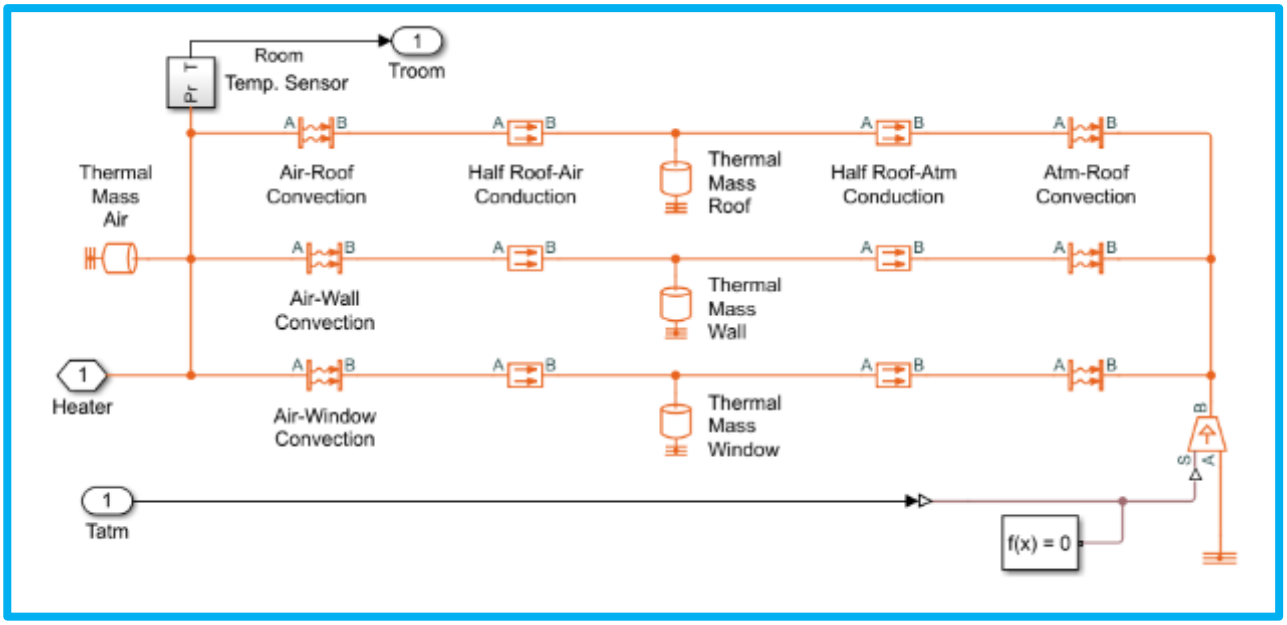


Figure 70: *House Block and Parameters - all values were calculated using our physical models specifications*

With the set up complete, the Simulink model of the 'House Heating System' was run for one iteration (as this was also a model simulation, the output was going to be the same in every iteration; hence, it was not run for multiple iterations like the physical model.

Using the three different experimental outputs a comparison was carried out of the three different curves and their details such as rise time, overshoot, settling time etc. Using these results, we verified that our automation of Controlled Environment Agriculture was a viable method for implementing at a larger scale, at industrial level. The results are shown in the following **Section 4.0** and discussed in **Section 5.0**.

4.0 Results

This chapter presents the results that were obtained from the implemented methodology discussed previously. Each step and process is detailed, and every iteration is explained through images and graphs. The data obtained is also analysed and results are also discussed and compared to provide an accurate validation to prove the experiment successful.

The Flask Web Application was implemented and run through the terminals in Visual Studio Code – one terminal for user end and one for model end. **Figure 73**, shows the output in terminals once run.

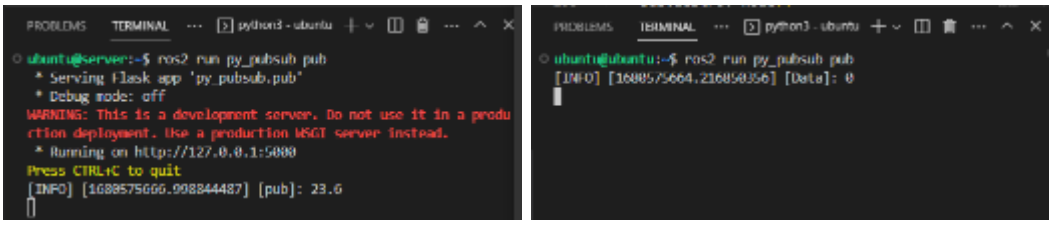


Figure 71: *Visual Studio Code Terminal Outputs when initiated*

The initial set point and current temperature was transmitted to the two sub-systems and the home page of the web application was opened – shown in **Figure 74**, below.

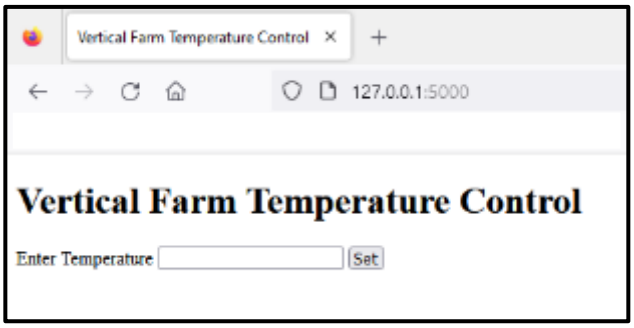


Figure 72: *Home page of Flask Web Application*

The output of entering the set point (30℃)in the input field on the web application is shown in **Figure 75**, below.

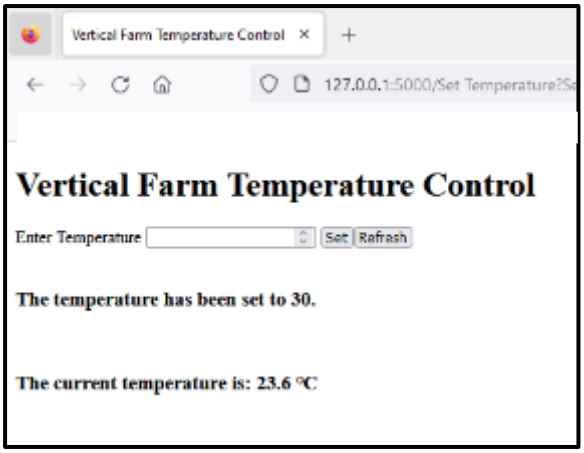


Figure 73: *Flask Web Application page after entering set point (30℃)*

Before setting the set point to 30℃, a dummy temperature of 15℃ was set to test the updating of terminal outputs once temperature was changed. As seen in **Figure 76**, below, the user end (server) was receiving the current temperature (in the figure below ~23.7℃) and the model end (client) was receiving a temperature of 30℃. As seen in the figure, the set point was changed from 15℃ to 30℃ to test that the system wass functioning as desired.

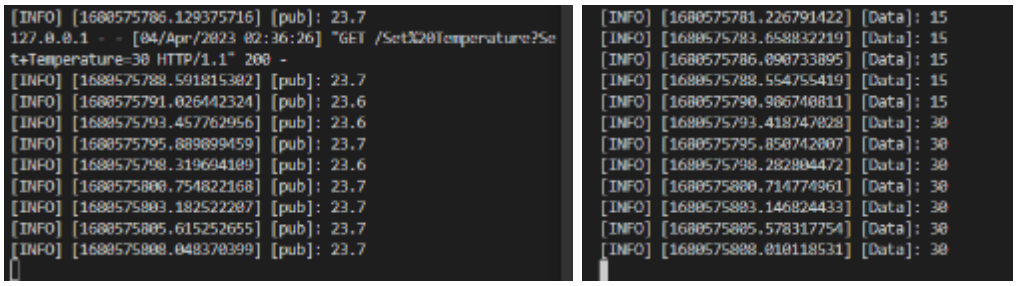


Figure 74: *Terminal Outputs (one on left, other on right) after temperature was set, including change in set point*

Figure 77, below, shows the Flask Web Application output after the web app was refreshed. This was done to ensure that the website was indeed updating after 'refreshing' and showing any new data. The figure, below, shows the current temperature as 23.7℃ which was an updated temperature reading as the previous reading was 23.6℃, as seen in **Figure 75**.

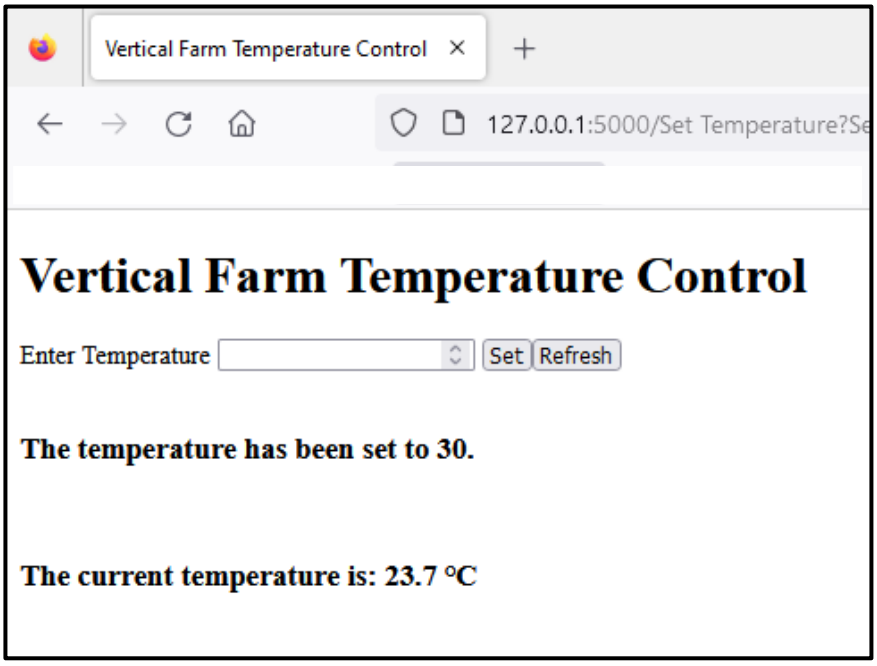


Figure 75: *Flask Web Application Output after 'refreshing' the webpage*

As discussed in **Section 3.7** (final experimental methodology), all the curve outputs (the 10 iterations and the final iteration (with Flask Web App) along with the two Simulink models) are shown in the figures below. Using these figures, key characteristic details such as the rise time, settling time, overshoot and steady state error were calculated and are tabulated under each iteration. All data has been summarised in **Error! Reference source not found.**.

