DEPARTMENT OF ELECTRICAL ENGINEERING & ELECTRONICS

Final Year Project

(ELEC340 / ELEC440)

Final Year Report

Headspace Sampling Device

Author: KITILI KISINGUH (201588006)

Project Supervisor: Dr. Barry Smith

Assessor: Ahmed Al-Irhayim

Declaration of academic integrity

I confirm that I have read and understood the University's Academic Integrity Policy.

I confirm that I have acted honestly, ethically and professionally in conduct leading to assessment for the program of study.

I confirm that I have not copied material from another source nor committed plagiarism nor fabricated, falsified or embellished data when completing the attached piece of work. I confirm that I have not copied material from another source, nor colluded with any other student in the preparation and production of this work.

SIGNATURE..... DATE 30/05/2025

Abstract 2
1. Introduction 2
1.1 Background and Motivation 2
1.2 Range of solutions 5
1.3 Project Design Overview 5
2. System Design 7
2.1 Heating Element 7
2.2 Micro Controller Unit 12
2.3 PWM Controller V1 Review 14
2.4 Temperature Sensor 14
2.5 Magnetic Stirrer 17
2.6 Headspace Distribution 18
2.7 Microcontroller Firmware 23
3. Results and Discussion 26
3.1 Circuit Analysis 26
3.2 Vial heating 30
3.3 Magnetic Stirrer 31
3.4 Headspace extraction 32
4. Conclusion 33
4.1 Further Development 33
4.2 Reflection and Learning 35
5. References 35
6. Appendices 37
LED specification 37
Optically Couple Gate Driver 37
Linear Drop Out Regulator (LDO) 38
Appendix 2: ESP32 Pinout 39
Appendix 3 PCB Design Constraints and Fabrication 40
Appendix 4: Vial Dimension 42

Appendix 5: Air Flow sensor application 43
Appendix 6: PWM Controller V1 Application Modifications 43
Appendix 7: PWM Controller V2 Application Modifications 44
Appendix 8: PWM Controller V1 Application Modifications 45
Appendix 9: Buck Converter 45
Appendix 10: 3D Housings and Attachments 46
Appendix 11: Little File System 49
Appendix 12: Gantt Chart 50
Appendix 13: Original Milestones 50
Appendix 14: Risk Assessment Form 54
Appendix 15: ESP32Webserver.INO 56
Appendix 16: index.html 60
Appendix 17: script.Js 67
Appendix 18 MATLAB Code; Response Analysis 3.3mL 71
Appendix 19 MATLAB Code; Response Analysis 6mL 74

Abstract

Automatic headspace sampling is a technique used to analyze the volatile organic compounds (VOCs) present in a liquid or solid phase sample. This method utilizes heat to separate lighter, volatile compounds from a heavier matrix. Designed as a low cost and efficient solution, this project aims to provide an automated headspace sampler for use in the mass spectrometry department. It employes a closed loop microcontroller system to precisely heat a sample vial to a specific temperature using a PID controller to maintain the temperature withing a 0.1° C error margin. Once the analyte reaches equilibrium in the gas phase, it is directed to a mass spectrometer via an arrangement piping an air pump and solenoid valve. A graphical user interface (GUI) integrates these components, providing cohesive control and monitoring of the process.

1. Introduction

1.1 Background and Motivation

Headspace sampling is a technique used to extract volatile compounds from a heavier matrix. The process involves heating the sample for a set period at a controlled temperature, promoting the transfer of volatile molecules from the sample phase into the gas phase, known as the headspace, until equilibrium is established [1]. The headspace, which is the layer of gas above the sample, illustrated in Figure 1. After equilibrium is reached, the volatile contents in the headspace are carefully extracted from the vial and introduced into analytical instruments such as a gas chromatograph (GC) or a mass spectrometer (MS) for detailed analysis.

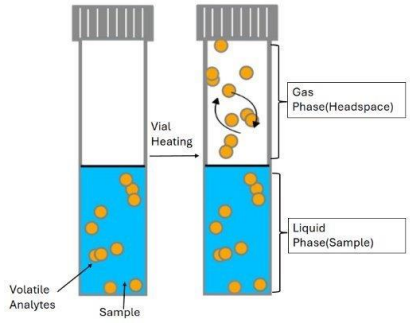


Figure1. Static Headspace Sampling overview diagram

There are two main types of headspaces sampling: static and dynamic. In static headspace sampling, the sample is sealed in a container and left to reach equilibrium, allowing volatile compounds to accumulate in the headspace until it becomes saturated with analyte vapor[1], once saturation is achieved the headspace is extracted and transferred into a gas chromatograph (GC) or mass spectrometer (MS) using clean, dry air or an inert gas. In contrast dynamic headspace sampling involves continuously purging the sample with an inert gas such as nitrogen or helium, which carries the volatile compounds into the headspace where they are captured by a trap, the trapped analytes are then desorbed typically by a heating element and introduced into the analytical instrument [1]. The effectiveness of static headspace sampling is influence by several factors, including preconcentration time and volume, which affect the time required for equilibrium; sample temperature, which alters vapor pressure and analyte solubility and the inherit vapor pressure of the analyte themselves. These variables determine how readily volatile compounds transition into the headspace. Moreover, solvent polarity can affect analyte

partitioning, while adjust pH or adding salts can reduce solubility or increase volatility, thus improving analyte transfer into the gas phase [2].

The equilibrium in static headspace sampling process can be characterized by the patriation coefficient (**K**), which represents the ratio of the concentration of the molecules between the two phases in equilibrium shown in Equation 1, it determines how a compound distributes between the two phases [1].

$$K = \frac{C_S}{C_G}$$

Equation 1

where Cs is the concentration of the sample in liquid phase and Cg is the concentration of the sample in gas phase, known as headspace, compounds with higher values of K will favor the liquid phase while compounds with lower values of K will favor the headspace, therefore in practice, it is essential to ensure that the analyte of interest has a significantly lower value of K than the interfering components in the sample matrix. Table 1 shows partition coefficients for various compounds between water an air phases at 60°C, the data indicates that light hydrocarbons are more easily extracted from water than liquid alcohols, highlighting how chemical properties affect extraction efficiency and stressing the importance of the matrix compound relationship.

The principle of static headspace sampling involves separating volatile compounds from a liquid or solid matrix by allowing them to partition into the gas phase, optimization of this technique requires adjusting conditions to ensure the analyte favors the headspace, improving sensitivity, which reduces chromatographic run time, and minimizes contamination from nonvolatile components.

A key parameter in static headspace optimization is the phase ratio where the concentration of the headspace is a relationship between K and the phase ratio β; the phase ratio of the system is a characterization of the volume of the analyte present in the gas phase versus the volume of the analyte in the liquid phase, shown in equation 2. The relationship between K and β, can be seen in equation 3[1].

$$\beta = \frac{V_G}{V_S}$$

Equation 2

$$C_G = \frac{C_O}{(K+\beta)}$$

Equation 3

Where Co represents the initial concentration of the analyte in the sample, the equation demonstrates that decreasing the phase ratio β results in an increased concentration of all compounds in the headspace, similarly reducing the partition coefficient K for example by raising the temperature of the solution drives more of the analyte into the gas phase. When both K and β are held constant across samples, the concentration of the analyte in the headspace becomes directly proportional to its original concentration in the liquid sample, enabling accurate and reliable quantification.

The dependance of K on temperature arises because K is inversely related to the analytes vapor pressure, as shown in Equation 4[1].

$$K = \frac{P_{total}}{P_i^o . \gamma_i}$$

Equation 4

Where p_i^0 is the vapor pressure of the pure compound *i* n the headspace vapor, γ_i is the activity coefficeint of the compound *i* in the sample matrix and Ptotal is the total pressure of the system, as temperature increases, the vapor pressure increases lowering K and shifting more analyte into the headspace. Another important consideration is the equilibrium time, the period required for the analyte to achieve phase equilibrium, while heating can accelerate this process, the required time depends on sample volume and composition, as the molecules take a finite time moving between the two-phase boundaries, to improve sample throughput, advanced instrumentation often allows multiple vials to equilibrate simultaneously [1].

Instrumentation for statistic headspace sampling typically consists of a sealed container, most commonly a closed vial, with a septum, and a temperature control mechanism such as a heating pad to maintain a constant sample temperature. Extraction of the headspace is generally performed manually using gas tight syringes, which withdraw a small volume of vapor from the vial and injects it into a standard gas chromatography (GC) analyzer, such as the Lonestar portable analyzer, a chemical detector capable of identifying trace compounds at part per billion (ppb) concentrations using Field Asymmetric Ion Mobility Spectroscopy (FAIMS)[3]. While this approach is cost effective and relatively simple, it presents several notable limitations, firstly the temperature control within the vial may lack precision and uniformity, potentially affecting analyte distribution, secondly manual syringe operation introduces variability and lacks reproducibility offered by automated systems. Additionally, vapor loss can occur during syringe withdrawal leading to reduced analyte recovery. Lastly because the syringe is typically at a lower temperature than the

vial, there is a high risk of sample condensation with the syringe, which can further compromise analytical accuracy.

1.2 Range of solutions

Both static headspace sampling and dynamic headspace sampling are widespread technique used in a myriad of different fields, including food, pharmaceutical and environmental. A key study conducted by Agilent Technologies, a leading manufacturer of analytical instrumentation including headspace sampling systems, provides valuable insight into the optimization of headspace gas chromatography for flavor analysis, using he Agilent 8890 GC system, the study focused on characterizing two distinct groups of volatile compound flavors in beer. This investigation serves as benchmark for understanding the operational parameters on the performance and reproducibility of headspace sampling. The study established a set of baseline conditions that enable efficient and consistent analyte extraction. With the primary condition being a specific oven temperature of 70°C, assisting in lowering the partition coefficient, a transfer line temperature of 100°C to ensure condensation does not occur, a vial equilibrium time of 30 minutes, certifying the compound migrates from the liquid phase to the gas phase, and lastly a flow rate of 3mL/min[4].

Beyond the baseline setup, the study emphasized the impact of secondary parameters on overall system performance and sensitivity, for example, vial shaking was identified as a critical enhancement to improve analyte partitioning by increasing surface interaction between the sample and headspace.

1.3 Project Design Overview

The headspace sampling device comprises two integrated systems, a heating system that uniformly heats the sample matric to promote the release of volatiles into the headspace and a distribution system that transfers the collected headspace to an analytical instrument, such as the MS. A high-level overview of the system is provided in Figure 2.

The heating system includes a heating element and a temperature sensor, with temperature regulation achieved through a PID algorithm. This setup allows the user to set and maintain a desired temperature via a graphical user interface (GUI). In contrast, the distribution system utilizes an air pump and flow meter to manage airflow through the system, while a solenoid valve enables precise control over the timing and delivery of the headspace to the MS inlet.

The following outlines the system's design specifications and objectives established prior to the commencement of the project:

- Temperature range: 24°C to 150°C
- Temperature Accuracy: 1°C
- Heating Element Power Consumption: 20Watts
- Temperature Rise Time: 20 seconds
- Air Flow Rate: 1 to 500mL/min
- Air Flow Rate Accuracy: 0.5mL/min
- Simple to navigate GUI
- Minium GUI response time: 30 seconds
- Neat Enclosure

The project spanned a total of 22 weeks, divided into two 12 weeks sprints. Detailed milestones and objectives are outlined in the Gantt chart located in Appendix 12.

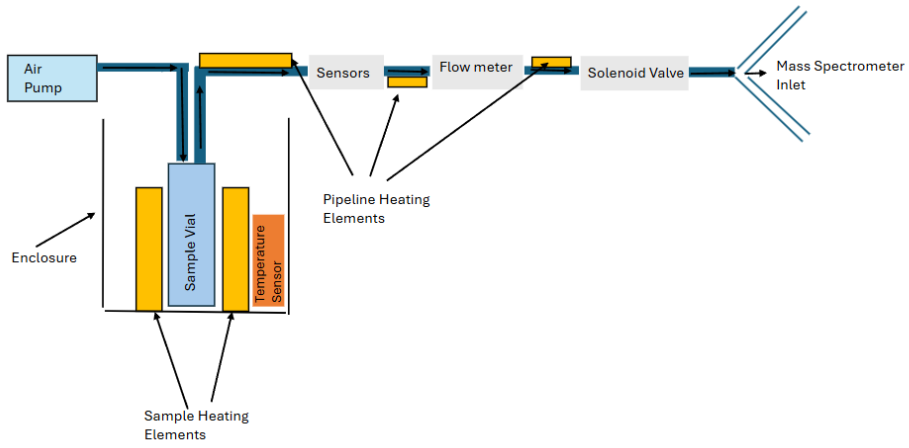


Figure 2. Automated Headspace Sample High Level Overview

2. System Design

2.1 Heating Element

The primary system to ensure the sample reaches equilibrium, i.e. moves from liquid phase to gas phase, is the use of a Kapton Polyimide heater; a flexible heating element made from Kapton, used to vaporize the sample, this is regulated by a proportional integral and derivative control system to ensure a setpoint is reached. The overall goal of the system would be to drive the temperature of a generic 22ml vial from ambient to a target

setpoint specified by a user, ideally 120°C, following Equation 5 that specifies the energy required for the resultant temperature change the required wattage for the heater can be determined, assuming the sample matrix to be comprised primary of water [1], using the specific heat capacity of water as 4200J/kg°C and substituting 1ml as equivalent to 1 gram, 8870 Joules of energy would be required for the desired temperature change of ambient to 120°C.

$$Q = mc\Delta T$$

Equation 5

$$Q = 0.022 \times 4200 \times (393.15 - 297.15)$$

$$Q \approx 8870J$$

With an ideal thermal inertia of 20 seconds 443.2Watts of energy would be required to raise the temperature to 120°C from ambient , the Kapton heater chosen would need to dissipate 8870J of energy to heat adhering to the project guidelines specifying a voltage limit of 24V and current of 3A, the KHLVA-2020/10[5] Kapton polyamide heater was chosen, primality for its high watt density and power rating of 10W/in^2 and 40W respectively, as well as its ability to flex, allowing an even thermal distribution to be applied to the vial which is round in nature, this system would theoretically achieve the desired temperature change within 3 minutes and 30 seconds.