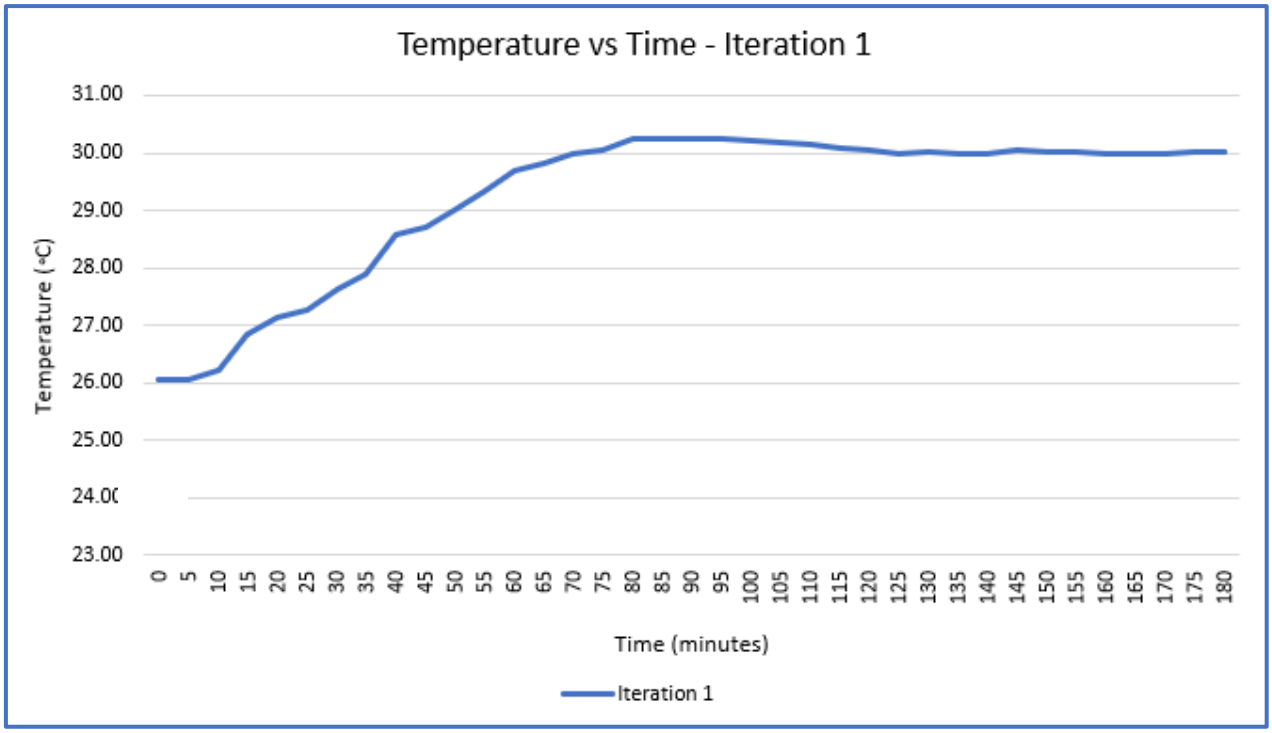


Figure 76: *Iteration 1 - Physical Model Experimentation Result*

Figure 78, above, shows the first experiment iteration that was carried out. Using our physical model set up, a set point of 30℃ was entered. The key data obtained from this experiment was as below.

Data	Value	
Initial Temperature (℃)	26.06	
Peak Temperature (℃)	30.24	
Rise Time (Minutes)	47.50	
Settling Time (Minutes)	115	
Overshoot (%)	0.80	
Steady State Error (℃)	0.01	
Initial Delay (Seconds | Minutes)	400	6.67

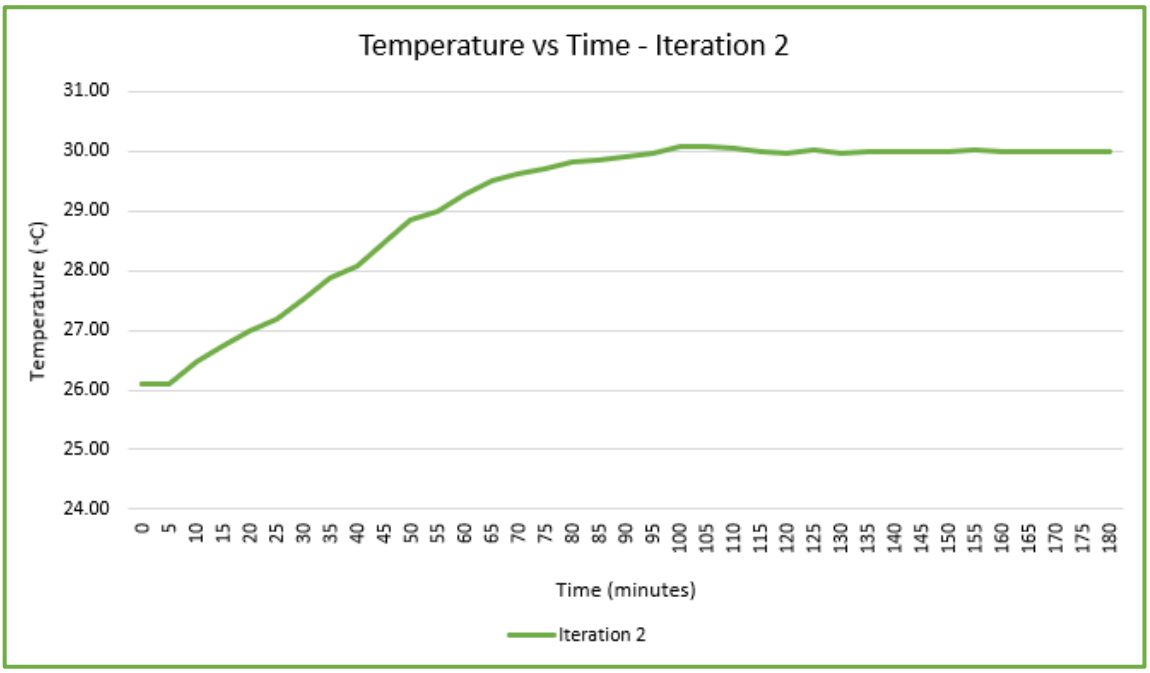


Figure 77: *Iteration 2 - Physical Model Experimentation Result*

Figure 79, above, shows the second experiment iteration that was carried out. The same setpoint of 30℃ was used and the key data obtained from this experiment was as below.

Data	Value
Initial Temperature (℃)	26.10
Peak Temperature (℃)	30.09
Rise Time (Minutes)	60
Settling Time (Minutes)	103.50
Overshoot (%)	0.30
Steady State Error (℃)	0.01
Initial Delay (Seconds | Minutes)	400 | 6.67

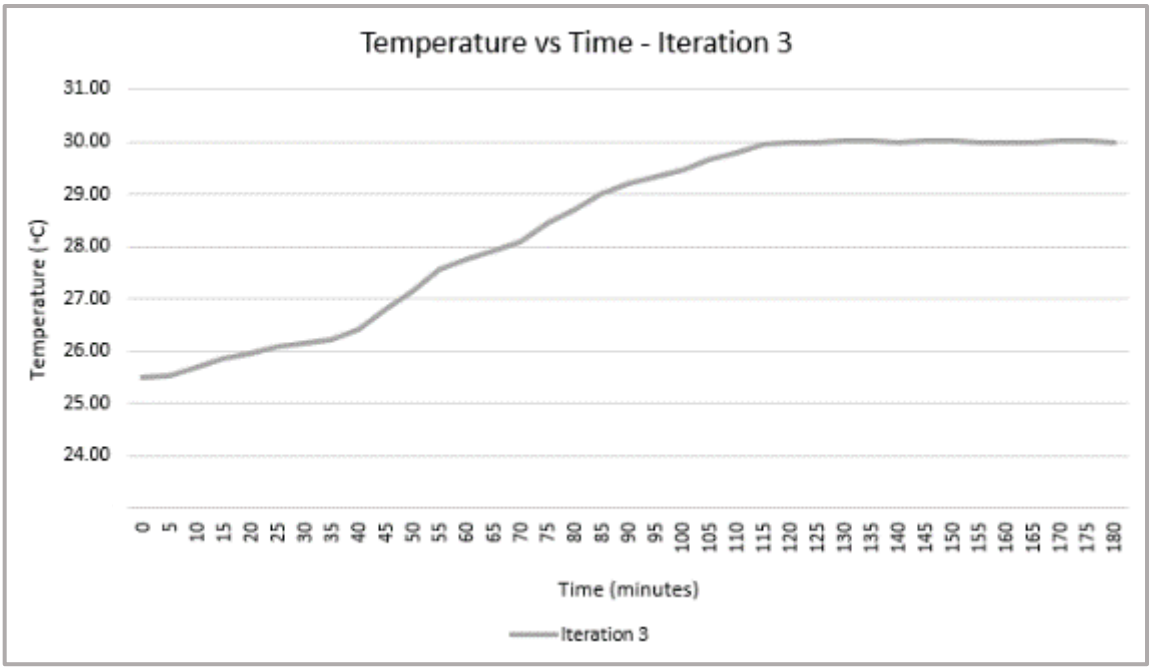


Figure 78: *Iteration 3 - Physical Model Experimentation Result*

Figure 80, above, shows the third experiment iteration that was carried out. Using the same setpoint as the previous iterations, the key data obtained was as below.

Data	Value	
Initial Temperature (℃)	25.50	
Peak Temperature (℃)	30.03	
Rise Time (Minutes)	82.5	
Settling Time (Minutes)	132.50	
Overshoot (%)	0.10	
Steady State Error (℃)	0	
Initial Delay (Seconds | Minutes)	400	6.67

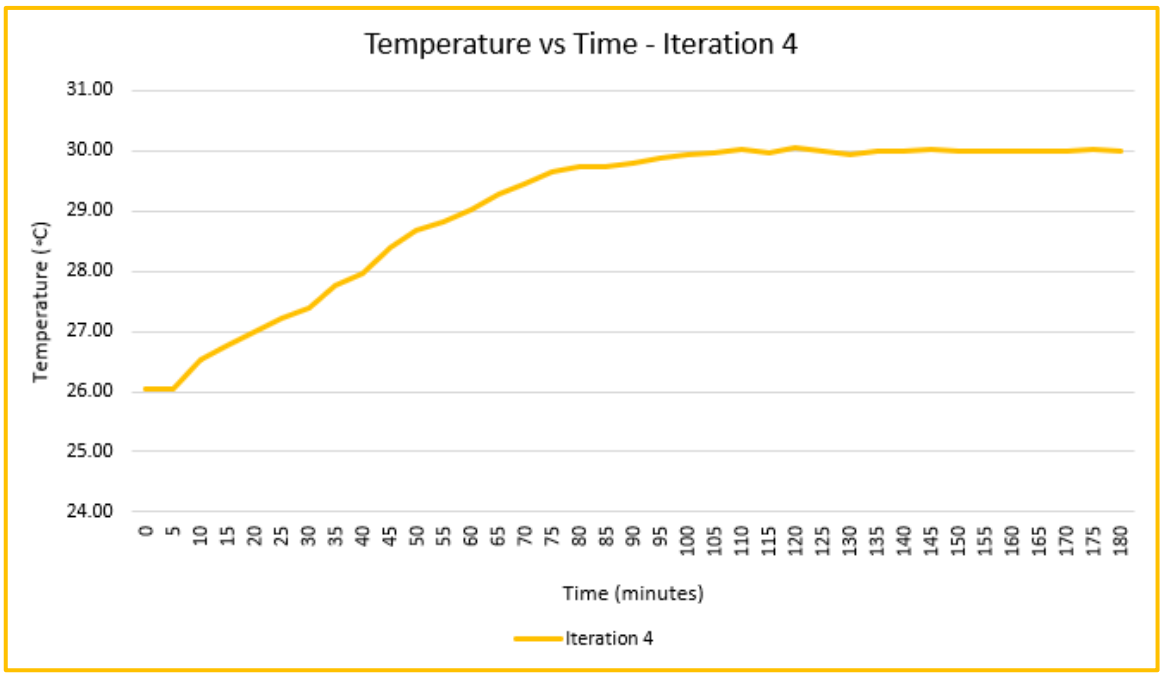


Figure 79: *Iteration 4 - Physical Model Experimentation Result*

Figure 81, above, shows the fourth experiment iteration that was carried out. Using the same setpoint as the previous iterations, the key data obtained was as below.

Data	Value
Initial Temperature (℃)	26.03
Peak Temperature (℃)	30.04
Rise Time (Minutes)	65
Settling Time (Minutes)	125
Overshoot (%)	0.13
Steady State Error (℃)	0.01
Initial Delay (Seconds | Minutes)	400 | 6.67

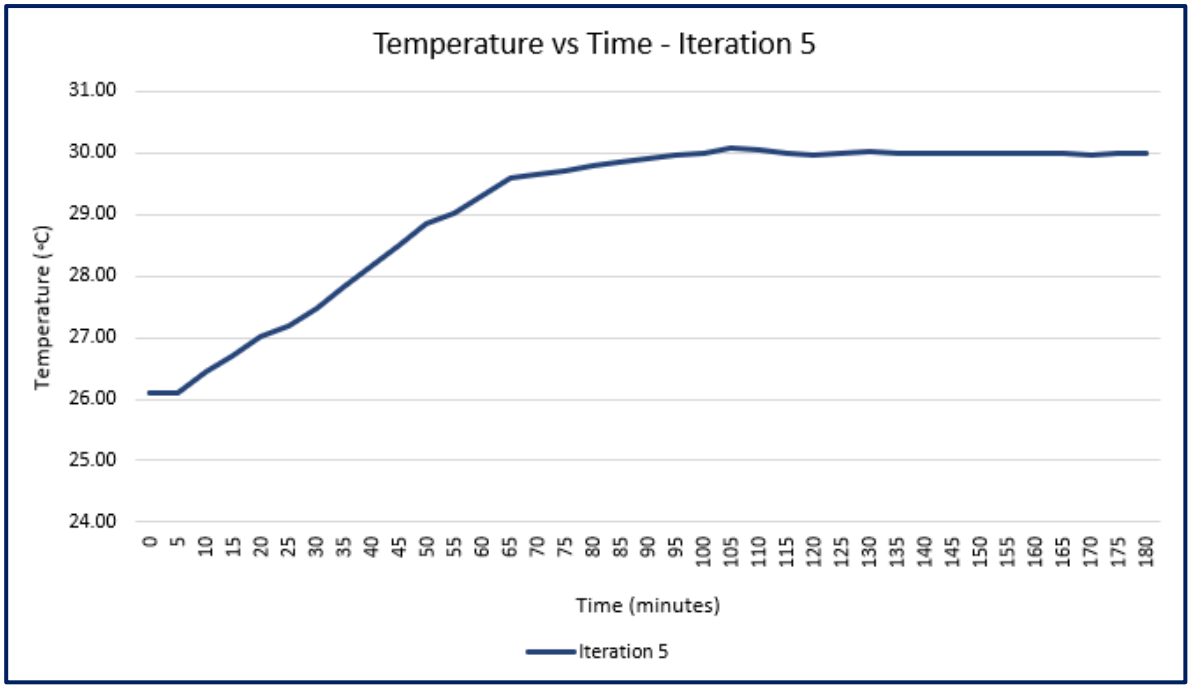


Figure 80: *Iteration 5 - Physical Model Experimentation Result*

Figure 82, above, shows the fifth experiment iteration that was carried out. Using the same setpoint as the previous iterations, the key data obtained was as below.

Data	Value	
Initial Temperature (℃)	26.09	
Peak Temperature (℃)	30.07	
Rise Time (Minutes)	55	
Settling Time (Minutes)	105	
Overshoot (%)	0.23	
Steady State Error (℃)	0	
Initial Delay (Seconds | Minutes)	400	6.67

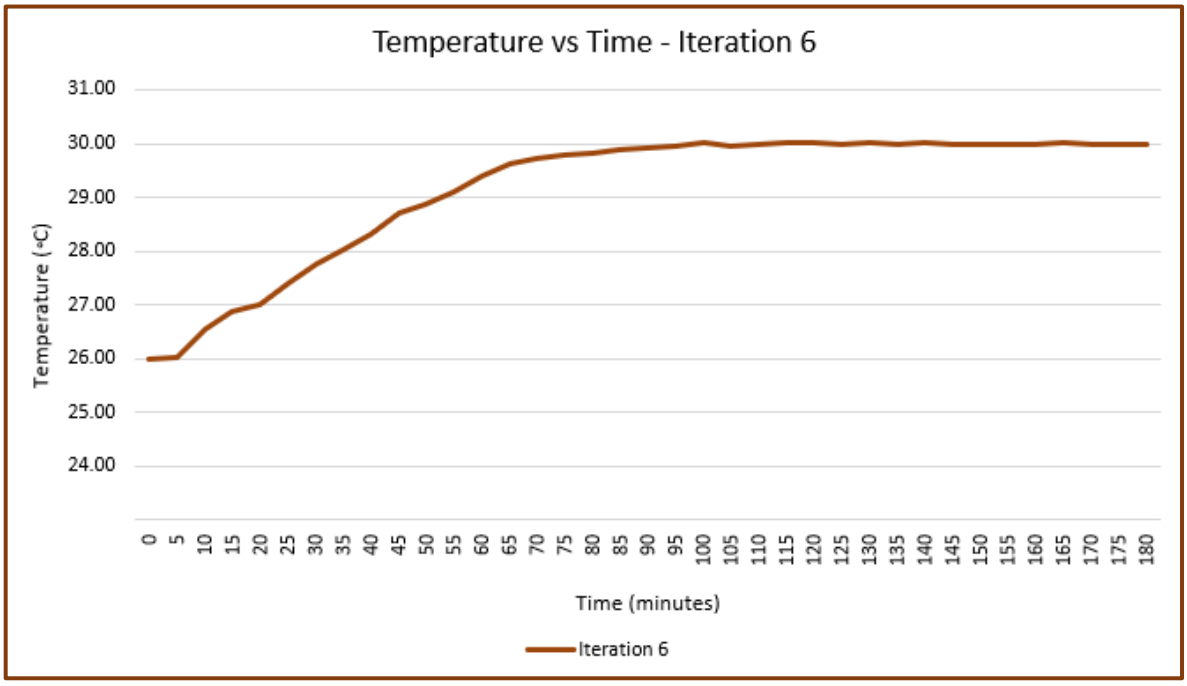


Figure 81: *Iteration 6 - Physical Model Experimentation Result*

Figure 83, above, shows the sixth experiment iteration that was carried out. Using the same setpoint as the previous iterations, the key data obtained was as below.

Data	Value	
Initial Temperature (℃)	26.00	
Peak Temperature (℃)	30.03	
Rise Time (Minutes)	57	
Settling Time (Minutes)	117.50	
Overshoot (%)	0.10	
Steady State Error (℃)	0.01	
Initial Delay (Seconds | Minutes)	400	6.67

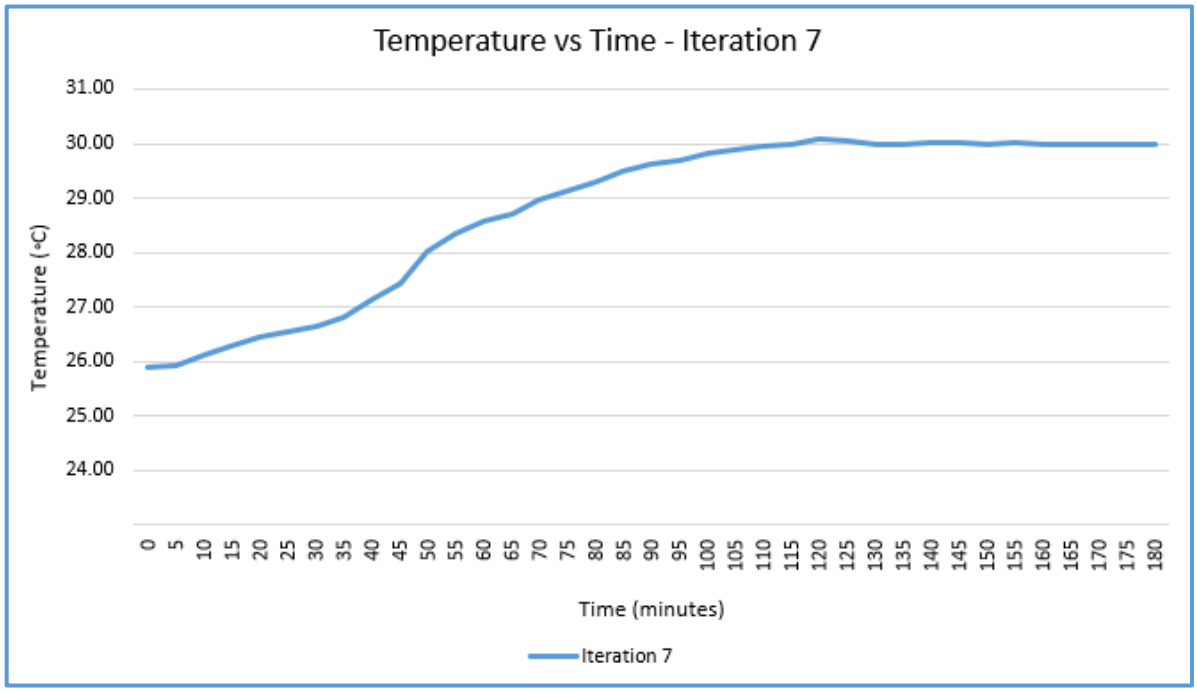


Figure 82: *Iteration 7 - Physical Model Experimentation Result*

Figure 84, above, shows the seventh experiment iteration that was carried out. Using the same setpoint as the previous iterations, the key data obtained was as below.

Data	Value
Initial Temperature (℃)	25.90
Peak Temperature (℃)	30.07
Rise Time (Minutes)	75
Settling Time (Minutes)	122.50
Overshoot (%)	0.23
Steady State Error (℃)	0.01
Initial Delay (Seconds | Minutes)	400 | 6.67

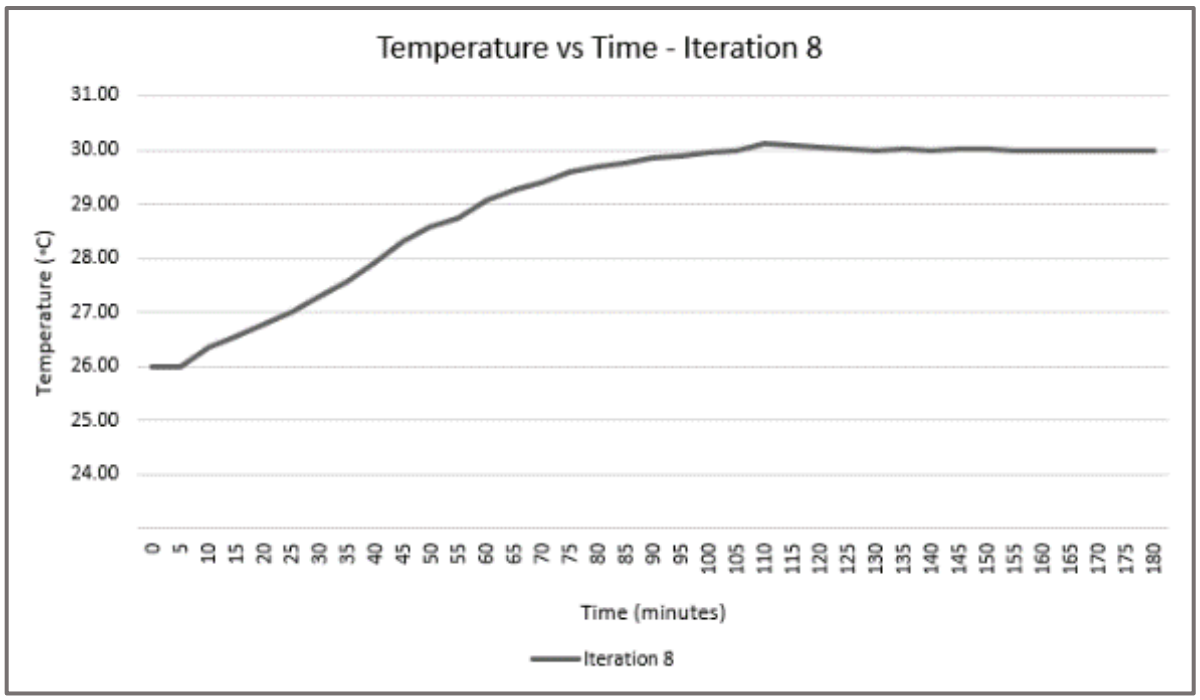


Figure 83: *Iteration 8 - Physical Model Experimentation Result*

Figure 85, above, shows the eighth experiment iteration that was carried out. Using the same setpoint as the previous iterations, the key data obtained was as below.