To vary the power delivered to the heater pulse width modulation (PWM) is used; a series of high and low pulses, which allow the temperature to be varied. PWM operates by rapidly switching the load on and off, adjusting the average power supplied to the load, PWM consists of two elements, its corresponding duty cycle and switching frequency.

$$Average\ Voltage(V) = Peak\ Voltage(V) \times DutyCycle$$

Equation 6

$$DutyCycle = \frac{T_{on}}{T_{on}+T_{off}}$$

Equation 7

Duty cycle relates to the duration of “on time” to “off time”, varying the duty cycle varies the average power delivered to the load described in Equation 6 , the frequency of the PWM dictates the rate at which the power is cycled, typically in the range of tens of hertz to several thousand hertz, thermal elements usually have slow response times , as the temperature gradually increases in proportion to the power delivered, this slow response

time allows the use of a switching frequency between 10 and 1000Hz, a frequency range commonly offered by most micro controllers, although the required switching frequency can be achieved by the microcontroller unit(MCU), the power required to drive the load cannot, to source the required power a simple switching circuit is implemented, with the primary element of the circuit being a logic switch such as a transistor, to operate as a switch the switch is operated in two regions cut-off and saturation(linear), as a BJT are current dependent devices a MOSFET will be used for its simplicity with its ideal characteristics represented in Table 1.

$R_{ds(on)}$/Ω	I_{ds}/A	V_T/V	V_{ds}/V
<1	>2	<5V	>25V

Table 1. Ideal transistor characteristics

A low resistance between the transistors drain and source ensures less power dissipation during switching; with a low voltage threshold the switch can be operated by a micro controller without the need for additional circuits, from the specifications, the ideal transistor for operation is the STP55NF06L [6] with the following characteristics:

Manufacture code	V_T/V	$R_{ds(on)}$	$I_{ds}(on)$/A	V_{ds}/V
STP55NF06L	1.7	0.020	55	60

Table 2. STP55NF06L N channel MOSFET characteristics [6]

From the STP55NF06L transfer characteristics a gate voltage of less than 5V is sufficient to fully drive the MOSFET into saturation.

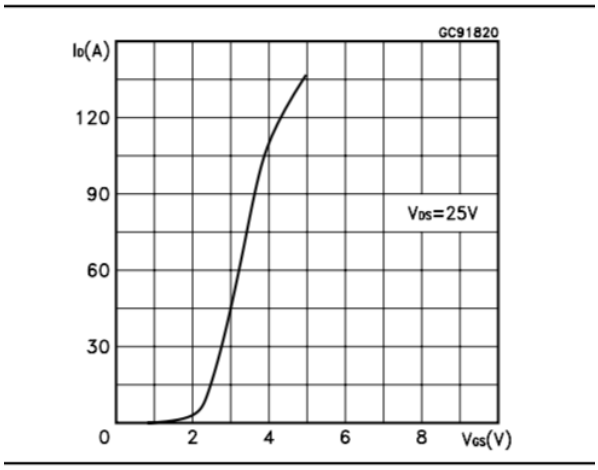


Figure 3. STP55NF06L Transfer characteristics [L]

Although, the transistor can be directly powered by the MCU, the MCU may not be able to source the required current to achieve specific switching speeds, due to the transistors

gate capacitance, whereby the current draw is proportional to the gate capacitance and the switching frequency.

$$I = \frac{Q_G}{T}$$

Equation 8

With the transistor connected to the MCU, during normal operation a voltage is applied to the MOSFETs gate, once the voltage is above the gate threshold voltage, the drain current starts to flow and the gate capacitance starts charging, the drain current increases proportionally as the gate current charges, until the gate capacitor is completely charged, the amount of current required to operate the transistor in saturation is described in Equation 8[7]. With a standard switching frequency of 1000KHz a current of 37µA is needed to operate the transistor, a current draw capable from most micro controllers which offer 20mA a maximum from its terminals, the overall device will use components that require switching frequencies greater than 1Khz increasing the overall current from the MCU, to reduce the current stress on the MCU a gate driver UCC23513QDWYRQ1[8] is implemented to compensate the required current draw, as well as offer optical isolation protecting the MCU in an eventuality of a short circuit, with the circuit application is described in Figure 4.

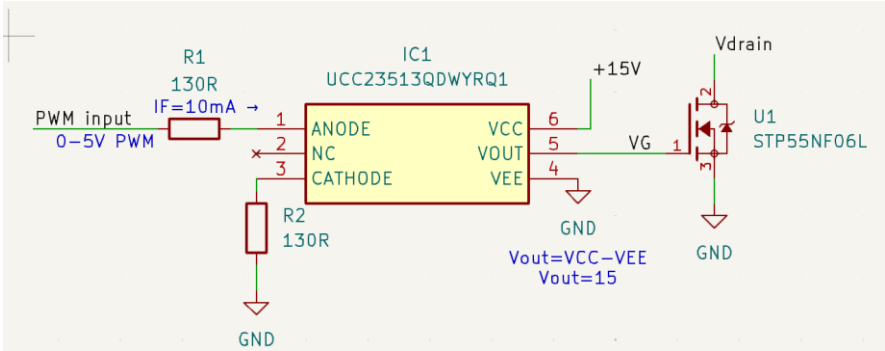


Figure 4. UCC23513QDWYRQ1(Optically Coupled Gate Driver Circuit)

A PWM signal of between 0 and 5V is applied to the anode of the gates drivers e-diode, and the cathode is connected to ground, following the data sheet specifications the total resistance between the anode and cathode is 260Ω allowing a current of 10mA to flow through the e-diode, the gate driver maximum propagation delay is 105ns allowing for maximum switching speeds of 9.5Mhz, the output voltage can source 4.5A, with a maximum of 35V, defined by VCC-VEE, absolute voltage rating for the transistors gate s 16V , applying 15V to VCC and VEE to ground, the output voltage achieves 15V ensuring the MOSFET is fully saturated.

The overall circuit architecture to power the heater is shown in Figure 5.

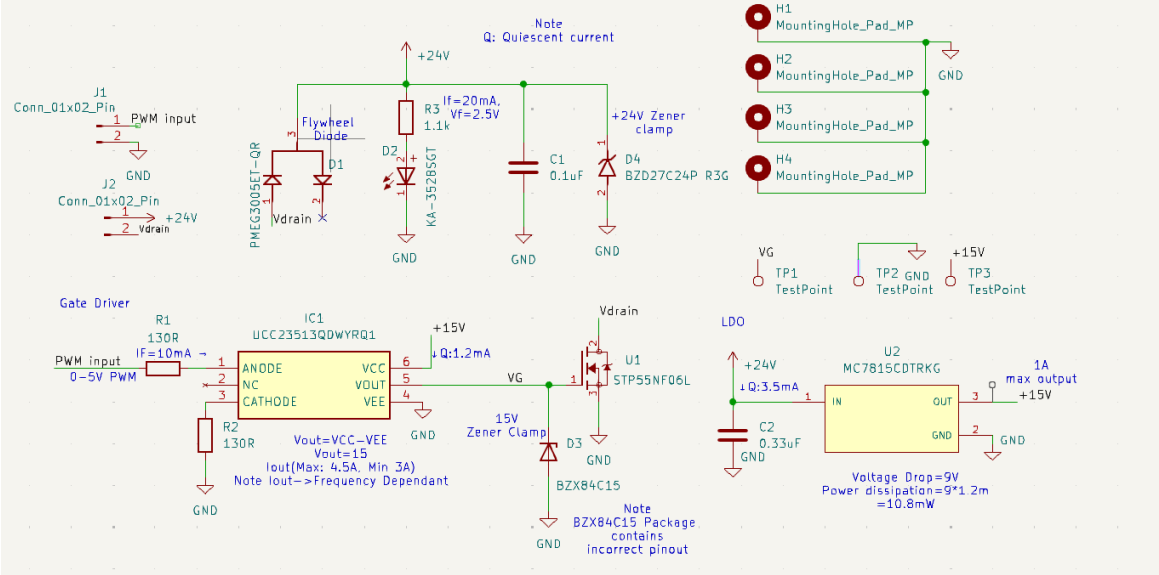


Figure 5. PWM Controller V1 Schematic Layout

The primary element of the circuit is the gate driver, other elements in the circuit are considered as peripherals to ensure the essentials operate as intended, component U2 is a linear dropout regulator (LDO), providing +15V to component IC1; opto-coupled gate driver, 24 Volts is sunk through the device with a quiescent current of 6mA providing +15V output voltage with a 1A maximum current sink, it achieves this by dropping the remaining 9V(9W) as heat.

Components C1 and C2 are bypass capacitors, which serve to enhance the circuits noise immunity. They achieve this by filtering out voltage transients, brief spikes or dips in voltage that can interfere with circuit performance, by smoothing the respective voltage potentials at their nodes, these capacitors help maintain stable power deliver and reduce the impact of high frequency noise.

Components D1, D3 and D4 server complementary functions. D1 is a Schottky diode, chosen for its fast-switching speed and low forward voltage drop, it is primarily used to suppress back Electromagnetic Interference (EMF); damaging voltage spikes that can occur when an inductive load, i.e. a motor, suddenly de energizes, these voltage spikes result from the magnetic field collapsing and inducing a reverse voltage in the circuit, the Schottky diode, provides a path for this reverse current, effectively clipping the spike and protecting sensitive components, components D3 and D4 are Zener diodes placed in reverse bias configurations, essentially clamping their respective voltage rails, ensuring they never exceed their safe operating thresholds. When the voltage at either node rises above the Zener diodes break down voltage, the diode conducts in reverse bias, diverting excess current and stabilizing the voltage.

Lastly, component D2 is a light emitting diode, a semiconductor device that emits light when forward biased, with an appropriate voltage potential, the luminosity of the diode is dependent on the current flowing through the diode, the illumination of the diode in the circuit architecture showcases that the primary +24V supply is present and active, schematic application is located in Appendix 1.

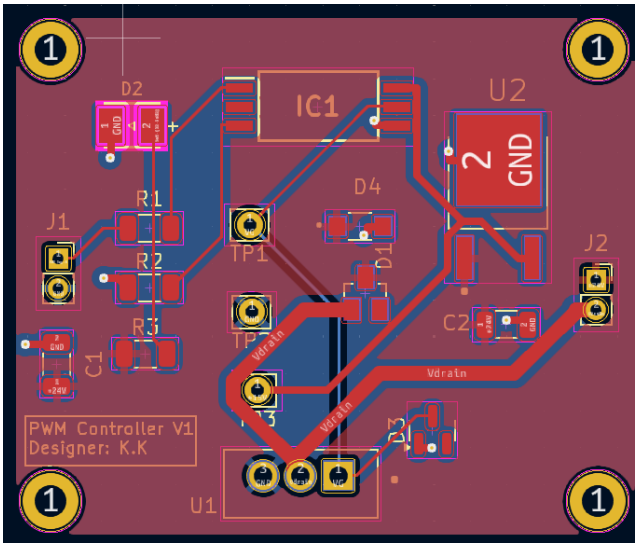


Figure 6. PWM Controller V1 Printed Circuit Board (PCB) layout design

The PCB layout illustrated in Figure 6 Is designed as a two-layer board, consisting of two copper layers, +24V and ground respectively and a dielectric layer in between them, component J1 provides the main system voltage and ground respectively, component J2 provides a connection for the heaters terminals. Bypass capacitors are placed such they are as close as possible to their corresponding components i.e. C2 is placed as close as possible to U2, providing a low impedance path for high frequency noise to be distributed to ground, two ground pours are used helping to optimize current return paths and reducing electromagnetic interference (EMI), these ground pours also minimize the need for long and complex routing by using vias; plated holes that electrically connect the top and bottom layers of the PCB, allowing current to flow through vertically through the board instead of across long traces,, additionally traces such as “Vdrain” that experience high power signals; 72Watts have greater widths ensuring appropriate current distribution, lastly, the resistor and capacitor package sizes are set to 3216 metric making them easy for manual soldering, ensuring a high assembly yield and efficient PCB turnover.

2.2 Micro Controller Unit

The ESP32 Devkit V1 has been selected as the microcontroller for both the design and testing phase of the project, the development board offers a wide range of input and output

interfaces, illustrated in Appendix 2, along with built in Wi-Fi and Bluetooth capabilities, all packaged in a compact form factor. These features make it highly suitable for space constrained embedded applications where connectivity and versatility are essential, in contrast to the original choice the Arduino Uno Wi-Fi Rev2, which also supports Wi-Fi connectivity, and offers a larger pinout, its bulkier physical footprint makes it less ideal for integration into smaller designs.

The ESP32's slim profile and dual core processing power gives it a distinct edge in modern internet of things (IoT) automation projects, its native ability to operate as an access point is an additional compelling feature over the Arduino Uno Wi-Fi Rev2 allowing other devices to connect remotely to the microcontroller without the need for an external router, enabling the user to provide remote monitoring or control platforms, the board also integrates several advanced features including , a real time clock (RTC), a 12 bit analogue to digital converter (ADC), for precision sensor reading, and a 32 bit RISC architecture, which contribute to an efficient and high performance system, despite the powerful capabilities provided by the ESP32 it remains at a lower cost compare to its counterpart the Arduino Uno Wi-Fi Rev2, solidifying its suitability for both prototyping and scalable product development.