Data	Value	
Initial Temperature (℃)	25.98	
Peak Temperature (℃)	30.11	
Rise Time (Minutes)	65	
Settling Time (Minutes)	115	
Overshoot (%)	0.37	
Steady State Error (℃)	0	
Initial Delay (Seconds | Minutes)	400	6.67

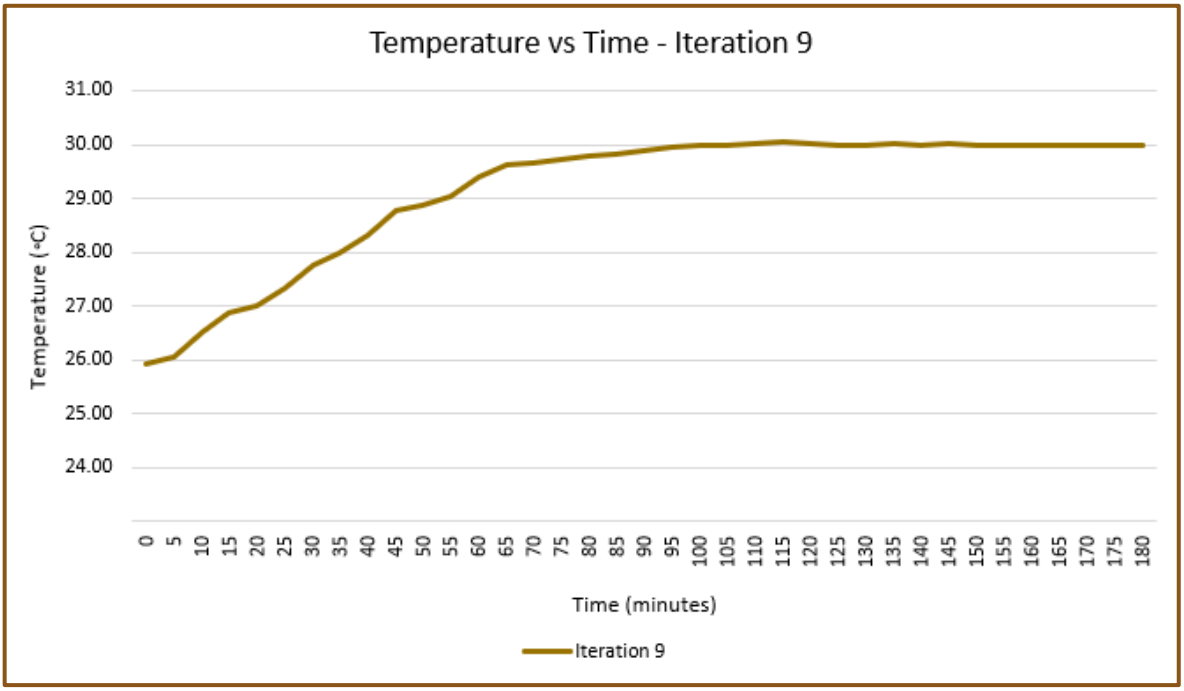


Figure 84: *Iteration 9 - Physical Model Experimentation Result*

Figure 86, above, shows the ninth experiment iteration that was carried out. Using the same setpoint as the previous iterations, the key data obtained was as below.

Data	Value	
Initial Temperature (℃)	25.93	
Peak Temperature (℃)	30.05	
Rise Time (Minutes)	57	
Settling Time (Minutes)	120	
Overshoot (%)	0.17	
Steady State Error (℃)	0	
Initial Delay (Seconds | Minutes)	400	6.67

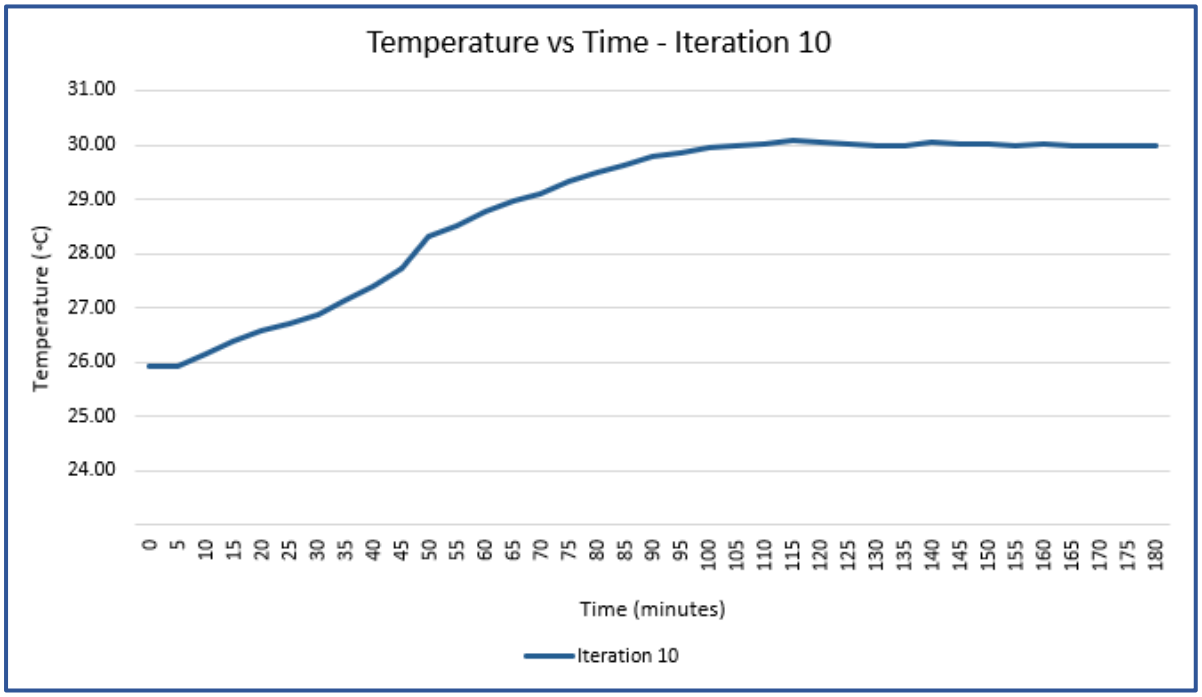


Figure 85: *Iteration 10 - Physical Model Experimentation Result*

Figure 87, above, shows the tenth experiment iteration that was carried out. Using the same setpoint as the previous iterations, the key data obtained was as below.

Data	Value
Initial Temperature (℃)	25.92
Peak Temperature (℃)	30.09
Rise Time (Minutes)	70
Settling Time (Minutes)	117.50
Overshoot (%)	0.30
Steady State Error (℃)	0
Initial Delay (Seconds | Minutes)	400 | 6.67

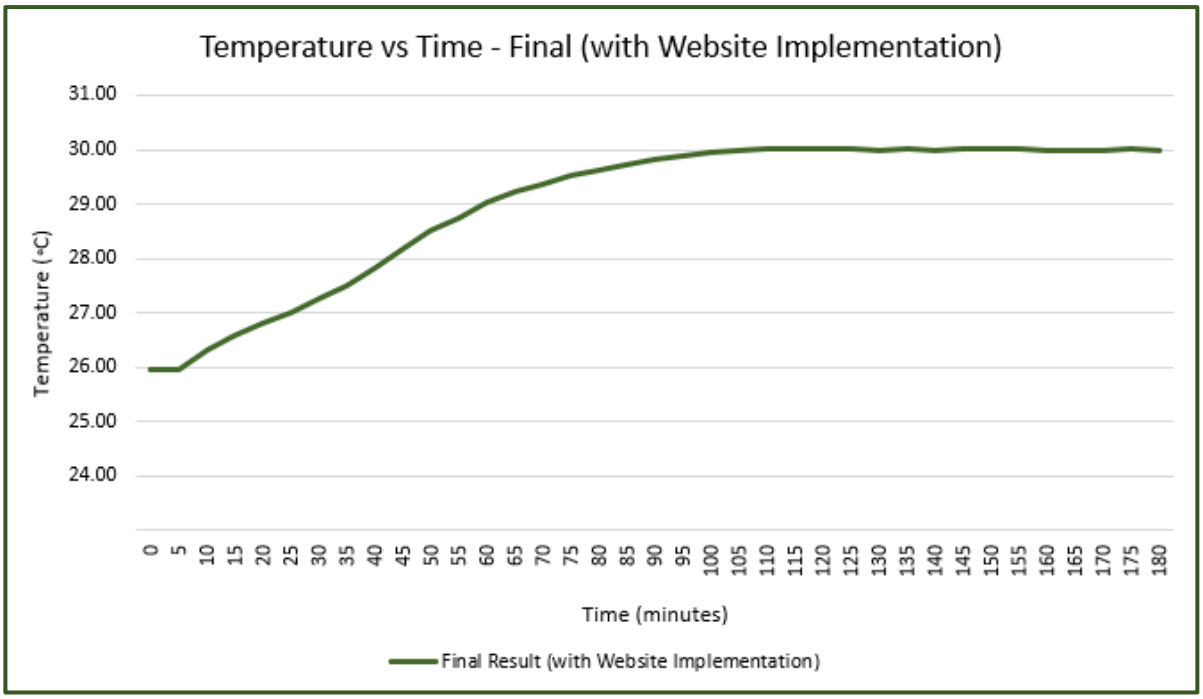


Figure 86: *Iteration 11 - Final Physical Model Experimentation Result (with Flask Web App)*

Figure 88, above, shows the eleventh experiment iteration that was carried out after the Flask Web App was implemented. Using the same setpoint as the previous iterations, the key data obtained was as below.

Data	Value	
Initial Temperature (℃)	25.95	
Peak Temperature (℃)	30.03	
Rise Time (Minutes)	69	
Settling Time (Minutes)	120	
Overshoot (%)	0.08	
Steady State Error (℃)	0.02	
Initial Delay (Seconds | Minutes)	400	6.67

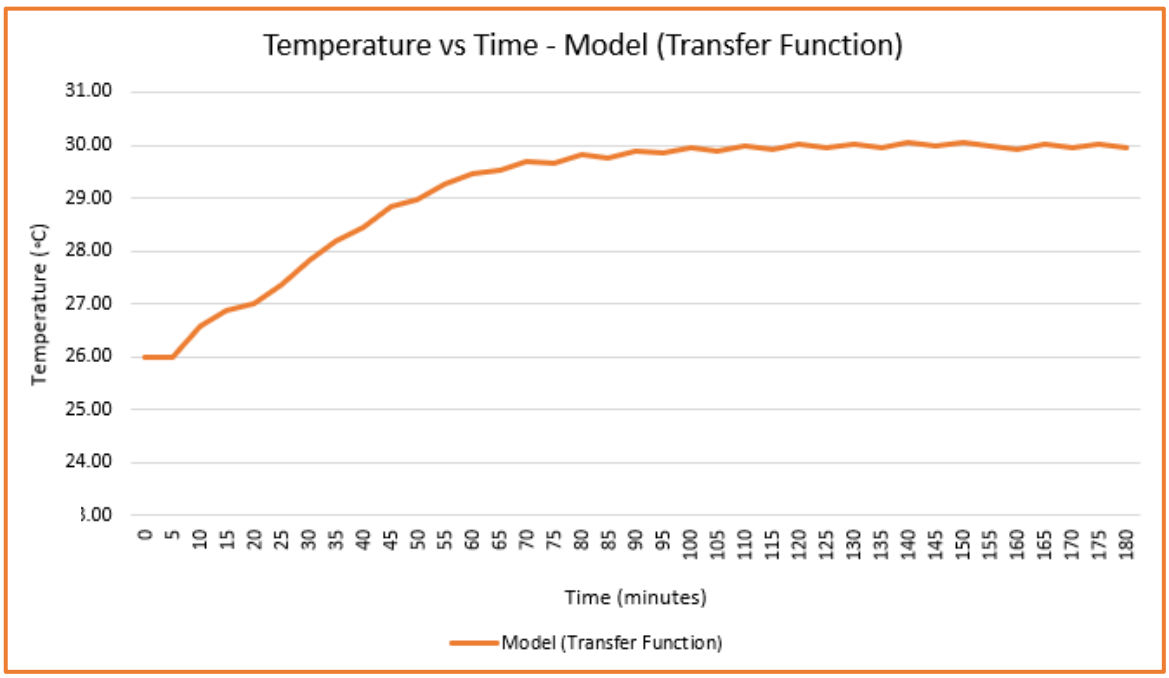


Figure 87: *Simulink Transfer Function Model Experimentation Result*

Figure 89, above, shows the Simulink Transfer Function experiment that was carried out. Using the same setpoint as the previous iterations, the key data obtained was as below.

Data	Value	
Initial Temperature (℃)	26.00	
Peak Temperature (℃)	30.06	
Rise Time (Minutes)	62.50	
Settling Time (Minutes)	140	
Overshoot (%)	0.20	
Steady State Error (℃)	0	
Initial Delay (Seconds | Minutes)	400	6.67

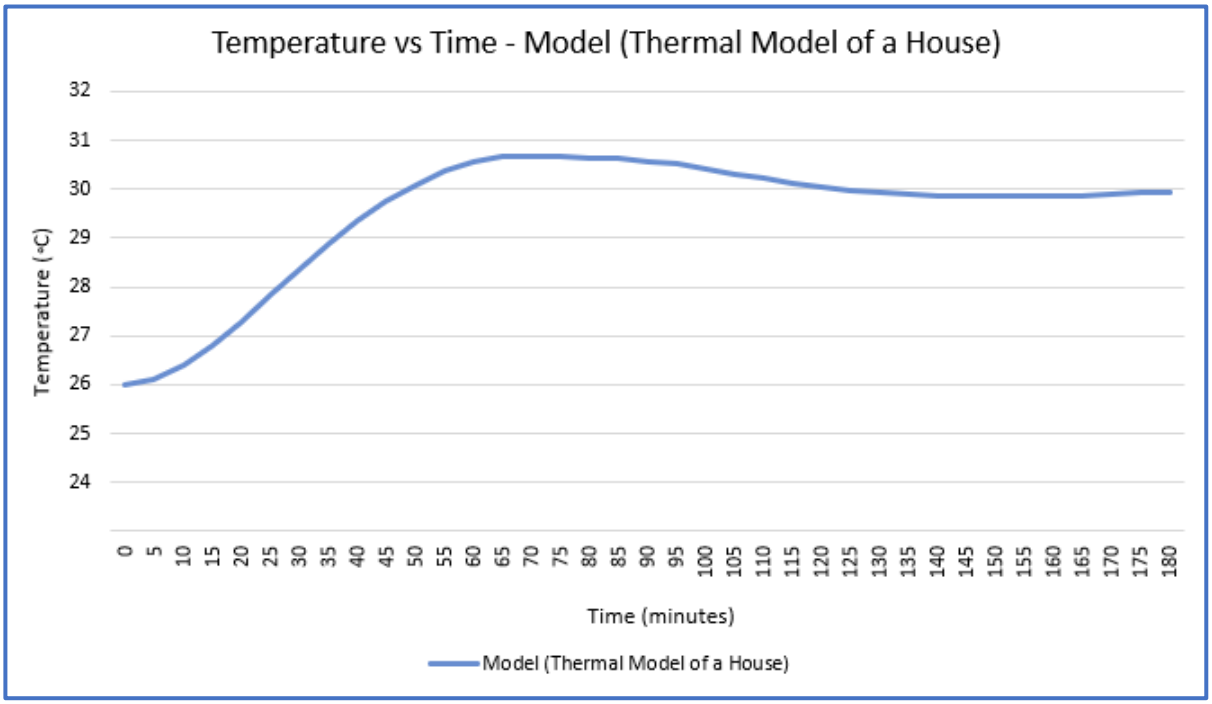


***Figure 88**: Simulink House Heating System Model Experimentation Result*

Figure 90, above, shows the Simulink House Heating System experiment that was carried out. Using the same setpoint as the previous iterations, the key data obtained was as below.

Data	Value	
Initial Temperature (℃)	26.00	
Peak Temperature (℃)	30.69	
Rise Time (Minutes)	35	
Settling Time (Minutes)	160	
Overshoot (%)	2.30	
Steady State Error (℃)	0.06	
Initial Delay (Seconds | Minutes)	400	6.67

Table 8: Summary of all curves' characteristics

Iteration	Initial Temperature (℃)	Peak Temperature (℃)	Rise Time (mins)	Settling Time (mins)	Overshoot (%)	Steady State Error (℃)	Closed - Loop Stability
1	26.06	30.24	47.5	115	0.80	0.01	Stable
2	26.10	30.09	60	103.5	0.30	0.01	
3	25.50	30.03	82.5	132.5	0.10	0	
4	26.03	30.04	65	125	0.13	0.01	
5	26.09	30.07	55	105	0.23	0	
6	26.00	30.03	57	117.5	0.10	0.01	
7	25.90	30.07	75	122.5	0.23	0.01	
8	25.98	30.11	65	115	0.37	0	
9	25.93	30.05	57	120	0.17	0	
10	25.92	30.09	70	117.5	0.30	0	
11 (incl. Flask Web App)	25.95	30.03	69	120	0.08	0.02	
Simulink Transfer Function Model	26.00	30.06	62.5	140	0.20	0	
Simulink House Heating System Model	26.00	30.69	35	160	2.30	0.06	
Minimum Value	25.5	30.03	35	103.5	0.08	0	**Overall Statistics**
Maximum Value	26.10	30.69	82.50	160	2.30	0.06	
Mean Value	25.96	30.12	61.58	122.58	0.41	0.01	
Lower Quartile	25.93	30.04	57	115	0.13	0	
Upper Quartile	26.03	30.09	69	125	0.30	0.01	

Error! Reference source not found., above, presents a summary of all curve outputs and key data including the overall statistics such as mean values etc. As was predicted from the theoretical curves, observed in **Section 3.5**, all output curves had an approximate delay of around 400 seconds (6.67 minutes). In practical terms, this is the time taken for the heating element to turn on and heat up itself. From this point, the heating element begins heating the ambient air in the enclosure. For the analysis of all parameters, the practical experimentation is compared to two Simulink models; a) the Transfer Function model i.e., the

theoretically expected results, and b) the House Heating model i.e., the results expected from a system with very similar specifications to our system.

The room temperature, in which the enclosure was placed, was 26 ℃. Hence, the two Simulink models were set with an initial temperature to match the room temperature. All the practical experimentation values of initial temperature were around the desired point, with the mean initial temperature being 25.96 ℃. The mean value was slightly less than the desired point and the minimum initial temperature was 25.50 ℃. The days on which these experiments were conducted, may have been slightly cooler, which explains the small shift below the desired initial temperature value 26 ℃. However, this can be considered a negligible difference when studying the experiment as a whole.

The setpoint, in all iterations, was set to 30 ℃. The theoretical Transfer Function Simulink model predicted a peak temperature of 30.06 ℃, while the House Heating Simulink model predicted 30.69 ℃. This can be considered a significant difference. During the setup of the House Heating Simulink model, it was observed that as the model was for a house, which required window and insulation specifications that were not applicable to our practical design. In simple terms, the model was set up to capture heat within its enclosure and hence, even though the heater would theoretically stop heating the ambient air, the temperature would still rise to 30.69 ℃. We can consider this result to be an outlier to all other results and it has hence skewed our mean peak temperature value. Our other Simulink model, the Transfer Function model, was created using the same transfer function that was used to create the PID controller of the physical system. Therefore, it can be considered quite accurate and close to our practical results. The peak temperature prediction was 30.06 ℃ and this was within the lower quartile peak temperature value of 30.04 ℃, and upper quartile peak temperature value of 30.09 ℃, of our practical results. Therefore, using peak temperature as a parameter, we can conclude that the PID controller design was successful, and the overall practical experiment behaved as expected.

Like the Peak Temperature analysis, the House Heating Simulink Model was once again an outlier (35 minutes rise time), significantly lower than the mean rise time of 61.58 minutes. Once again, it can be assumed to be due to the differences in window and insulation specifications (as discussed previously) meaning that the Simulink Model predicted that the enclosure would capture heat more efficiently than was physically feasible. However, the predicted rise time from our other Simulink Model using the Transfer Function was very similar – 62.5 minutes (Transfer Function Model) to 61.58 minutes (Mean Rise Time). Therefore, we could conclude again that our physical experiment proved successful and presented outputs that were very close to the theoretically expected results.

While the experiment was successful, another result of note was iteration three, in which the rise time was 82.5 minutes – considerably longer than the mean rise time. There was some fluctuation in the rise time across all the iterations. This could be due to variations in the heating element's behaviour and fluctuations in heat flow across the enclosure, causing delays or quickening the heating process. For example,

as the enclosure was narrow and long, there may have been instances in several iterations where the heat may not have flowed across the full enclosure evenly, thus causing a fluctuation to the rise time. There may be methods to validate and improve this, which has been discussed in **Section 6.0** (Future Work).

Like the previous results, the Simulink House Heating Model was an outlier in terms of settling time as well (160 minutes). In simple terms, this may be because the heat is "trapped" inside the model's enclosure for a longer period, so it takes longer for the ambient temperature in the enclosure to cool down, resulting in in a longer settling time. This theory may be further proven by the 2.30 % overshoot that was recorded (also an outlier from all results).

Therefore, in our entire experiment, the Simulink House Heating System, barring a few experiment set up elements as discussed before, can be considered as a useful means of comparison to our system (only to verify our system is functioning). However, when comparing results with our practical experiment results, the House Heating Simulink Model was observed to be an outlier in all recorded variables and thus couldn't be used to determine how perfectly and precisely our system was working.

In contrast, the Transfer Function Simulink Model was based on the same transfer function as our practical experiment, so the comparison was more accurate and could be used precisely to determine how well our practical system's results were. In terms of settling time, the Transfer Function Simulink Model took 140 minutes, while the practical experiment's mean settling time was 122.58 minutes. While the difference between both can be considered large (as the time unit is minutes), the Simulink Model predicted a longer settling time.

For any similar experiment, one would want the system to reach and settle around the desired set point as quickly as possible, which would mean that the practical results were better. Although the transfer function is the same for both experiment types, it is important to note that the practical system has more factors that determine whether the theoretical result would be obtained. For example, the lower settling time means that the enclosure ambient temperature cooled around the desired set temperature faster. The reason for this; however, can be due to heat loss. Although the enclosure was fully sealed (as explained in the set up sections above) there is a chance of small gaps for air to flow out, due to human error.

When accounting for such possibilities, the difference between both settling times can be considered small. Therefore, based on settling time, we can conclude that the practical system was functioning as expected and the experiment was successful.

Likewise, the overshoot in the Transfer Function Simulink Model (0.20 %) was similar to the mean overshoot value (0.41 %). However, the mean overshoot value is skewed due to the outlier values obtained from the House Heating Simulink Model. When the mean overshoot value was recalculated without the outlier result, the mean value was 0.25 %. This value is much closer to the Transfer Function Simulink Model and confirms that our practical implementation was working correctly, and the experiment was successful.

The steady state error in the practical experimentation was also very close to the theoretical error (difference of only 0.01℃) and we can conclude that the difference was negligible.

Overall, the House Heating Simulink Model can be considered an outlier. However, despite the differences in the results, the overall pattern of the data is similar, which suggests that the practical experiment model is capturing the essential dynamics of the system. Likewise, the similarity of the practical results to the Simulink Transfer Function Model results gives us confidence that our practical experiment behaved as expected and also supports the validity of our practical experimentation model. Therefore, this methodology and experiment can be considered successful. The successful outcome of the experiment demonstrates that the methodology and setup were effective, and that the experiment produced reliable results.

The implications of this experiment & results, their relation to the objective of this thesis, and the relevant connections to previous studies and literature, has been discussed in **Section 5.2** (Discussion). Although a successful experiment, the Transfer Function Simulink Model was a useful tool for comparing the results of the practical experiment and helped to identify the areas where the practical system could be improved, these are further discussed in **Section 6.0** (Future Work).

5.0 Conclusions & Discussion

5.1 Conclusion

We introduced modern farming trends, such as the transition towards organic farming, and also discussed the current issues faced, such as climate change and loss of land. CEA practices, in particular vertical farming, solves or reduces the issues by enabling soilless farming and vertical stacking of crops. However, due to limited implementation and technology, there are various gaps, such as lack of knowledge, high implementation cost, cyber security risks, need for human intervention, and lack of temperature control implementation.