The ESP32 development board provides a wide range of general-purpose input/output (GPIO) pins, many of which provide PWM, I2C, and SPI functionality, out of the boards 30 available pins, 25 are programmable, the board supports 16 PWM channels, divided into 8 high speed and 8 low speed channels, these channels can be configured with a maximum resolution of 16 bits a maximum frequency of 40Mhz, enabling precise control over duty cycles and signal modulation for motors, LEDs and other system peripherals, the maximum current sink per PWM pin, is 40mA, with a recommended operating current of 20mA, to avoid damaging or overloading the microcontroller pins. Additionally, the PWM frequency and resolution can be configured however not independently, increasing one will affect the limits of the other, as the PWM resolution is a function of the frequency and the system clock frequency, where the product of the resolution and frequency must not exceed the clock frequency of the timer used, seen in Equation 9 [9].

$$PWM\ frequency \times 2^{PWM\ resolution} < Clock\ frequency$$

Equation 9

The remaining five pins on the board consist of an enable (EN) pin, two ground pins, a 3.3V power output and a voltage input pin (Vin). The 3.3V pin can supply a maximum current of up to 1A, only when the board is powered through an external supply, not including the USBC cable, the ESP32 micro controller is capable of reading analogue voltages from 0 to

3.3V. It utilizes a 12-bit ADC which maps the input range to digital values from 0 to 4095 providing a resolution of 0.8mV per ADC count, although the ESP32 offers 15 ADCs through pin multiplexing, the reading is nonlinear [10], where there is no distinction between 0V and 0.1V and 3.2V and 3.3V.

An interposer board is designed as an interface fixture for the ESP32development board, with its schematic illustrated in Figure7, engineered to supply the necessary voltage potentials to power the ESP32 micro controller as well as the different PWM controller boards, that interface with the system peripherals. To enhance usability and system monitoring, indicator LEDs are incorporated to visually confirm the presence of each voltage rail, additionally, bypass capacitors are strategically placed near the power delivery points to suppress voltage transients and electrical noise, ensuring stable and reliable operation of all connected components, power regulation on the interposer board is achieved through the use of buck converters; a step down DC to DC converter that takes a high voltage potentials and subsequently reduces it to a lower voltage potential through high switching frequency techniques , a single primary voltage input of 24V is supplied to the board, which is then stepped down to the required operational voltages, of 12V, 5V and 3.3V, with the buck converter detailed in Appendix 9.

A similar layout strategy is employed as in the PWM Controller V1, including component placement, the selection of standardized component footprints, the use of copper pours and vias, where the copper pours are utilized for efficient thermal dissipation and low impedance power distribution, while vias provide electrical connectivity between board layers, aiding in both signal integrity and power routing.

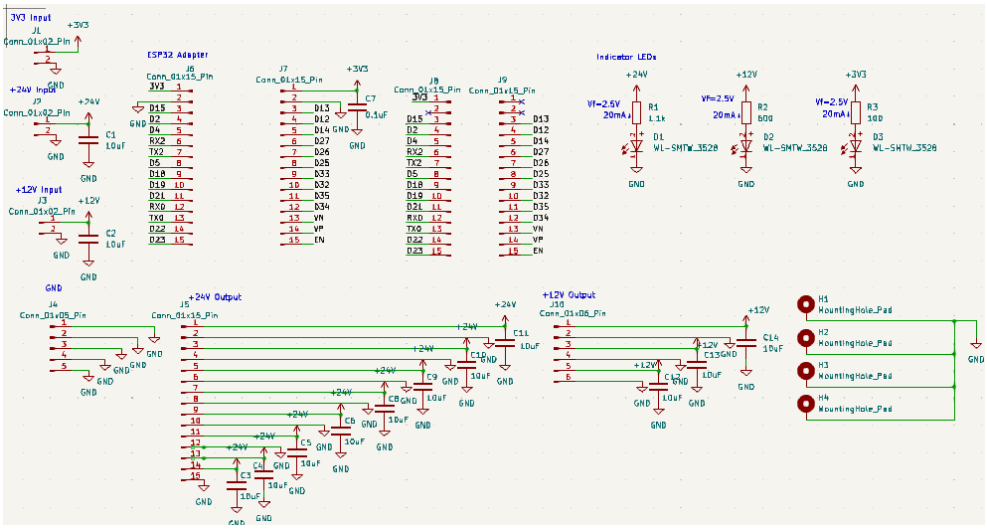


Figure 7. ESP32 Interposer Schematic

2.3 PWM Controller V1 Review

The first iteration of the voltage controller board successfully amplifies the input PWM wave to the required voltage level with little to no loss, however, contains minor errors being:

a. The board needs an additional connection for +24V and GND, ideally a 4-pin input connection.
b. Current limiting resistors need to be added to the +24V Zener clamp, the components total power dissipation is 1W, with 24V nominal input voltage the maximum current that can flow through the diode in reverse bias is 41mAm adding a 1.1kΩ resistor in series with the diode's cathode limits the current to 22mA.
c. The 15V Zener clamp also requires a current limiting resistor of 15kΩ limiting the current flowing through the device in reverse bias to 13.75mA ensuring a power dissipation less than 330mW.
d. A current limiting resistor of 2.4kΩ needs to be added to component D1; Schottky diode, limiting the current flowing through the device in reverse bias is 11.67mA, ensuring a power dissipation less than 280mW.

The errors and inconsistencies experienced by PWM controller V1 are handled by PWM controller V2 with its schematic design presented in Appendix 3.

2.4 Temperature Sensor

The system feedback to regulate the temperature is a temperature sensor, the system specification specifies an error margin of 1°C and an ideal temperature rise time of 20 seconds, to measure the temperature there exists a plethora of sensors, including:

- Thermocouple
- Resistive temperature detector (RTD)
- Thermistor
- Semiconductor based IC.

Thermocouples operate of the principle that when two conjoined dissimilar metals experience a temperature change a corresponding potential is generated which is then passed to an analogue to digital converter after amplification, they offer high temperature ranges and fast response times however have low accuracy 0.8°C to 3.5°C depending on the type of thermocouple used[11][12], RTDs have a positive temperature coefficient offering a near linear temperature response, they offer high temperature ranges and accuracy but slow response times, an example being the NB-PTCO-126[13], with a response time of 1.2 seconds to 22 seconds depending on the sensors dimensions, thirdly,

thermistors are a semiconductor type of resistor whose resistance is strongly dependent on temperature, their exists both a negative coefficient thermistors (NTC) and a positive coefficient thermistors (PTC), they are nonlinear devices emphasizing on the need for signal conditioning, their accuracy is less than that of a RTD and have a limited temperature range, however have faster response time usually less than 1 second. Lastly a semiconductor temperature sensor is an electronic device capable of measuring temperatures accurately within a limited range, they use temperature sensitive diodes to provide a linear response, they have generally low accuracy (1-5°C) and slower response times [14].

Although the semiconductor temperature sensor falls in terms of temperature range accuracy and response time, it offers an average of all the mentioned characteristics in a smaller package size, this enables the integrated circuit (IC) to be placed in small or tight spaces, additionally as a semiconductor IC there is no need for additional signal conditioning, the device operates as a plug and play only needing a analogue to digital converter provided by all MCUs, making it simple to integrate in the overall system architecture, the devices slow thermal response time can be compensated by placing the sensor as close as possible to the mass being heated, preferably making contact with the mass.

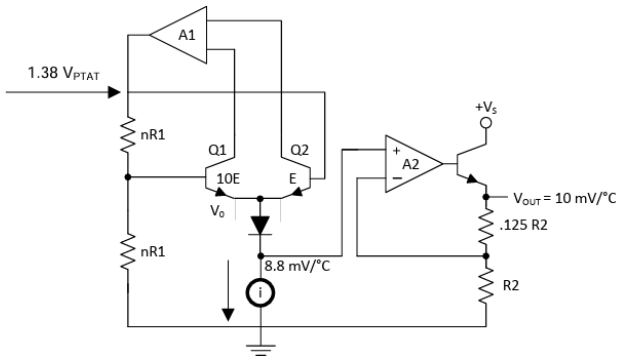


Figure 8. LM35 Functional Diagram [15]

The LM35 is a bandgap temperature sensor that operates based on the temperature dependance of the base emitter voltage, of two connected transistors, Q1 and Q2, the sensor leverages the principle that the base emitter voltage of a bipolar junction transistor varies predictably with temperature, a behavior described by Equation 10.

$$\Delta Vbe = \frac{K_T}{q} \times \ln\left(\frac{I_{Q1}}{I_{Q2}}\right)$$

Equation 10

Where, VBE is the base emitter voltage, k is Boltzmann constant, T is the absolute temperature in kelvin, q is the charge of an electron and IQ1 and IQ2 are the emitter current of the transistor, respectively, a constant current source I biases the transistors and the emitter area of Q1 is ten times that of Q2, this mismatch causes a temperature dependent voltage difference to develop between the transistors, the diode connected transistor provides a stable reference voltage where the amplifier A1 rectifies and amplifies the resulting differential signal, subsequently, amplifier A2 conditions the output, converting the signal into a linear analog voltage that corresponds to temperature, the output is scaled to represent the temperature in either degrees Celsius or Fahrenheit, depending on component variation of A2.

The accuracy of the sensor corresponds to the transfer function in equation 11, where Vout is the LM35 output voltage and T is the temperature in °C [15].

$$\text{Vout}=10\text{mv}/°\text{C} \times \text{T}$$

Equation 11

The analogue output from the LM35 temperature sensor is referenced to one of the micro-controller ADCs, the ADC converts this voltage into a digital value, which serves as the process variable (PV) in a Proportional Integral Derivative (PID) control algorithm. The PID controller continuously compares this PV against a predefined temperature setpoint and dynamically adjusts the system's output to regulate the temperature accordingly, the PID algorithm is a closed loop feedback system, designed to minimize the difference, referred to as the error, between the measured PV and the setpoint, this error is processed through three distinct components, the proportional term described in Equation 12, the integral term in Equation 13 , and lastly the derivative term in Equation 14. The combination of these three terms seen in Equation 15, provides an optimal and corrective output that drives the temperature as close as possible to the setpoint, ensuring stable and accurate thermal and airflow regulation of the system.

$$K_P \times e(t)$$

Equation 12

$$\int_0^\tau K_i \times e(t)dt$$

Equation 13

$$K_d \times \frac{d}{dt}e(t)$$

Equation 14

$$output = K_p.e(t) + \int_0^\tau K_i\,.e(t)dt\ + K_d\,.\ \frac{d}{dt}e(t)$$

Equation 15

The Proportional (**P**) term addresses the current, or instantaneous, error in the system, it produces a corrective response proportional to the magnitude of the error, ensuring a stronger reaction when the PV is far from the desired value, the Integral (**I**) term accounts for the cumulative sum of past errors over time, beginning from the start of the process. By integrating the error, it helps eliminate residual steady state error, ensuring the PV reaches and maintains the setpoint, lastly the derivative (**D**) term focuses on the rate of change of the error, predicting future system behavior by assessing how quickly the error is increasing or decreasing, this action allows the controller to apply damping, reducing the likelihood of overshoot or instability in the control loop, the weighted sum of these components provide the signal used to drive the Kapton Polyamide Heater and Vacuum Pump, respectively.

The calibration of the PID coefficients is performed to achieve dynamic system performance objectives, such as desired rise time, settling time, overshoot, and steady state error. Proper tuning ensures the system remains responsive, stable, and precise under varying operating conditions.

2.5 Magnetic Stirrer

A magnetic stirrer is used to agitate the vial contents; the system operates using a direct current (DC), permanent magnet motor, in conjunction with a neodymium magnet attached to the motor shaft and a smaller inert magnet placed inside the vial. When the DC motor is energized, it causes the shaft, and subsequently the neodymium magnet to rotate. This rotation produces a dynamic magnetic field, which interacts with the inert magnet inside the vial, effectively stirring the solution without direct physical contact.

The speed of the DC motor is controlled via a PWM signal, the signal is generated by the ESP32 microcontroller at a frequency of 5Khz, and is delivered to the motor through a modified version of the PWM controller V2, in this variant, the transistor configuration utilizes high side switching, where the transistor is placed above the load in the circuit, connecting the positive voltage supply to the motor. This ensures that the load, remains referenced to ground, avoiding a scenario where the load is floating from ground, this configuration is detailed in the schematic diagram available in Appendix 3.

2.6 Headspace Distribution

The headspace distribution system is the secondary component of the architecture, key to ensuring the headspace travels to the mass spectrometer, comprised of three components, a vacuum pump, a flow rate sensor and a solenoid valve.

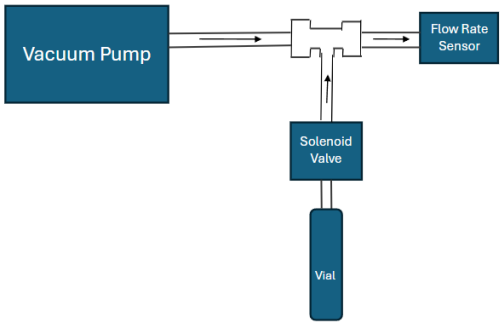


Figure 9. VOC extraction unit

An inexpensive vacuum pump is used to create the necessary air flow required to distribute the headspace generated from the vial, this technique follows the principal design discussed in using an Arduino based air constant pressure fluid pump [16], in the study a vacuum pump intake is used to draw air from the lab atmosphere, and the outtake expels the air to a separate air chamber, solenoid valves or similar with the addition of pressure sensors, create a closed loop system to control the air flow rate of the system.

From the plethora of vacuum pumps available, the primary criterion for selection was an oilless pump, lubrication would contaminate the air during the compression process contaminating the VOC profile, using the criteria the D2028B Spark fun DC vacuum pump was chosen [17], it's a membrane/diaphragm pump, providing 12 to 15LPM air flow when sourced with 12V, the pumps airflow is varied using a PWM signal from the ESP32 controller, in conjunction with the PWM Controller V1 board to source the required 12V to energize the pump.

An airflow rate sensor is used to provide feedback, following the vial dimension in Appendix 4, the expected flow rate is 0-0.1Litres per minute, the sensor used is the D6F-P0001A1 flow rate sensor[18], the sensor is powered by 5V, with a quiescent current of 15mA, the expected output voltage vs air flow can be seen in Figure 10.

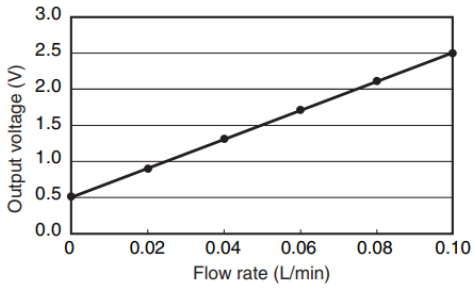


Figure 10. D6F-P0001A1 Expected Output Voltage vs Flow Rate

The sensor mounting configuration is that of a bamboo joint seen in Appendix 5.

The last component is the solenoid valve, the VX212AZ1D a normally closed solenoid valve [19], with a 2mm diameter orifice, the component is primally comprised of brass, meaning it shouldn't affect the VOC profile, as an inductive load that operates using 12V, the solenoid valve is actuated using the PWM controller V2 board with the transistor configuration operating as a high side driver, as seen in the schematic diagram in Appendix 3.

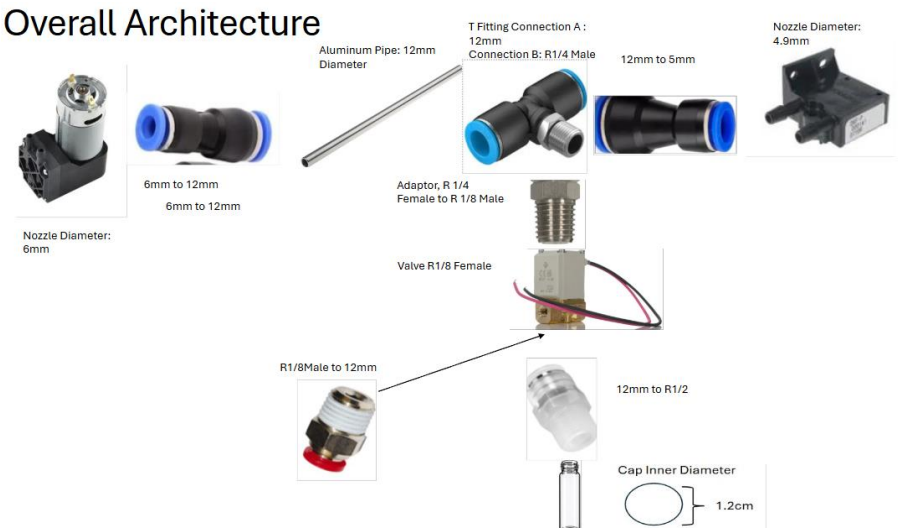


Figure 11. VOC extraction unit component architecture

The overall system to connect the various components of the distribution system should be primary be comprised of brass or aluminum interconnects to ensure the VOC profile remains uncontaminated.

For initial testing, standard polyethylene piping was used to evaluate the system's functionality, with the complete system configuration shown in figure. The system architecture integrates the various physical components of the system into a cohesive operational setup. An array of 3D printed structures supports the main elements of the system, with the PCBs mounted onto 3D printed boards to ensure electrical isolation from the base footprint, reducing the risk of short circuits through the mounting screws. The vial is housed in an aluminum enclosure, which improves the thermal distribution of the Kapton heater secured to the inner diameter of the enclosure. For the magnetic stirrer to function, the vial is vertically offset using a 3D printed stand, with the motor positioned directly below the vial enclosure, with a custom 3D printed attachment is designed to attach the neodymium magnet to the end of the motor's shaft. Lastly separate 3D printed

attachments, were designed to mount the solenoid valve and vacuum pump to the base footprint as well as provide component stability allowing the connection of the pipes and push fit connectors, drawing of these attachments are provided in Appendix 10.

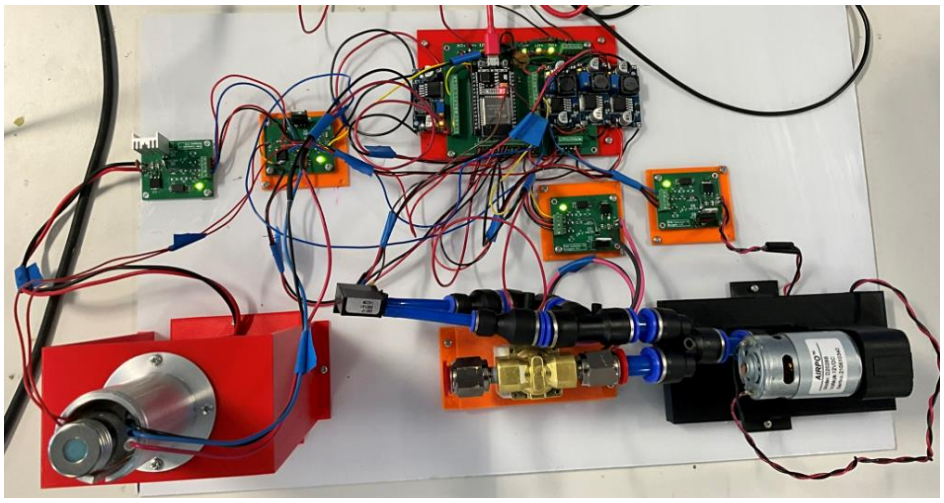


Figure 12. Complete Circuit Architecture

The first of the physical housing elements of the design is the vial chamber, the vial chamber, the vial chamber drawing, illustrated in Appendix 10, is designed to securely contain the 22ml vial with the aluminum fabrication seen in Figure 13. The drawing represents the revised version of vial chamber V1, with the initial 3D print shown in Figure 13. The original design exhibited several critical issues, firstly the orifice intended to accommodate the vial was too small, fitting only the vial and the Kapton heater but leaving insufficient space for the temperature sensor; the wall thickness surrounding the vial was too thin, resulting in poor heat retention; the base was also too thin compromising the overall structure durability, lastly, the mounting hole sizes were undersized, leading to inadequate retention, when attaching the chamber to a base footprint. These issues were addressed and corrected in the revised version to improve functionality, thermal performance and mechanical stability.

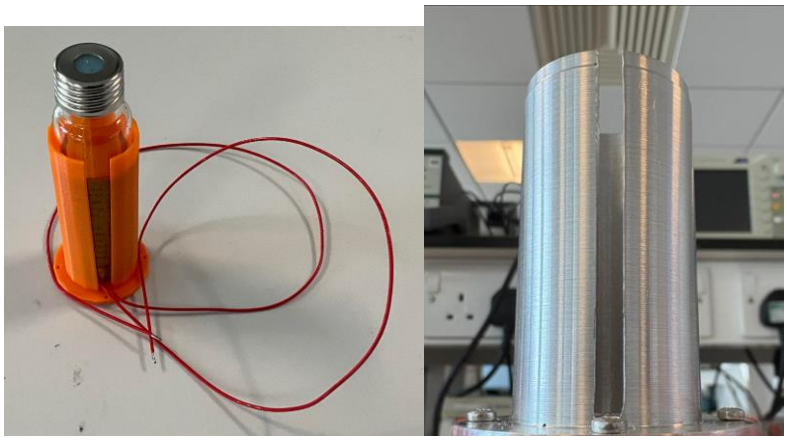


Figure 13. Vial chamber V1 and aluminum fabrication respectively

The next primary housing elements of the system is the vial stand, shaft mount and the motor holder showcased in Figure 14.

Appendix 10 showcases the drawing of the vial stand, which is designed to vertically offset the vial, allowing a motor to be positioned directly beneath it. The primary feature of the stand includes a central hole where the vial seats, enabling the neodymium magnet to interact with the inert magnet in the vial solution. Mounting holes are incorporated into the design to securely attach the stand to a base footprint, extra holes are also positioned along the sides of the stand, intended to server as fixtures for routing or securing piping as needed.

Adjacent to the vial stand drawing is the design for the shaft mount, which is used to attach the neodymium magnet to the motor shaft, designed for 3D printing only due to overly complex geometry.

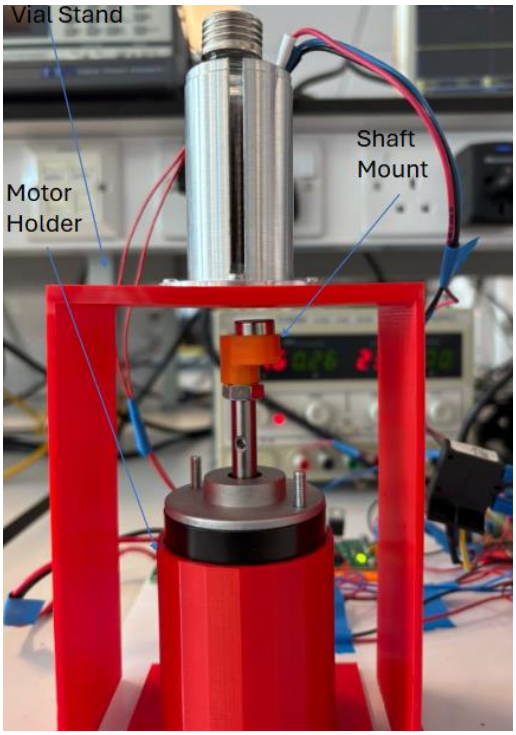


Figure. 14 Vial Stand, Motor Shaft and Shaft Mount Assembly

The vacuum pump is mounted on a custom holder, illustrated in Figure 15. This holder is designed not only to support the vacuum pump but also to accommodate a push fit connector for easy piping connections. Additionally, it features an integrated groove to securely position a 12mm pipe, it also includes mounting holes for a bracket, allowing the pipe to be firmly fixed in place to prevent movement during operation, the solenoid valve holder achieves the same functionality respective to the solenoid valve, also illustrated in Figure 15.

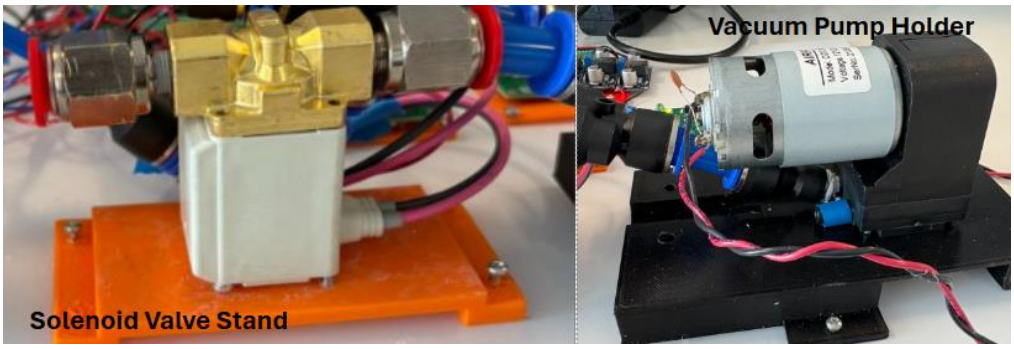


Figure 15. Solenoid Valve Stand and Vacuum Pump Holder respectively

2.7 Microcontroller Firmware

The firmware consists of two primary elements; the first element is the PID element, responsible for adjusting the power delivered to the heater, motor and pump, it's comprised of feedback and writing processes, with the feedback processes being the temperature logger, as well as the air flow logger, both of which return the temperature of the vial and flow rate of the vacuum respectively, the writing process involves using the PID coefficients as well as the temperature and air flow rate feedback values to write an appropriate PWM to the Kapton polyamide heater and Vacuum pump respectively.

The temperature logger function reads the ADC pin connected to the Vout of the LM35; the ADC value is then converted into the appropriate voltage using Equation 16, returning a voltage in millivolts, which is converted into a corresponding temperature value in degrees Celsius by diving the voltage by 10 as seen in Equation 11, as the potential received has a resolution of 10°C per millivolt according to the LM35 Datasheet [15].

$$millivolt = ADCValue \times \left(\frac{ADC_{Reference}}{ADC_{Resolution}}\right)$$

Equation 16

The corresponding value from the temperature reading function is passed to a variable "temp_set", used as a setpoint in the PID function.

The Flowrate sensor function follows a similar methodology, the D6F-P0001A1 flow rate sensor outputs a voltage, the voltage is referenced to a corresponding ADC pin of the ESP32 controller, providing an equivalent ADC value, which is then converted to a voltage using Equation 15, the value is then used in a similar PID function, respective to controlling the air flow of the vacuum pump.

The ESP32 is configured as an access point with a predefined service set identifier (SSID) and password, allowing the user to generate an Accesspoint accessed using the credentials defined in the function, the Accesspoint enables the user to generate a webserver, used to monitor or directly interact with the peripherals connected to the microcontroller through the use of a web hosted graphical user interface (GUI), the system is comprised of four files, with the firmware being the ESP32 sketch file, additional files including the JavaScript, Html and CSS files that are used to maintain the webserver that hosts the GUI.

The secondary element utilizes all four described files. A temperature form handler, defined in the HTML file, generates a simple slider that allows the user to adjust the desired temperature. When the form is submitted, it sends a HTTP GET request to the server with a query parameter named temperature “temp”. This request is handled by the ESP32 web server defined in the ESP32 sketch. The sketch reads the value associated with “temp” parameter and assigns it to the variable “temp_set”, which us then used as the setpoint in the PID control loop, a similar methodology is used for the motor duty cycle, the HTML file defines a slider input which allows the user to set a desired duty cycle. When submitted, the form sends a HTTP GET request with a “motor” parameter. This request is routed to the”/setMotor” endpoint in the ESP32 sketch, the sketch passes the “motor” value to a variable that is then used to control the PWM output adjusting the motor’s duty cycle accordingly. This same architecture is followed by the air flow control feature using a slider defined in the HTML file, and an endpoint in the ESP32 sketch to appropriately adjust the airflow.