To understand how these gaps can be reduced, we designed and implemented a small-scale Nutrient Film Technique (Vertical Farming method) based solution. This included various components and frameworks, such as a temperature sensor, heating element, cooling element, LEDs, Raspberry Pi microcontroller, and ROS etc. Using a user input on a website, we used the set point and a specifically designed PID controller to automatically control the temperature using the heating and cooling elements. The results, i.e., the temperature curve outputs, provided information to validate our solution and as discussed, our practical experiment results were very similar to the theoretically expected results. The minor differences were believed to be due to human error, variation in elements' behaviours, and fluctuations in heat flow, etc. Overall, the solution provided an expected pattern of results and consequently proved to be a possible solution to the gaps observed.

In summary, the contributions of this thesis include:

- A robust literature review into the current systems used for various farming methods.
- Developing a prototype CEA system, which uses technologically advanced methods, that are currently less employed in the industry.
- Including temperature monitoring and control, as well as automation i.e., lack of human intervention required; these can be considered as the novel aspects of this thesis.
- Highlighting potential improvements that can be incorporated in the future to enable upscaling of the system.

Section 5.2, below, discusses the specific details of our experiment and their relation to the barriers identified. In the future, using improvements discussed in **Section 6.0**, the solution can be enhanced to narrow the difference between practical and theoretical results. As a result, the solution can then be upscaled to enable it as an industrial application.

5.2 Discussion

The scope of this project was to find possible solutions or methods to bridge the gaps and barriers researched and summarised in **Section 2.7**. As discussed, five main barriers or gaps were identified and the possible methods in bridging them, that were attempted in this project, are detailed below.

Cyber Security: The first gap identified was cyber security and the main reasons was the extensive use of IoT in the industry. Our solution, attempted to implement the system in a different way i.e., using two Raspberry Pi microcontrollers. This way there was limited cyber security risks and our solution provides a proven method in implementing a CEA enclosure with reduced security risks. However, when looking at future implementations and upscaling the system, a robust solution needs to be identified as our solution is more suitable for small-scale. This is further discussed in Section 6.0 (Future Work).

Lack of Knowledge: Not having sufficient or adequate knowledge was also a barrier identified for farmers to implement a CEA system. The aim of our project was to make the solution simple and therefore propose a small but robust method that can be used as an example for implementing by users with limited knowledge in this field. Consequently, our solution was designed and developed using components that were easily accessible, and their information or data was easily accessible on the internet. This ensures that any user can find component information and use it to solve problems. This also bridges the 'limited knowledge' gap. However, our system was not simple in all aspects. I.e., the connection of two microcontrollers and the communication between them was not minimal. Hence, when creating such a solution, documenting the process and reasoning would also aid in bridging the gap. It may be beneficial for new users to implement a smaller CEA solution like ours before upscaling towards the industry-like solutions.

High Implementation Cost: Similar to the barrier discussed above, high implementation cost was also a gap found. Our solution was designed with small-scale and low-cost components; hence, for small-scale implementations it was a solution with low implementation cost. With a focus on upscaling this project, if similar components are used, the implementation cost may be lower than the current industry solutions. However, this is subject to future work carried out as this may increase the implementation cost. Overall, our solution positively addresses the gap and also provides an interim solution towards reducing the gap.

Human Intervention: Various studies detailed that their method required human intervention for any action to be carried out once the monitoring data was received. I.e., the user had to analyse the data and consequently decide what components were to be actuated. Our system avoided this and provided an autonomous solution once the set point had been entered. The user could monitor the temperature change on the website in case of any malfunction; however, overall, the PID controller and program maintained the temperature by automatically switching the heating and cooling components on or off. This was an attempt to address the gap and provide a proven solution towards an autonomous CEA.

Temperature Control has not been explored in great depth with many CEA implementations focussing on other variables such as lighting, humidity etc. Hence, our project aimed to explore this, and our solution was targeted towards this gap to understand how the gap can be bridged. Our solution proved successful and hence addressed the gap discussed in **Section 2.6**.

6.0 Future Work

The experimentation phase of the project was successful; however, there are various improvements that can be made to the project to gain a better understanding of how the gaps found during the research phase can be filled or made smaller. The possible improvements or future work is presented below. The structure follows a similar order to this thesis' sections.

Research: While various studies and current implementations were researched, a few more findings would be beneficial in developing a more robust solution. Below are three topics for future research.

- Researching current vertical farming solutions in vastly different and various conditions i.e., current implementation in cooler or hotter conditions as it will affect the ambient temperature around the enclosure significantly and how this is currently combated. While this was not required for the scope of this project, this additional research will ensure a flexible solution is designed to ensure the system would work in various conditions.
- It was noticed that only some small-scaled CEA implementations use heating elements; while many small-scaled CEA implementations use an active cooling and passive heating approach i.e., fans used for cooling but no heating element (the small enclosure and plant characteristics are expected to heat up the enclosure – takes longer but still effective). As our approach included active heating, i.e., using a heating element, in the future more research into why small-scaled CEA approaches are not implementing heating elements would be beneficial.
- While this project's scope was small-scaled and focussed on technology, it may be beneficial to research the current CEA implementations that converted from small to large scale i.e., upscaled. Consequently, problems that were solved and any difficulties in such an operation

would also be beneficial so that those problems can be rectified in the small-scaled design in future work of this project.

Practical Experiment Manufacturing and Design: Various results from our practical experimentation, detailed in **Section 4.0** (Results), can be refined and made more precise with changes to our enclosure and its setup and design. Below are several elements that can be improved in the future.

- Reduce enclosure size to include only one plant pot (instead of the seven pots in the current enclosure). This will ensure that the enclosure size will have a reduced effect on the variables measured.
- LEDs were used as the lighting source in the current system and was used to replicate daytime and night-time. As found during the research phase of this project, various current industry implementations use LED grow lights i.e., LEDs that enhance or aid plant growth (using UV and IR light). This will improve the lighting status currently designed.

- Explore different heating & cooling element options to enable the control of rate of heating or cooling. For example, in the current set up, the element was either on or off depending on the temperature. However, this can be enhanced by controlling the rate of heating or cooling (e.g., for cooling - controlling fan speed) rather than switching the element completely off. This may enhance the PID controller and consequently enhance the temperature control output.
- Re-design the material used to add insulation material or an opaque reflective film (e.g., Mylar [86]) inside the enclosure. This will reduce heat loss inside the enclosure and therefore will enhance the rise time (i.e., time required to heat the enclosure to the desired set point). A reflective film, like above, will also reflect light which will also aid in heating the enclosure.
- As the design revolves around the enclosure being opaque, an additional technological enhancement will be installing cameras. This will enable the user (accessing the website remotely) to view the live stream or feed of the cameras. This will be better implemented if the feed from the camera is on the implemented website. If the setup is upscaled, then the website can give the user an option to select from multiple cameras and can check for progress or any issues live on screen.
- A smaller feature added may be handles to the enclosure; hence making it portable and allowing the user to transport the enclosure anywhere.
- The current design was a prototype with the main objective to test temperature control; however, when repeating with more enhancements, the wiring set up and accessibility designs can be improved to reduce manufacturing time and costs. Also, the enclosure can be designed to be flexible in expanding. I.e., the enclosure could be built to have the flexibility of attaching

to other similar enclosures to build a larger enclosure – this will make the system modular for any type of plant and will also ensure that upscaling is easier to achieve.

Website & System Integration: The integration of the elements can be refined and made more user friendly with some improvements to the setup and architecture. Below are a few elements that can be improved in the future.

- Explore potential improvements to the overall setup and architecture by using only one Raspberry Pi microcontroller. This would be used for the component control e.g., heating element etc. The other end i.e., the user end and website as well as data storage, could be online and cloud based – like some current industry studies discussed in **Section 2.5** (System Integration). This may reduce set up complexity as there would be fewer physical components. Contrastingly, there may be added complexity in the software set up; however, as the website is also online it may be advantageous having all the set up together. Also, designing this will also provide more knowledge and solutions in terms of cyber security which was discussed as a barrier or gap in **Section 2.3**.
- Improve the website to automatically refresh. Currently, the website needs to be manually refreshed for an updated current temperature value to be displayed on the website. In the future, the website can be improved to automatically refresh so that the current temperature value is up-to-date, and the user can monitor without intervention.
- Currently, the software set up uses threads to run the publisher and subscriber node simultaneously and continuously, as discussed in **Section 3.6** (System Integration). More advanced and concise methods to achieve this should be explored in the future as the threading method increases complexity of the program and is also memory intensive.

Overall Experimentation: While the tests and experiments carried out were sufficient and adequate for our scope, in the future, it may be beneficial to test the system with several other temperature set points to ensure that the system is working as expected in all conditions. Similarly, it may be beneficial to test the system with water flowing through the plant tray as the flow of water could have some effect on the ambient temperature within the enclosure.

The items discussed above for future work will ensure that the current system is improved, enhanced and further robust. This will mean that the system will be more technologically advanced and be able to near or match the current industry small-scale implementations and studies.

7.0 References

[1] G. M. Zinati, “Transition from conventional to organic farming systems | Challenges, Recommendations, and Guidelines for pest management,” vol. 12, p. 5, 2002.

[2] J. P. Reganold and J. M. Wachter, “Organic agriculture in the twenty-first century,” *Nature Plants,* vol. 2, 2016.

[3] J. Paull, “The Lost History of Organic Farming in Australia,” *Journal of Organic Systems,* vol. 3, no. 2, p. 18, 2008.

[4] Admin, “Organic vs Traditional Farming,” 16 August 2010. [Online]. Available: https://nofari.org/organic-farming/organic-vs-traditional-farming/#.X4f0zu1S-Uk. [Accessed October 2020].

[5] H. Willer, J. Lernoud and L. Kemper, “The World of Organic Agriculture 2018: Summary,” p. 10, 2018.

[6] H. Willer and J. Lernoud, “The World of Organic Agriculture Statistics and Emerging Trends 2019,” p. 351, 2019.

[7] Rainforest-alliance.org, “What is the Relationship between deforestation and climate change?,” 12 August 2018. [Online]. Available: https://www.rainforest-alliance.org/articles/relationship-between-deforestation-climate-change. [Accessed October 2020].

[8] P. Borrelli, D. A. Robinson, L. R. Fleischer, E. Lugato, C. Ballabio, C. Alewell, K. Meusburger, S. Modugno, B. Schutt, V. Ferro, V. Bagarrello, K. Van Oost, L. Montanarella and P. Panagos, "An assessment of the global impact of 21st century land use change on soil erosion," *Nature Communications,* 2017.

[9] M. Cunningham, "Problems in agriculture: Loss of land and decreased varieties," 2013. [Online]. Available: https://study.com/academy/lesson/problems-in-agriculture-loss-of-land-decreased-varieties-smaller-crop-yields.html. [Accessed 2020].

[10] K. C. Seto, M. Fragkias, B. Guneralo and M. K. Reilly, "A meta-analysis of global urban land expansion," 2011.

[11] R. Checa-Garcia, K. P. Shine and M. I. Hegglin, "The contribution of greenhouse gases to the recent slowdown global-mean temperature trends," p. 8, 2016.

[12] S. Russell, "Everything you need to know about agricultural emissions," May 2014. [Online]. Available: https://www.wiri.org/blog/2014/05/everything-you-need-to-know-about-agricultural-emissions. [Accessed August 2020].

[13] H. Ritchie and M. Roses, "Our World In Data - Agriculture," 2018. [Online]. Available: https://ourworldindata.org/emissions-by-sector. [Accessed August 2020].

[14] H. Goldstein, "The Green Promise of Vertical Farms," 02 June 2018. [Online]. Available: https://spectrum.ieee.org/energy/environment/the-green-promise-of-vertical-farms. [Accessed October 2020].

[15] R. R. Shamshiri, F. Kalantari, K. C. Ting, K. R. Thorp, I. A. Hameed, C. Weltzien, D. Ahmad and Z. M. Shad, "Advances in greenhouse automation and controlled environment agriculture: A transition to plant factories and urban agriculture," vol. 11, no. 1, p. 22, 2018.

[16] S. Goddek, B. Delaide, U. Mankasingh, K. v. Ragnarsdottir, H. Jijakli and R. Thorarinsdottir, "Challenges of sustainable and commercial aquaponics," 2015.