The solenoid valve is managed using a toggle switch in the HTML file, enabling the user to activate or deactivate the valve accordingly, when toggled, a JavaScript function is triggered which sends an XMLHttpRequest(XHR) to either turn on or turn off the solenoid valve. Additionally, a polling function runs every second, sending a GET request to “/stateSolenoid” to confirm the state of the solenoid valve. The ESP32 sketch processes these requests and sets the digital output on pin 14 accordingly.

A graph illustrated in Figure 16, is used to display real-time temperature values from the LM35 temperature sensor. A JavaScript function utilizes the XHR feature to fetch sensor readings from the ESP32 every second via the “/readings” endpoint”. Each retrieved value is parsed from the JSON response, converted into a floating-point number and stored in an array called “dataPoints”. To maintain a consistent number of data points, the oldest value is removed once the array exceeds a predefined limit. The temperature data is then plotted on a canvas element as a simple line graph, dynamically redrawn each second using the “dataPoints” array, providing a continuously and visually responsive representation of temperature changes over time.

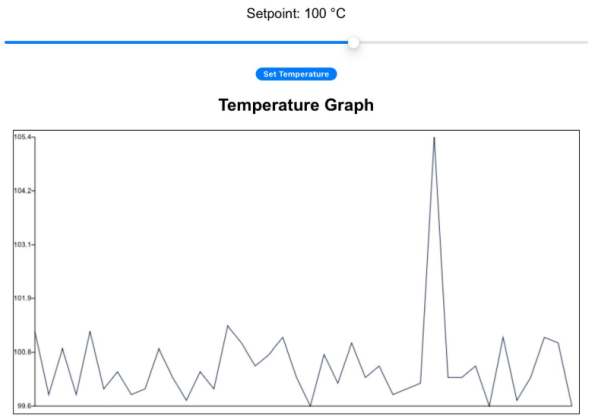


Figure 16. Webserver graphical user interface temperature graph

The responsiveness of the web interface is enhanced by intercepting form submissions related to temperature, flow rate, and motor duty cycle, and replacing the default page refresh behavior with (asynchronous JavaScript and XML) AJAX-based HTTP requests, each form submission is handled using an even listener attached to its submit event. Within the event handler, the method “e.preventDefault()” is called to prevent the browser from reloading the page when the form is submitted. Instead, a JavaScript function creates and sends an XHR to the appropriate endpoint on the ESP32 sketch, passing the selected value as a query parameter, preventing disruption to the live temperature plotting function, allowing continuous data plotting.

A simple GUI illustrated in Figure 17, is generated providing, a simple to navigate architecture, due to the unique HTML elements the webpage is available to devices of any form factor. the respective files can be found in Appendix.

Figure 17. Graphical User Interface (GUI)

3. Results and Discussion

3.1 Circuit Analysis

The Assembled PCBs for PWM controller V1 and PWM controller V2, as shown in Figure 18, were completed by populating each board with their respective components. The illumination of the green status LED on each board confirms successful operation and indicates that both systems are function as expected. Application level modifications required to ensure proper board operation are detailed in Appendix 6.

Figure 18. Assembled PWM controller V1 and PWM controller V2 respectively

A thorough analysis of the boards was conducted to verify operation of the PCBs within expected parameters and tolerances. This involved probing various points on the PCB using an oscilloscope and cross referencing the results with the schematic diagram, during this investigation, several anomalies were identified which are detailed below.

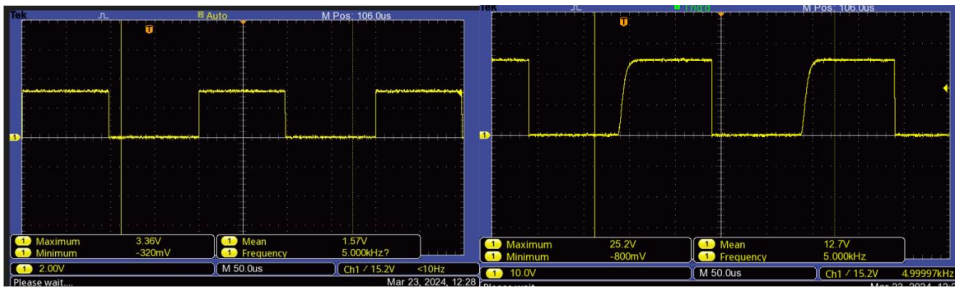


Figure 19. 3.3V PWM input and 24V PWM output to vacuum pump; 50% duty cycle 5khz square wave

The MCU; ESP32 provides a 3.3V PWM wave with a frequency of 5Khz and a duty cycle of 50% measured from resistor R2, the corresponding output from the PWM controller V1 is a positive 24V square wave.

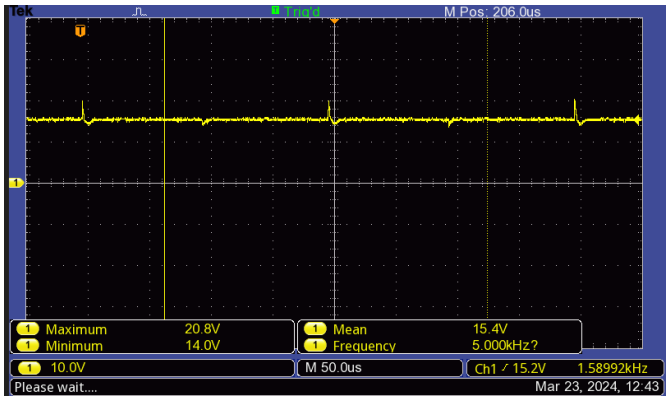


Figure 20. +15V LDO (Linear Dropout Regulator) output

The LDO generates a 15V output, however, it exhibits transient spikes, with a voltage of 20.8V and a minimum of 14V. This behavior indicates additional smoothing is required on the 15V rail. To address this, a larger capacitance can be added in parallel to better filter out voltage fluctuations. Furthermore, a voltage clamp utilizing a Zener diode can be implemented to ensure the output voltage does not exceed 15V, providing an extra layer of protection against overvoltage conditions.

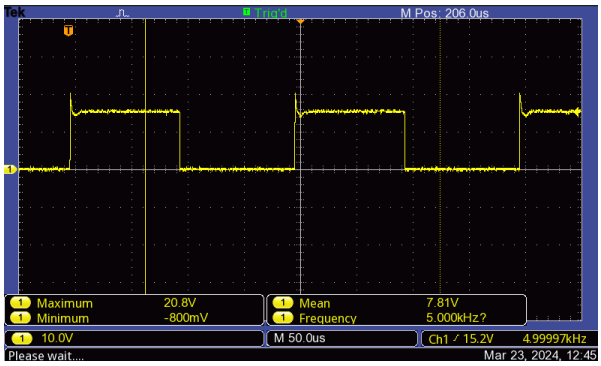


Figure 21. STP55NF06L Gate voltage; +20.8V

The gate voltage of the MOSFET STP55NF06L exhibits multiple overshoot spikes at on the onset of each pulse, with some exceeding the transistors maximum gate voltage of 16V, which could potentially damage the transistor. To mitigate, this a bypass capacitor can be added at the gate to suppress high frequency noise, however this would form a low pass filter, restricting the frequency response of the circuit.

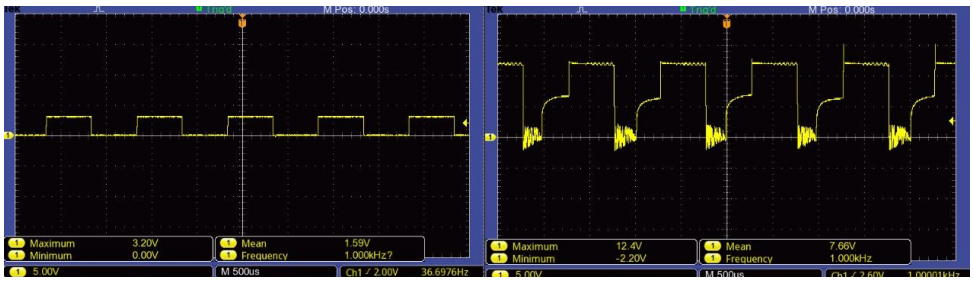


Figure 22. 3.3V PWM input and 24V PWM output to motor; 50% duty cycle 5khz square wave

During motor operation, voltage ringing is observed at the motor's input terminals, which interferes with the performance of the overall circuit, including disruptions to the GUI illustrated in Figure 24, where the noise generated creates extreme digital temperature outliers, to mitigate this, a 0.1µF capacitor is placed across the motor terminals, see Appendix 8. Additionally, twisting the motor leads together and minimizing their length further helps to reduce the transmission of voltage noise to the ESP32 controller board.

Similarly, during the vacuum pump operation, as substantial amount of electrical noise is produced as shown in Figure 23, direct probing of the PWM output from the microcontroller reveals that the noise propagates from the vacuum pimp back into the microcontroller, which compromises the reliability of the control signals, addressing the need for additional noise mitigation strategies.

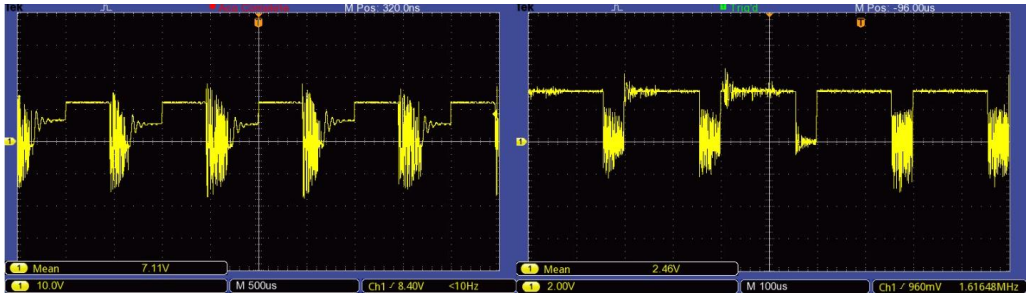


Figure 23. 12V PWM input to vacuum pump and output from microcontroller

This noise disrupts other system functions such as the GUI, displayed in Figure 24, the noise generates extreme digital outliers in the temperature plotting, also affecting the PID function, as erroneous temperature data is passed to the PID function.

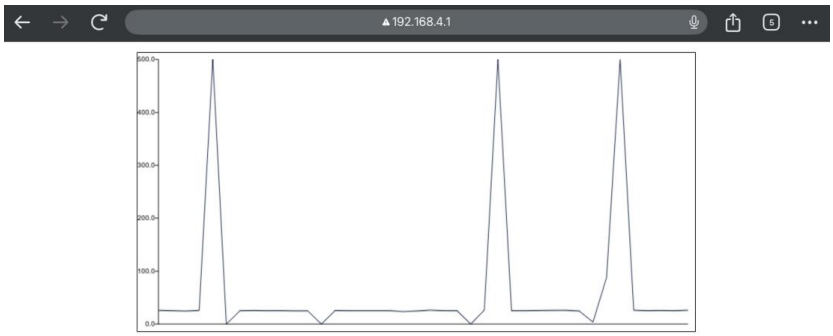


Figure 24. Graphical user interface temperature graph digital noise transients.

When the ESP32 microcontroller is connected via USB-C for programming, an artifact is observed, as shown in Figure 25, where the LEDs on the connected PCBs illuminate, falsely indicating the presence of valid system voltage potentials, noticeable when the

main power leads are not connected to an external power source, the system is designed to step down a 24V input from the main power leads to a .3V rail, using buck converters. However, when the USB-C is connected, the 3.3V line receives power from the USB source, which then back feeds to the rest of the system. This unintended power path can result in excessive current draw from the PCBs through the USB line, posing a risk of damaging the ESP32 microcontroller. To mitigate this issue a reverse biased diode can be added to the 3.3V line to prevent current from flowing from the 3.3V line to the buck converter.

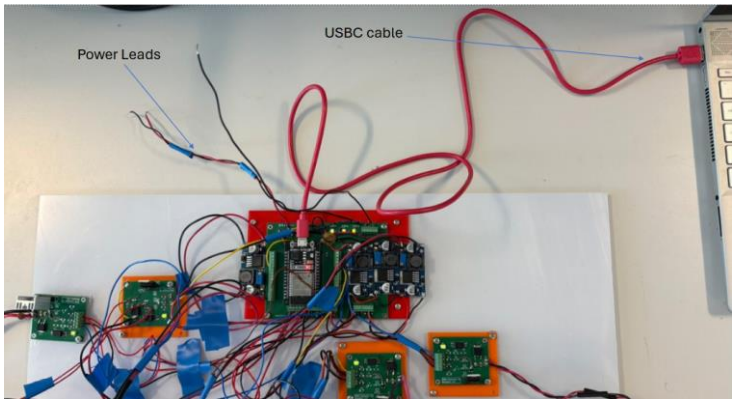


Figure25. EP32 Interposer board power artifact

3.2 Vial heating

The vial is placed in an enclosure with the Kapton polyamide heater wrapped around its circumference, with the leads connected to the PWM controller board, the LM35 temperature sensor is placed inside the enclosure ensuring the heater is flush against the vial.

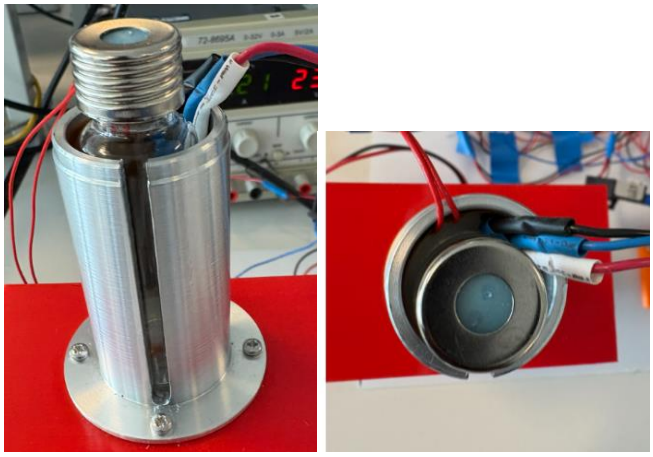


Figure 26. Vial enclosure with Kapton Polyamide Heater wrapped around its circumference, and LM35 temperature sensor

A small volume of water is placed in the vial and the setpoint is set to the desired temperature, the final PID coefficients used to obtain the data in Figure 25 are:

$$K_p = 10$$

$$K_i = 1$$

$$K_d = 0.01$$

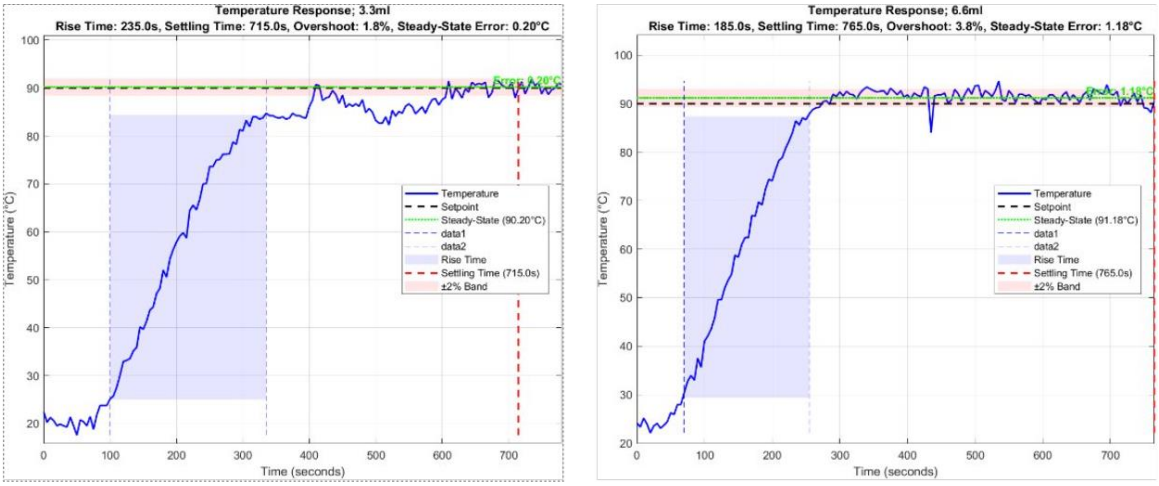


Figure 27. Temperature response with vial volumes at 3.3ml and 6ml respectively

Two distinct volumes of water were used for the experiment, one where the volume is at 3.3ml and one with the volume at 6.6ml, in both instances the setpoint of the temperature is set to 90°C.

Average Rise Time/s	Average Settling Time/s	Average Percentage Overshoot (%)	Average Steady State Error/°C
210	740	2.8	0.69

Table 3. Average system metrics; rise time, settling time, percentage overshoot and steady state error

The average time it takes the system to heat a volume of water from ambient; 24°C to 90% of the setpoint;81°C is 210s, the temperature then settles within 740s, with an error margin of 0.69°C, the system metrics highlight for an increase in the proportional coefficient to reduce the rise time, and a reduction in the integral coefficient to reduce the steady state error .

As shown in Figure 26, an air gap is present between the Kapton Polyamide heater and the vial, primarily caused by the physical form factor of the LM35 temperature sensor, this gap compromises the heat transfer of the system and suggests for a different sensing approach. Implementing a thermocouple, with its leads directly attached to the surface of the vial, would improve the thermal contact and measurement accuracy. Additionally, the dimensions of the vial enclosure could be optimized and reduced to ensure the vial seats

flush against the Kapton Polyamide heater, increasing the efficiency of the heat transfer and overall system reliability.

3.3 Magnetic Stirrer

The rotation of the motor, with a neodymium magnet mounted at the end of its shaft, works together with the inert magnet placed in the solution to achieve an effective stirring. This continuous agitation ensures a uniform temperature distribution throughout the solution, minimizing temperature gradients and allowing the system to reach thermal equilibrium more quickly.

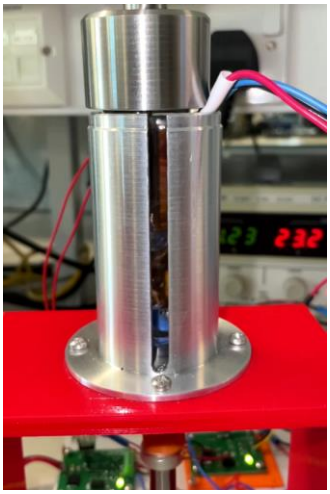


Figure 28. Solution agitation using magnetic stirrer

3.4 Headspace extraction

The primary component of the headspace extraction unit, the vacuum pump operated via the webserver enabled GUI, draws air from the laboratory environment and directs it through the piping connected the adjacent nozzle however the systems performance was very suboptimal, with a clear lack of airflow being generated from the vacuum pump. This indicates an issue in either the pump selection and or the piping not making appropriate fitting to the corresponding system components, additionally, despite correct assembly in regard to the air flow rate sensor illustrated in Appendix 5, the sensors output returns 0V, indicating no air flow in the system, suggesting that the sensor may have been damaged during the prototyping phase.

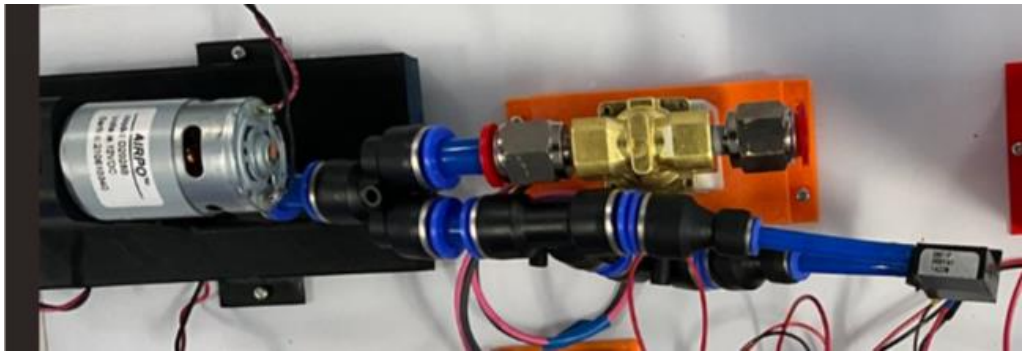


Figure 29. Vacuum pump, solenoid valve and flow rate sensor

4. Conclusion

The initial prototype of the headspace sampling device successfully executes all core functionalities and supports remote operation of system peripherals via a web-based graphical user interface hosted on the onboard ESP32 microcontroller; the device meets the majority of predefined design specification and incorporates sever additional enhancements.

4.1 Further Development

Several areas remain for future development to enhance overall system robustness, performance and reliability, one key limitation identified is inadequate noise suppression, which can be mitigated by integrating RC snubbers circuitry across inductive components such as the solenoid valve and vacuum pump to suppress voltage transients and reduce electromagnetic interference (EMI), other extreme solutions involved using ferrite beads across the vacuum pump leads to reduce electrical noise.

To address the presence of data outliers observed in the GUI output, Figure 24 implementing a moving average filter over the most recent five to ten data points would help smooth input signals and minimize the impact of transient anomalies. This filtering technique would improve the data stability and enhance the readability of system behavior over time.

The current system does not evaluate the effect of the magnetic stirrer on its ability to expedite equilibrium, of the sample matrix, to quantify this additional temperature measurement should be collected during stirrer operation, allowing empirical data that proves the stirrer facilitates headspace distribution could then be obtained.

To prevent condensation within the transfer line, the system can utilize the Kapton Heater shown in Figure 30, which maintains the line at a controlled temperature, this heater can be integrated into the systems software architecture to dynamically regulate the heating. Originally the design intended to incorporate a humidity sensor within the transfer oath to detect for moisture accumulation ad enable feedback control with the Kapton heater, however the sensor was not integrated due to project timelines.

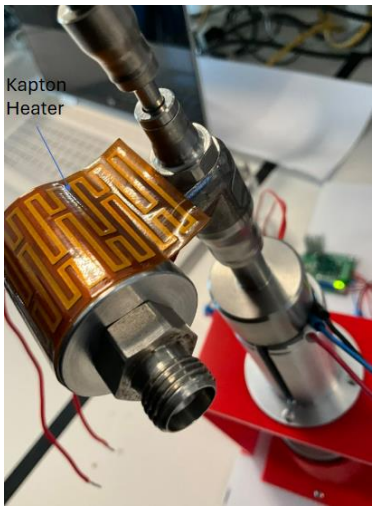


Figure 30. Transfer line incorporating Kapton Heater to prevent condensation

A significant hardware limitation is the lack of a dedicated and properly fitted connection between the sampling system and the mass spectrometer inlet, a custom adaptor or connector could be designed to ensure a secure connection. Furthermore, replacing certain polymer-based components with inert materials such as stainless steel or brass would ensure accurate mass spectrometer results.

From a user interface perspective, several improvements are necessary, the temperature logging feature in the GUI currently lacks a timestamp or time axis, making it difficult to correlate temperature readings with sampling events. Additionally, sever GUI elements omit essential units e.g., the flow rate is represented as a percentage rather in standard units like mL/min.

Communication architecture also offers room for improvement. Implements the Message Quering Telemetry Transport (MQTT) protocl would enable more efficient and scalable data exchange between the ESP32 micro controller and the. MQTT is a lightweight, publisher, subscriber messaging protocol ideal for IoT systems.

The current approach for determining the PID control coefficients relies on manual trial and error tuning which is both time consuming and unlikely to yield optimal performance, to improve the tuning efficiency, established methods such as the Ziegler Nicholas tuning

method can be employed. The technique involves first configuring the PID controller in proportional control only and gradually increasing the proportional gain until the system output exhibits oscillations, the gain at which these oscillations occur is referred as the ultimate gain Ku, and the oscillation period is called the ultimate period Pu, these parameters are then used to compute the optimal PID coefficients using the standard Ziegler Nicholas tuning rules [20]. Alternatively control system design tools such as MATLABs PID Tuner offers automatic tuning capabilities.

Lastly, the system needs adequate enclosure protection, a more robust enclosure design is required to shield circuit components from environmental elements, as well as safeguard the user from high temperature elements such as the Kapton heater.

4.2 Reflection and Learning

Throughout the course of this project, I have significantly advanced my skills as an electronics engineer. This work has strengthened my proficiency in custom PCB design and deepened my ability to identify, troubleshoot, and resolve hardware and firmware level issues in real time. I also gained practical experience in implementing control algorithms within an embedded electronics context, reinforcing theoretical concepts covered in coursework through hands on application.

A key area of growth involved learning computer aided design (CAD) for mechanical components and enclosures. Although I began the project with limited experience in CAD, I acquired the necessary skills through the duration of the project and saw steady improvement as the project progressed, providing a better insight into mechanical integration alongside incorporating electrical systems.

In addition, I became familiar with version control practices using Git, which allowed me to manage code changes systematically, and track development milestones. Overall, the project provided me a comprehensive platform for integrating electronic design, software development and system level engineering skills in a cohesive and practical setting.

5. References

[1] Research Gate, “An Introduction to Headspace Sampling in Gas Chromatography.” [Online]. Available: https://www.researchgate.net/profile/Mohamed-Dadamouny/post/Is_anybody_in_Egypt_using_the_headspace_technique/attachment/59d61dd779197b807797aabd/AS%3A273665973784576%401442258460620/download/GDE_Intro_to_Headspace.pdf

[2] “Headspace Techniques for Volatile Sampling,” ScienceDirect. [Online] Available: https://www.sciencedirect.com/science/article/abs/pii/S0166526X17300090
[3]“Lonestar Portable Analyzer,” MRIGlobal CBRNE Tech Index. [Online] Available: https://www.cbrnetechindex.com/p/3687/Owlstone-Inc/Lonestar-Portable-Analyzer
[4] Accessed: [Online]. Available: https://www.agilent.com/cs/library/applications/an-flavor-compounds-beer-8697-headspace-8890-gc-5994-4292en-agilent.pdf
[5]Fanell, “KHLVA-2020/10 .” [Online]. Available: https://uk.farnell.com/omega/khlva-202-10/flexible-heater-polyimide-40w/dp/3274625
[6] Accessed: [Online]. Available: https://www.mouser.co.uk/datasheet/2/389/stp55nf06l-1851309.pdf
[7] ACONNOR1, “an-1084p1.p65”. [Online]. Available: https://www.infineon.com/dgdl/Infineon-power_mosfet_basics-Article-v01_00-EN.pdf?fileId=8ac78c8c8d2fe47b018e625961741a0e#:~:text=Turn%2Don%20delay%2C%20td
[8]T. I. [SLUSDT9,B] Incorporated, “UCC23513-Q1 4-A Source, 5-A Sink, 5.7-kVRMS Reinforced, Opto-Compatible, Single-Channel Isolated Gate Driver datasheet (Rev. B)”. [Online]. Available: https://www.ti.com/lit/ds/symlink/ucc23513-q1.pdf?ts=1746037346108&ref_url=https%253A%252F%252Fgoogle.com
[9] L. E. Staff, “ESP32 Basics: Generating a PWM Signal on the ESP32,” *Last Minute Engineers*, Oct. 21, 2023. [Online]. Available: https://lastminuteengineers.com/esp32-pwm-tutorial/
[10]“Analog to Digital Converter (ADC) - ESP32 - — ESP-IDF Programming Guide v4.4 documentation.” . [Online]. Available: https://docs.espressif.com/projects/esp-idf/en/v4.4/esp32/api-reference/peripherals/adc.html
[11]“Thermocouple Accuracies,” TC Direct. [Online]. Available: https://www.tcdirect.co.uk/thermocouple-sensors/thermocouple-accuracies.aspx
[12]“Thermocouple accuracies - Thermocouple Accuracy,” Thermocouple Accuracy Comparison Chart. Accessed: Apr. 30, 2025. [Online]. Available: https://www.thermocoupleinfo.com/thermocouple-accuracies.htm
[13]“Pt1000, 2.0x5.0, Class B, PTFD102B1G0,” TE Connectivity. [Online]. Available: https://www.te.com/en/product-NB-PTCO-126.html
[14] O. Ali, “A Guide to Semiconductor Temperature Sensors,” AZoSensors. [Online]. Available: https://www.azosensors.com/article.aspx?ArticleID=2852
[15] T. I. [SNIS159,H] Incorporated, “LM35 Precision Centigrade Temperature Sensors datasheet (Rev. H)”. [Online]. Available: https://www.ti.com/lit/ds/symlink/lm35.pdf?ts=1745992953932&ref_url=https%253A%252F%252Fwww.mouser.cn%252F