[17] A. AlShrouf, "Hydroponics, Aeroponics, Aquaponics as compared with conventional farming," vol. 27, no. 1, p. 9, 2017.

[18] M. F. Saaid, N. A. Yahya, M. Z. Noor and M. S. A. M. Ali, "A development of an automatic microcontroller system for deep water culture (DWC)," *Signal Processing and its Application,* p. 5, 2013.

[19] K. A. El-Kazzaz and A. El-Kazzaz, "Soilless agriculture a new and advanced method for agriculture development: an introduction," *Research Article,* vol. 3, no. 2, p. 10, 2017.

[20] K. Benke and B. Tomkins, "Future food-production systems: vertical farming and controlled-environment-agriculture," vol. 13, no. 1, p. 15, 2017.

[21] I. A. Lakhiar, J. Gao, T. W. Syed, F. A. Chandio and N. A. Buttar, "Modern plant cultivation technologies in agriculture under controlled environment: a review on Aeroponics," vol. 3, no. 1, 2018.

[22] J. I. Montero, M. Teitel, J. C. Lopez and M. Kacira, “Good Agriculture practices for greenhouse vegetable crops: Principles for mediterranean climate areas,” p. 640, 2013.

[23] U. Nations, “United Nations: Population,” 2019. [Online]. Available: https://www.un.org/en/sections/issues-depth/population/. [Accessed October 2020].

[24] J. Badgery-Parker, “The Greenhouse,” p. 2, 1999.

[25] Garden-How-To, “Greenhouses: Pros & Cons,” 2014. [Online]. Available: www.gardenweasel.com/greenhouses-pros-cons/. [Accessed September 2020].

[26] A. Garg and R. Balodi, “Recent trends in agriculture: vertical farming and organic farming,” vol. 1, no. 4, 2014.

[27] C. Kremen and A. Miles, “Ecosystem services in biologically diversified versus conventional farming systems: Benefits, externalities and trade-offs,” 2012.

[28] D. Touliatos, I. C. Dodd and M. McAinsh, “Vertical farming increases lettuce yield per unit area compared to conventional horizontal hydroponics,” *Food and Energy Security,* vol. 5, no. 3, p. 8, 2016.

[29] W. Goodman and J. Minner, “A case study of controlled environment agriculture in New York City,” *Will the Urban Agriculture revolution be vertical and soilless?,* 2018.

[30] A. Abdullah, S. Al Enazi and I. Damaj, “AgriSys: A smart and ubiquitous controlled-environment agriculture system,” p. 6, 2016.

[31] AeroFarms, “Our Indoor Vertical Farming Technology,” [Online]. Available: https://aerofarms.com/technology. [Accessed September 2020].

[32] G. Greens, “Our Farms: Cultivating Cities, and growing Veggies too,” [Online]. Available: https://www.gothamgreens.com/our-farms/. [Accessed September 2020].

[33] Plenty, “About Vertical Farming,” [Online]. Available: https://www.plenty.ag/vertical-farming/. [Accessed September 2020].

[34] W. Marchant and S. Tosunoglu, “Robotic Implementation to automate a vertical farm system,” p. 6, 2017.

[35] S. Bhowmick, B. Biswas, M. Biswas, A. Dey, S. Roy and S. K. Sarkar, “Application of IoT-Enabled Smart Agriculture in Vertical Farming,” in *Advances in Communication, Devices, and Networking*, 2019, p. 7.

[36] Y. S. Chin and L. Audah, “Vertical Farming monitoring system using the Internet of Things (IoT),” 2017.

[37] C. Kulatunga, L. Shalloo, W. Donnelly, E. Robson and S. Ivanov, “Opportunistic Wireless networking for smart farming,” p. 8, 2017.

[38] Y. D. Chuah, “Implementations of smart monitoring system in vertical farming,” p. 7, 2019.

[39] U. Sharma, M. Barupal, N. S. Shekhawat and V. Kataria, “Aeroponics for propogation of horticulture plants: an approach for vertical farming,” vol. 2, no. 6, 2018.

[40] C. Gnauer, H. Pichler, M. Tauber, C. Schmittner, K. Christl, J. Knapitsch and M. Parapatits, “Towards a secure and self adapting smart indoor farming framework,” 2019.

[41] L. Barreto and A. Amaral, “Smart Farming: Cyber Security Challenge,” 2018.

[42] M. Gupta, M. Abdelsalam, S. Khorsandroo and S. Mittal, “Security & Privacy in Smart Farming: Challenges & Opportunities,” vol. 8, p. 21, 2020.

[43] S. Wolfert, M.-J. Bogaardt, L. Ge and C. Verdouw, “Big Data in Smart Farming - A Review,” *Agricultural Systems,* p. 12, 2017.

[44] R. H. Kothiya, K. L. Patel and P. H. S. Jayswal, “Smart farming using Internet of Things,” vol. 13, no. 12, p. 5, 2018.

[45] S. Chandler, “What are the challenges of building a smart agriculture system,” 17 January 2019. [Online]. Available: https://internetofthingsagenda.techtarget.com/answer/What-are-the-challenges-of-building-a-smart-agriculture-system. [Accessed September 2020].

[46] H. Fakhruddin, “Precision Agriculture: Top 15 Challenges & Issues,” 2017. [Online]. Available: https://teks.co.in/site/blog/precision-agriculture-top-15-challenges-and-issues/. [Accessed September 2020].

[47] H. C. Punjabi, S. Agarwal, V. Khithani, V. Muddaliar and M. Vasmatkar, “Smart Farming using IoT,” vol. 8, no. 1, p. 9, 2017.

[48] A. L. Virk, M. A. Noor, S. Fiaz, S. Hussain, H. A. Hussain, M. Rehman, M. Ahsan and W. Ma, “Concepts and developments,” *Smart Village Technology,* p. 12, 2020.

[49] F. W. Went, “Plant Growth Under Controlled Conditions. II. Thermoperiodicity In Growth And Fruiting Of The Tomato,” *Factors Affecting Plant Growth,* vol. 3, p. 16, 1944.

[50] T. E. Shomefun, C. O. Awoscope and O. D. Ebenezer, “Microcontroller-Based Vertical Farming Automation System,” vol. 8, no. 4, p. 8, 2018.

[51] EL-PRO-CUS, “BH1750-Specification and Application,” 2020. [Online]. Available: http://www.elprocus.com/bh1750-specifications-and-applications/. [Accessed December 2020].

[52] I. Laktionov, I. Getman, V. Lebediev, O. Vovna and A. Maryna, “Results of experimental research on computerised intellectual monitoring means of effective greenhouse illumination,” vol. 12, no. 1, 2019.

[53] F. Ruscio, J. Thomas, S. Fichera, P. Paoletti and P. Myers, “Low-Cost Monitoring System for hydroponic urban vertical farms,” vol. 13, no. 10, 2019.

[54] Anusha, “DHT11 Humidity Sensor on Arduino,” 21 June 2017. [Online]. Available: http://www.electronicshub.org/dht11-humidity-sensor-arduino/. [Accessed 2020].

[55] R. Thinakaran and S. Nagalingham, “Smart Vertical Farming using IoT,” vol. 49, 2020.

[56] C. Petrich, I. Saether, N. P. Dang, O. Kleven and M. O'Sadrick, “A note on Remote Temperature Measurement with DS18B20 Digital Sensors,” p. 3, 2020.

[57] adafruit, “Waterproof 1-Wire DS18B20 compatible digital temperature sensor,” [Online]. Available: https://www.adafruit.com/product/381. [Accessed December 2020].

[58] G. Rao, A. Kumar, R. N. Singh and T. P. Singh, “Water Level Monitoring System with Vertical Farming using IoT,” Pune, India, 2020.

[59] F. Hasanat, G. Nofal and S. Awad, “IoT based Customizable Vertical Farming Solution for Palestinian Plants,” Palestine, 2022-23.

[60] D. W. Wagner, “A novel system architecture for automated field-based tent systems for controlled-environment agriculture and experimentation,” Manhattan, 2017.

[61] H. Ibrahim, N. A. El-Mawla and G. Team, “Proposed Model for Sustainable and Scalable Vertical Farm,” Mansoura, 2022.

[62] R. C. Gustilo, D. Guillermo and F. Dim, “Automated IoT Enabled Vertical Farming: Planting Atypical Crops in an Urban Environment,” Manila, 2023.

[63] K. Khan Liwal, M. Vohra, H. Sheikh, O. Al-Khatib, N. A. Aziz and C. Copiaco, “Implementation of a sustainable and scalable vertical micro-farm,” Dubai, 2020.

[64] InterviewBit, “Arduino vs Raspberry Pi: What's The Difference?,” 11 November 2021. [Online]. Available: https://www.interviewbit.com/blog/arduino-vs-raspberry-pi/. [Accessed 2023].

[65] P. Hero, “What are the advantages and disadvantages of ESP32 compared to ESP8266,” 30 September 2022. [Online]. Available: https://www.pcb-hero.com/blogs/lisas-column/what-are-the-advantages-and-disadvantages-of-esp32-compared-to-esp8266. [Accessed 2023].

[66] M. Martin, T. Weidner and C. Gullstrom, “Estimating the Potential of Building Integration and Regional Synergies to Improve the Environmental Performance of Urban Vertical Farming,” vol. 6, 2022.

[67] D. D. Avgoustaki and G. Xydis, “How energy innovation in indoor vertical farming can improve food security, sustainability and food safety?,” 2020.

[68] J. C. S. Ltd, “Vertical Farming Solutions,” [Online]. Available: https://www.jdcooling.com/cooling/vertical-farming/. [Accessed 2023].

[69] T. Kozai, K. Fujiwara and E. S. Runkle, “LED Lighting for Urban Agriculture,” Sprinker Nature, Tokyo, 2016.

[70] E. C. Ferrer, J. Rye, G. Brander, T. Savas, D. Chambers, H. England and C. Harper, “Personal Food Computer: A new device for controller-environment agriculture,” Cambridge, MA, 2017.

[71] F. Ruscio, P. Paoletti, J. Thomas, P. Myers and S. Fichera, “Low-Cost Monitoring System for Hydroponic Urban Vertical Farms,” 2019.

[72] M. Ryu, J. Yun, T. Miao, N.-Y. Ahn, S.-C. Choi and J. Kim, “Design and Implementation of a Connected Farm for Smart Farming System,” Seongnam, 2015.

[73] Ubuntu, “How to install Ubuntu Desktop on Raspberry Pi 4,” [Online]. Available: https://ubuntu.com/tutorials/how-to-install-ubuntu-desktop-on-raspberry-pi-4#2-prepare-the-sd-card. [Accessed 2023].

[74] “Ubuntu (Debian),” [Online]. Available: http://docs.ros.org/en/foxy/Installation/Ubuntu-Install-Debians.html. [Accessed 2022].

[75] “Installation | Using Debian Packages,” [Online]. Available: https://colcon.readthedocs.io/en/released/user/installation.html. [Accessed 2022].

[76] D. F. K. Noble, “How to run ROS 2 in Windows via WSL 2,” Auckland, 2021.

[77] D. F. Noble, “Writing a simple publisher and subscriber,” Auckland.

[78] S. Campbell, “Raspberry Pi DS18B20 Temperature Sensor Tutorial,” [Online]. Available: https://www.circuitbasics.com/raspberry-pi-ds18b20-temperature-sensor-tutorial/. [Accessed 2022].

[79] “What is Flask Python,” 2021. [Online]. Available: https://pythonbasics.org/what-is-flask-python/. [Accessed 2022].

[80] D. F. K. Noble, “Getting Started with Flask,” Auckland.

[81] K. Ogata, Modern Control Engineering, Fifth Edition, Katsuhiko Ogata.

[82] C. T. f. M. &. SIMULINK, “Introduction: PID Controller Design,” [Online]. Available: https://ctms.engin.umich.edu/CTMS/index.php?example=Introduction&section=ControlPID. [Accessed 2023].

[83] P. Salmony, “Phil's Lab,” [Online]. Available: https://www.phils-lab.net/. [Accessed 2023].