[16]T. Lupinski, M. Ludwig, S. Fraden, and N. Tompkins, “An Arduino-based constant pressure fluid pump,” *The European Physical Journal E*, vol. 44, no. 2, pp. 1–7, Mar. 2021. [Online]. Available: https://link.springer.com/article/10.1140/epje/s10189-020-00002-9
[17]“D2028B,” RS. [Online]. Available: https://uk.rs-online.com/web/p/arduino-compatible-boards-kits/2857315
[18] D6F-P0001A1 Datasheet (PDF). [Online]. Available: https://www.mouser.co.uk/datasheet/2/307/en_d6f_series-1128136.pdf
[19] iMac1, “VX21-22-23-B_cat_en.indd”. Available: https://docs.rs-online.com/b218/0900766b815dd865.pdf
[20]ELEC303, Digital Control and Optimization. [Online]. Available: https://canvas.liverpool.ac.uk/courses/76964
[21] [Online]. Available: https://docs.rs-online.com/4066/0900766b8151e408.pdf
[22] O. Semiconductor, “MC7800 - Voltage Regulators – Positive”. [Online]. Available: https://www.onsemi.com/download/data-sheet/pdf/mc7800-d.pdf

6. Appendices

Appendix 1: Component Specification and Application

LED specification

Specification	**Value**
Forward voltage/V	2.5
Forward current/mA	25

Table 4. LED Specification

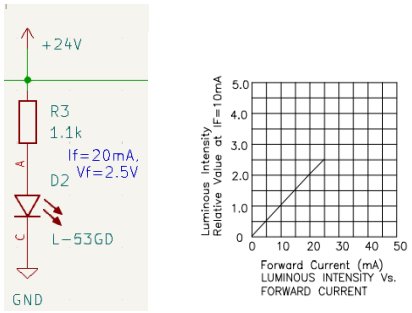


Figure 31. LED schematic & forward current vs Luminous intensity [21]

Series resistance of 1.1Ω is limits current through the LED to 20mA, with 20mA of current, the LED emits sufficient brightness.

Optically Couple Gate Driver

Gate driver UCC23513QDWYRQ1 [8], optically isolates micro-controller side to high voltage side, and sources current to MOSFET.

Application:

0 –5V PWM signal is connected to the Anode of the e-diode, Cathode to ground, total resistance is 260Ω, allowing a current of 10mA to flow through the LED.

E-diode characteristics:

Specification	**Value**
Vf (Forward Voltage)/V	Typ(2.1), Max (2.4)
If (Forward Current)/mA	Typ(10), Max (16)

Table 5. Gate Driver E-Diode Characteristics

Application summary:

Specification	**Value**
Vout(Output Voltage)/V	15
Output Current Max/A	1
Propagation delay/ns,	105
Input current/mA	10
Quiescent current/mA	Typ(1.2), Max (2.2)
Under voltage lockout (UVLO)/V	Typ(12.5), Max (13.5)
High Level Peak Output Current (IOH)/A	Min (3) ,Typ(4.5)
Low Level Peak Output Current	Min (3.5), Max (5.3)

Table 6. Gate Driver Application Summary

Quiescent current: small current drawn by a device or component when stable (current consumed for internal operation).

Input stage:

When IF exceeds the threshed current a frequency signal is transmitted across the isolation barrier, if the anode voltage drops below VF_HL (0.9V), or reversed bias the gate output is driven low, the reverse breakdown voltage of the diode is >15V

Interlock architecture such that the designer can select 3.3V, 5V or up to 12V PWM signal source to drive the UCC23513-Q1, varied use appropriate input resistor.

Linear Drop Out Regulator (LDO)

LDO MC7815CDTRKG [22] provides +15V to the circuit, the maximum current source from the device is 1A.

Application: +24V sink to the LDO, with a quiescent current of 6mA, maximum, providing +15V output voltage with 1A maximum current, with an input of +24V and an output of +15V the LDO will drop +9V(9W), which will be converted to heat.

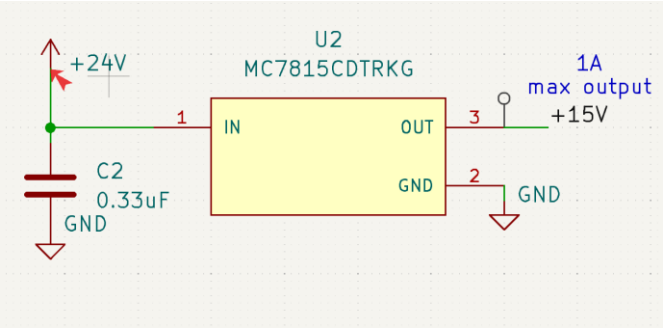


Figure 32. LDO MC7815CDTRKG

Application guideline recommends 0.33μF bypass ceramic capacitor at input.

Application summary:

Specification	**Value**
Vout(Output Voltage)/V	15
Iout(Output Current)/A	1
current/mA	Typ(3.5), Max (6)
Dropout Voltage/V	9
Power Dissipation/Watts	9

Table 7. LDO Application Summary

Appendix 2: ESP32 Pinout

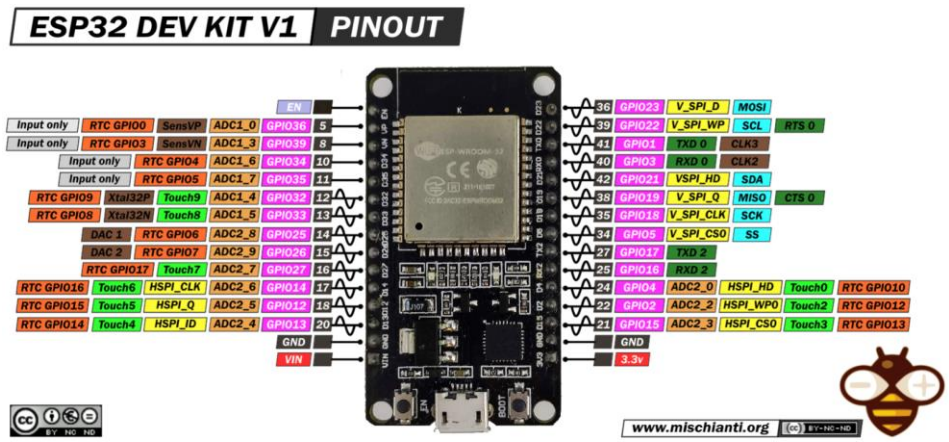


Figure 33. ESP32 Pinout [L]

Appendix 3 PCB Design Constraints and Fabrication

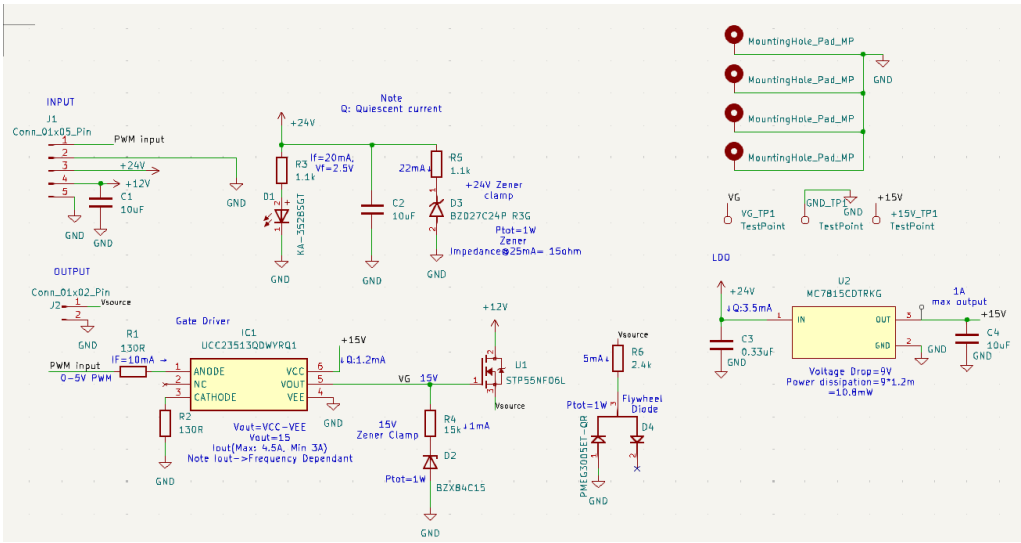


Figure 34. PWM Controller V2 Schematic Layout

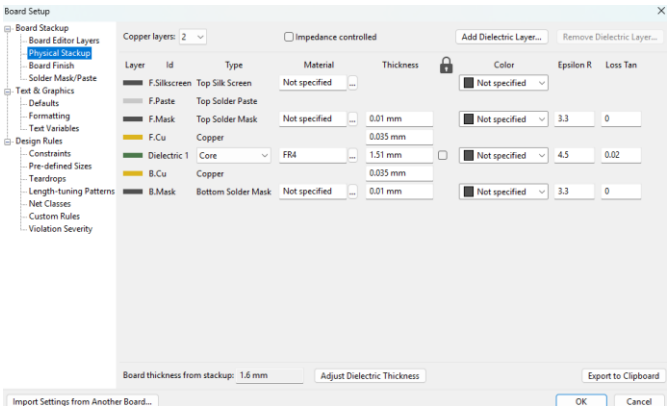


Figure 35. Physical Stack up

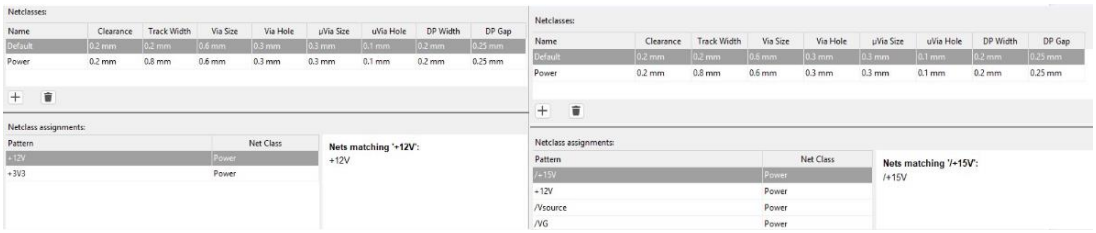


Figure 33. ESP32 Interpose Net classes and PWM controller Net Classes Respective

The physical stack up properties is consistent across the tree PCB variations, However, they differ in terms of net class arrangement. The ESP32 interposer features a distinct net class configuration, while the PWM controller V1 and V2 include more net assignments due to handling a greater number of signals, as shown in figure. Another key difference is the copper to edge clearance, the PWM controller V1 and V2 use a clearance of 0.5mm, compared to the 0.25mm on the ESP32 interposer. The increased clearance in the PWM controller versions helps reduce the risk of manufacturing defects, contributing to higher PCB yield and reliability.

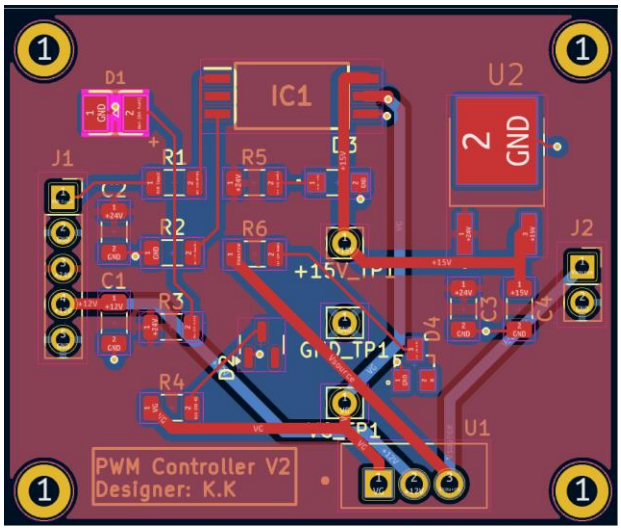


Figure 36. PWM controller V2 PCB layout

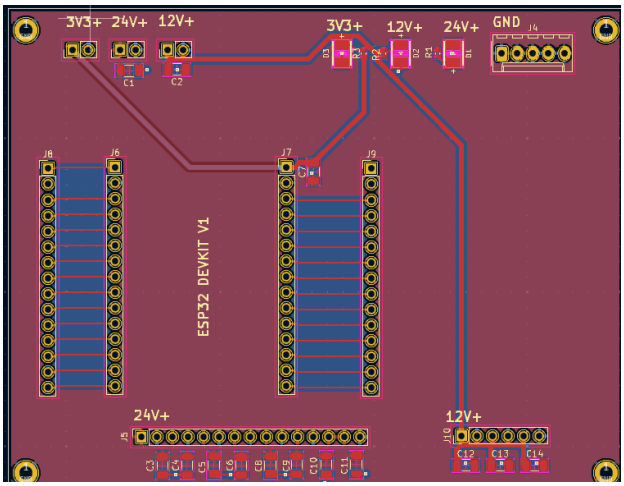


Figure 37. ESP32 Interpose PCB layout

In preparation for PCB manufacturing, all relevant fabrication documentation is generated, including the Gerber files, drill files, and Bill of Materials (BOM) only if component assembly is required. These files define the board's layer data, hole placements and the list of components with their specifications. Respectively. Aside from copper to edge clearance, other key design constraints such as minimum trace width, via diameter and solder mask expansion are carefully matched to manufacturer requirements, such as those outlined by PCBWAY, this ensures that the design is both manufacturable and compliant with industry standards.