[84] P. Salmony, “PID Controller Implementation in Software,” 23 May 2020. [Online]. Available: https://www.youtube.com/watch?v=zOByx3Izf5U. [Accessed 2023].

[85] MathWorks, “House Heating System,” [Online]. Available: https://au.mathworks.com/help/simscape/ug/house-heating-system.html. [Accessed 2023].

[86] G. &. Brew, “Mylar Reflective Film - 7.5mtr,” [Online]. Available: https://www.growandbrew.co.nz/products/mylar-film-7-5-mtr. [Accessed 2023].

[87] S. Sharma, A. Kaushik and J. V. Das, “View of Irish Farmers on Smart Farming Technologies: An Observational Study,” p. 24, 2019.

[88] S. Bertram, “Automation: The Final Frontier of Vertical Farming,” 4 May 2019. [Online]. Available: https://www.agritecture.com/blog/2019/5/10/automation-the-final-frontier-of-vertical-farming. [Accessed September 2020].

[89] Alexnld.com, “BH1750FVI Digital Light Intensity Sensor Module For AVR Arduino 3V-5V Power,” 2020. [Online]. Available: https://alexnld.com/product/bh1750fvi-digital-light-intensity-sensor-module-for-avr-arduino-3v-5v-power/. [Accessed December 2020].

[90] E. Bazaar, “DHT11 Temperature and Humidity Sensor,” [Online]. Available: https://www.indiamart.com/proddetail/dht11-temperature-and-humidity-sensor-19237198597.html. [Accessed December 2020].

[91] “Egmont Hydroponic Complete Kit,” 23 April 2023. [Online]. Available: https://www.bunnings.co.nz/egmont-hydroponic-complete-kit_p0238817?store=9491&gclid=Cj0KCQjwi46iBhDyARIsAE3nVrZ8Y99XuaH5Mad-m-2fRCQXcD6os_XDWrI24YaFujk-ZX2brwBMtZ4aAiThEALw_wcB&gclsrc=aw.ds.

[92] C. Ltd, “OpenSSH Server,” [Online]. Available: https://ubuntu.com/server/docs/service-openssh. [Accessed 2022].

[93] Element14, “RPI3-MODBP,” 2023. [Online]. Available: https://nz.element14.com/raspberry-pi/rpi3-modbp/sbc-board-raspberry-pi-3-model/dp/2842228. [Accessed 2023].

[94] DigiKey, “Arduino A000066,” 2023. [Online]. Available: https://www.digikey.co.nz/en/products/detail/arduino/A000066/2784006. [Accessed 2023].

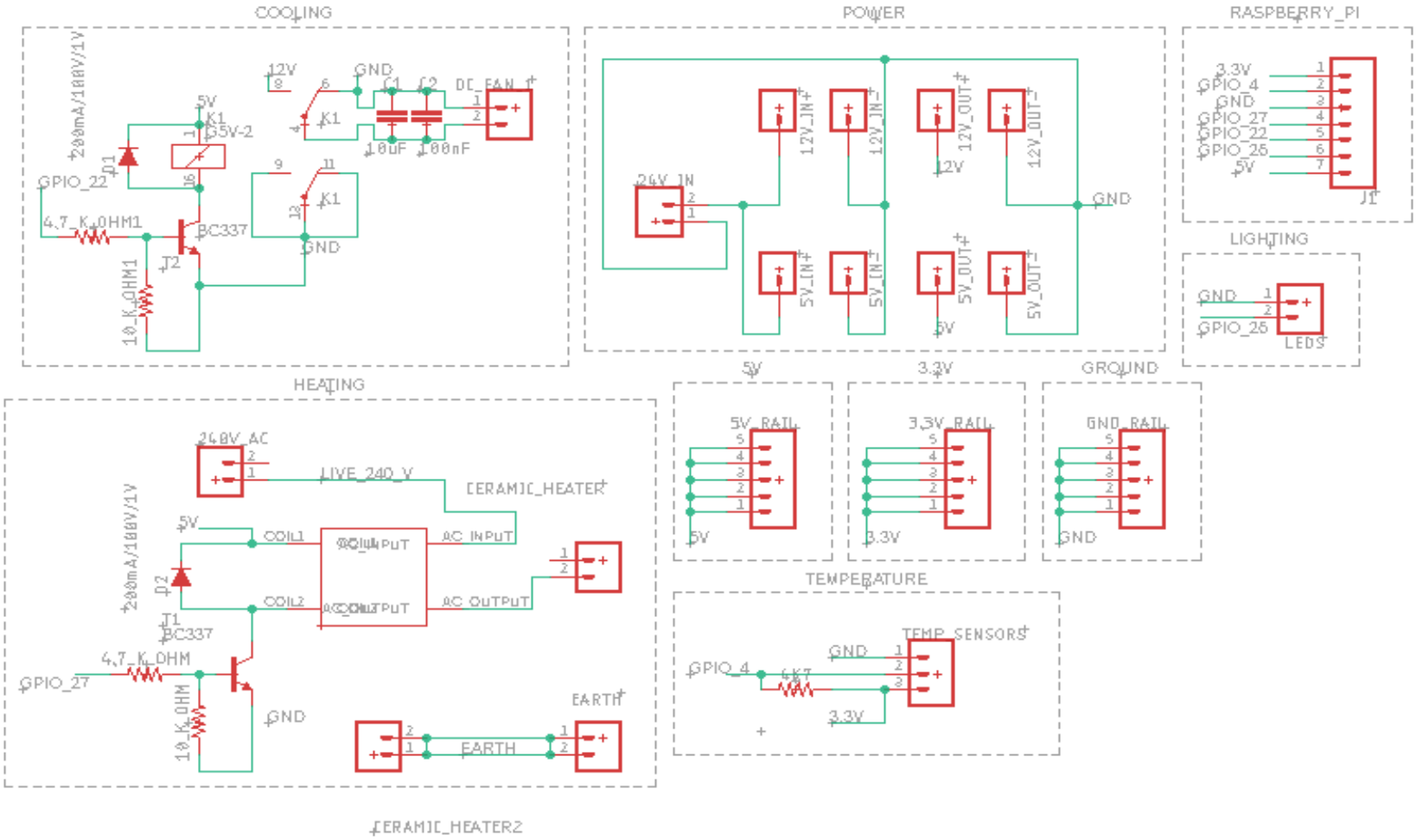
[95] A. Tutorial, “ESP8266 SPIFFs,” 2023. [Online]. Available: https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.teachmemicro.com%2Fesp8266-spiffs-web-server-nodemcu%2F&psig=AOvVaw0KyliPkbxfNj4Uisr_H4D2&ust=1687168496074000&source=images&cd=vfe&ved=0CBMQjhxqFwoTCICPx8nGzP8CFQAAAAAdAAAAABAG. [Accessed 2023].

[96] “Vindur Compact Air-Conditioning units,” [Online]. Available: https://www.weiss-technik.com/klimatechnik/en/products/detail/vindur-compact-air-conditioning-units~p13148.

[97] Amazon, “Ceramic Heating Element PTC Ceramic Air Heater 12V 150W Insulated PTC Heater Ceramic Aluminum Tube Air Heater PTC Heating Element,” 2023. [Online]. Available: https://www.amazon.com/12V-Ceramic-Heating-Element-Insulated/dp/B07XDHLLZJ. [Accessed 2023].

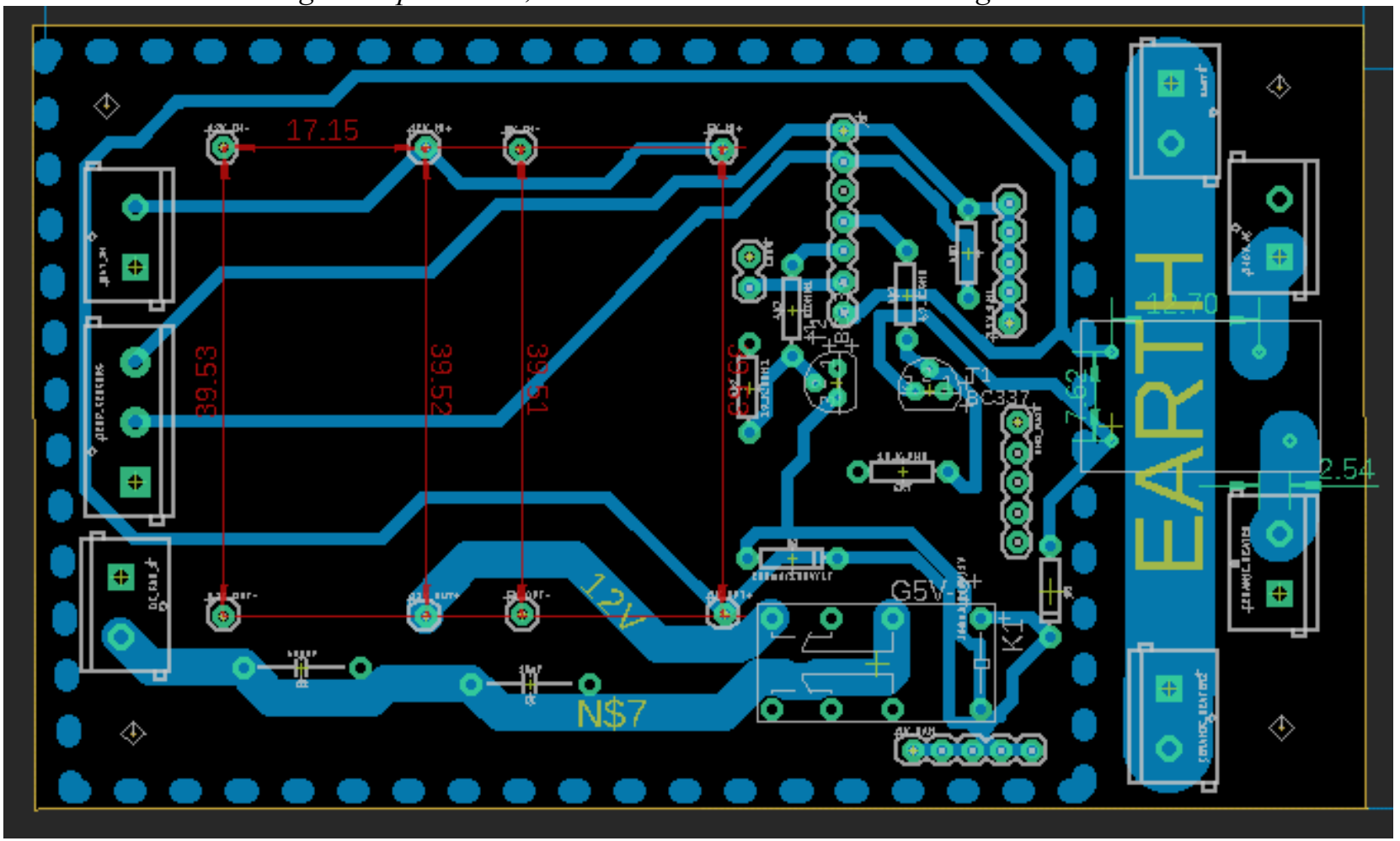
[98] A. Express, “40mm Powerful Cooling Fan 14000r For AVC 4028 12V 1A FFB0412UHN High Speed Server Fans 40x40x28mm Dual Ball Bearing 2Pin,” 2023. [Online]. Available: https://www.aliexpress.com/item/1005003749605579.html?pdp_npi=2%40dis%21NZD%21NZ%246.00%21NZ%244.13%21%21%21%21%21%402101c5c316873469178038869e6e4e%2112000027032460486%21btf&_t=pvid:320490e3-3247-4474-958f-c0d8fe3b767c&afTraceInfo=1005003749605579__pc__pc. [Accessed 2023].

8.0 Appendix



Schematic view of Printed Circuit Board in Eagle PCB Design Software

The blue dashed line on the Board layout of the PCB annotates the area on the PCB that is set as a 'ground plane' i.e., the area within the dashed area is grounded.



Board view of Printed Circuit Board in Eagle PCB Design Software