Appendix 4: Vial Dimension

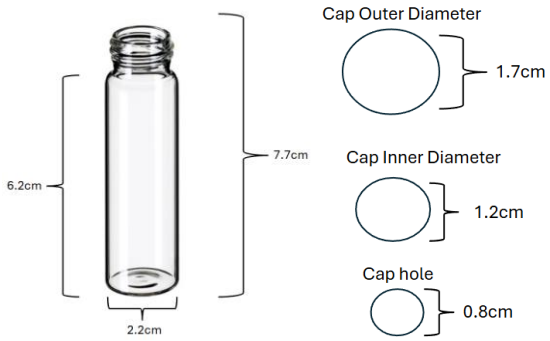


Figure 38. Vial & Cap dimensions

$$V = \pi r^2 h$$

Equation 17

$$V = \pi r \left(\frac{2.2}{2}\right)^2 \cdot 6.2$$

Equation 18

$$V = 23.57cm^3 \approx 0.02357Litres$$

Equation 19

$$Area = Circumfrence \times Length$$

Equation 20

$$Area \approx 49.76cm^2$$

Equation 21

Appendix 5: Air Flow sensor application

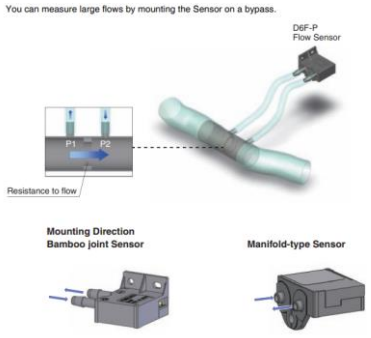


Figure 39. D6F-P0001A1 Flow Sensor Application [L]

Appendix 6: PWM Controller V1 Application Modifications

Several modifications were made to successfully connect the PWM Controller V1 to both the Kapton heating element and the power source. Initially, the input connector J1 did not provide the required 24V supply; therefore, power leads were instead soldered directly to the +24V pad of component D4, as shown in Figure 40, the footprint of connector J2 was found to be too small to accommodate both leads from the Kapton Polyamide heater without risking a short circuit. To resolve this, one of the Kapton leads was rerouted and connected to the +24V pad of component D1, ensuring proper electrical isolation and maintaining system safety. After board testing, it was observed that none of the mounting holes on the PCB had an effective ground connection, although these connections were present in the PCB schematic design, they were missing on the manufactured boards, to establish proper grounding, connections can be manually created by linking the mounting holes to a nearby ground pad available on any of the existing component footprints.

Figure 40. PWM Controller V1 PCB Modifications

Appendix 7: PWM Controller V2 Application Modifications

Three key modifications were implemented in the PWM Controller V2 to enhance its performance and reliability. First, a wire connection was added from the anode of diode D4 to the 2.4kΩ resistor R6 to correct for an earlier ordering mistake involving the wrong resistor package. Second, the PWM controller shown in figure is utilized to adjust the duty cycle of the magnetic stirrer, ensuring precise control of the stirring speed, to filter the voltage ringing at Vsource due to the inductive load a 0.1µF capacitor is installed across the pads of connector J2, which reduces the electrical noise experienced by the ESP32 development board. Thirdly due to the rapid switching behavior of the circuit, a heatsink was attached to the MOSFET to improve thermal management and maintain operational stability.

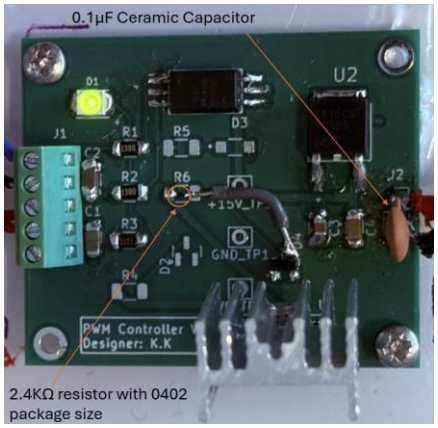


Figure 41. PWM Controller V2 PCB Modifications

Appendix 8: PWM Controller V1 Application Modifications

Figure showcases the 0.1µF bypass capacitor installation directly at the vacuum pumps, 12V and ground pin, to reduce the voltage ringing.

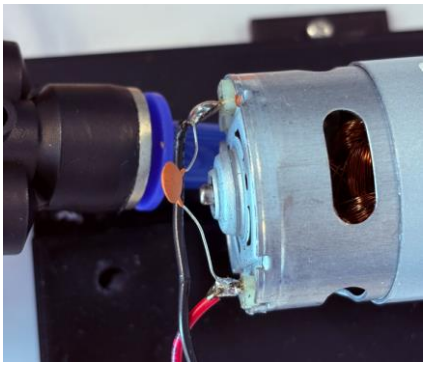


Figure 40. Vacuum Pump 0.1µF bypass capacitor installation

Appendix 9: Buck Converter

Figure 42 is the generic buck converter used to provide the different voltage potentials of the circuit, providing a 24V input, a variable output voltage is achieved by twisting the clockwise boost, illustrated in the diagram clockwise, until the desired voltage is reached.

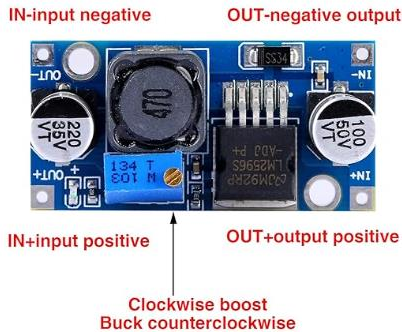


Figure 42. Generic buck converter [L]

Appendix 10: 3D Housings and Attachments

Vial chamber

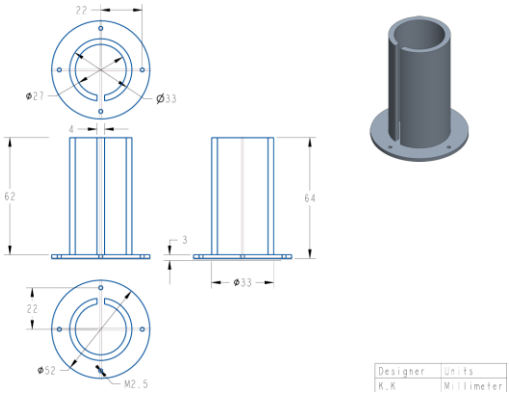


Figure 43. Vial enclosure drawing

Note: mounting holes on the aluminum fabrication are M3 holes rather than the drawing specified M2.5, this change was made because the M2.5 holes requiring nonstandard dill bits complicating the manufacturing process.

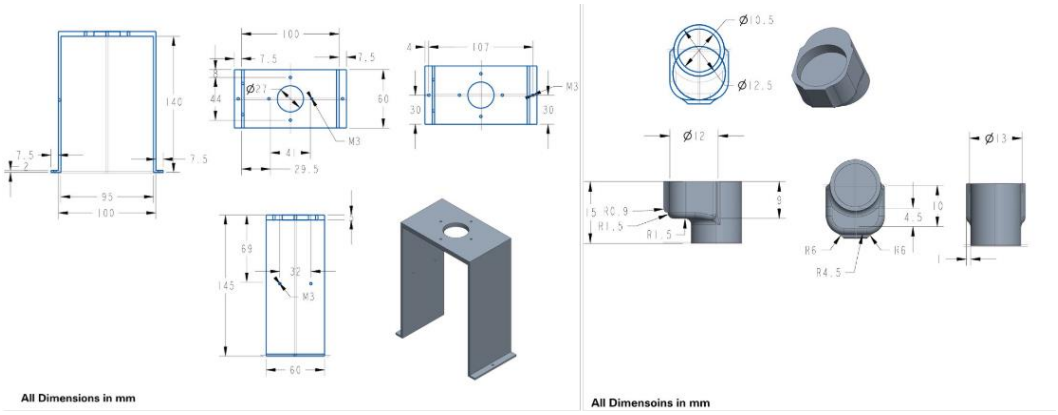


Figure 44. Vial stands and shaft mount drawings

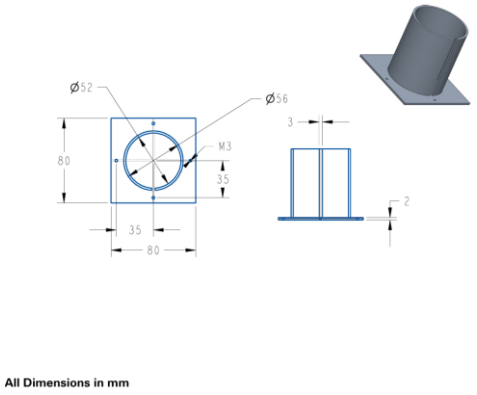


Figure 45. Motor Holder drawing

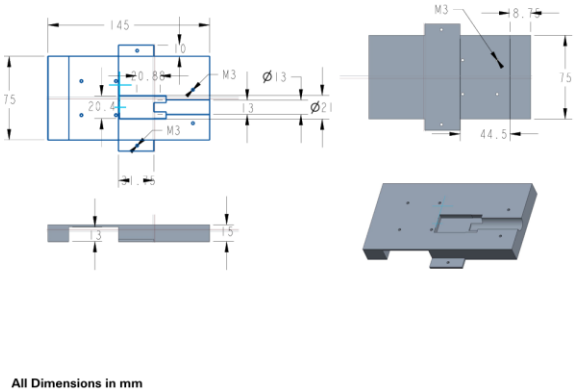


Figure 46. Motor Holder drawing

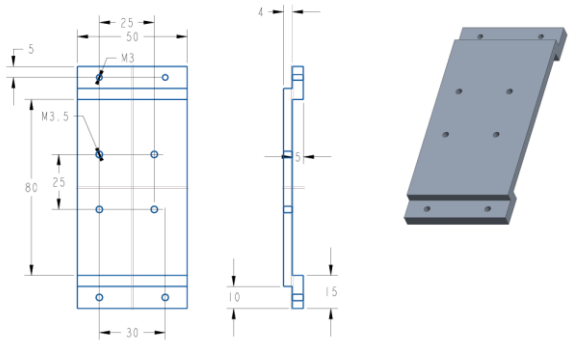


Figure 47. Solenoid Valve holder drawing

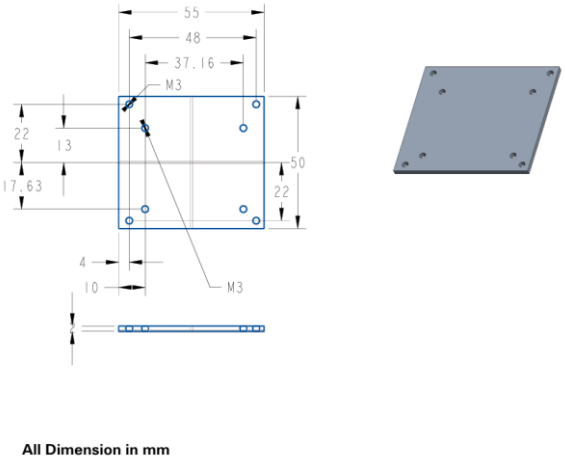


Figure 48. PCB holder V1 drawing

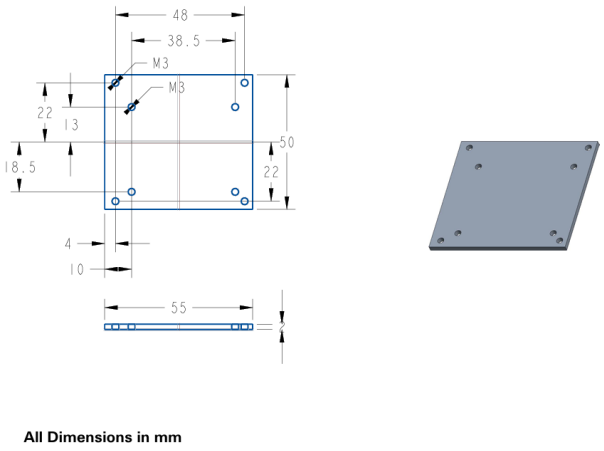


Figure 49. PCB holder V2 drawing

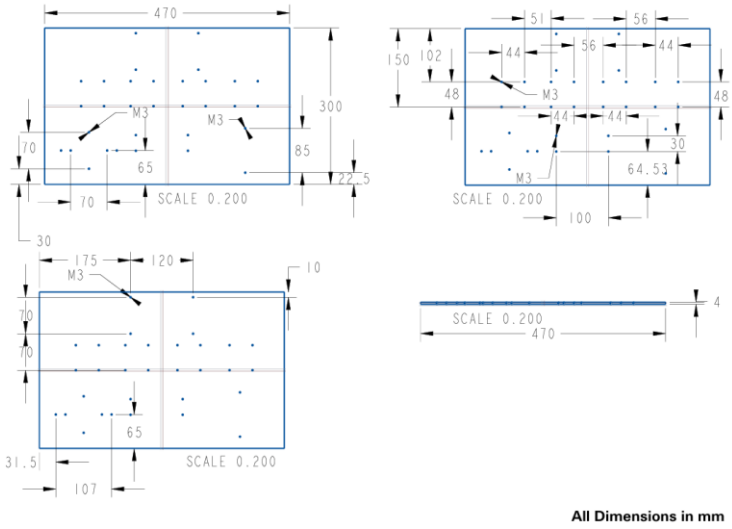


Figure 50. Base footprint drawing

Appendix 11: Little File System

A native feature supported by the ESP32 microcontroller is the Little File System (LittleFS), which allows the device to store and host front end files (HTML, CSS and JavaScript), directly from its internal flash memory. This enables the ESP32 to sever as a complete,

self-contained web interface without needing an external sever. The hosted files define the layout, styling and interactive behavior of the user interface.

Appendix 12: Gantt Chart

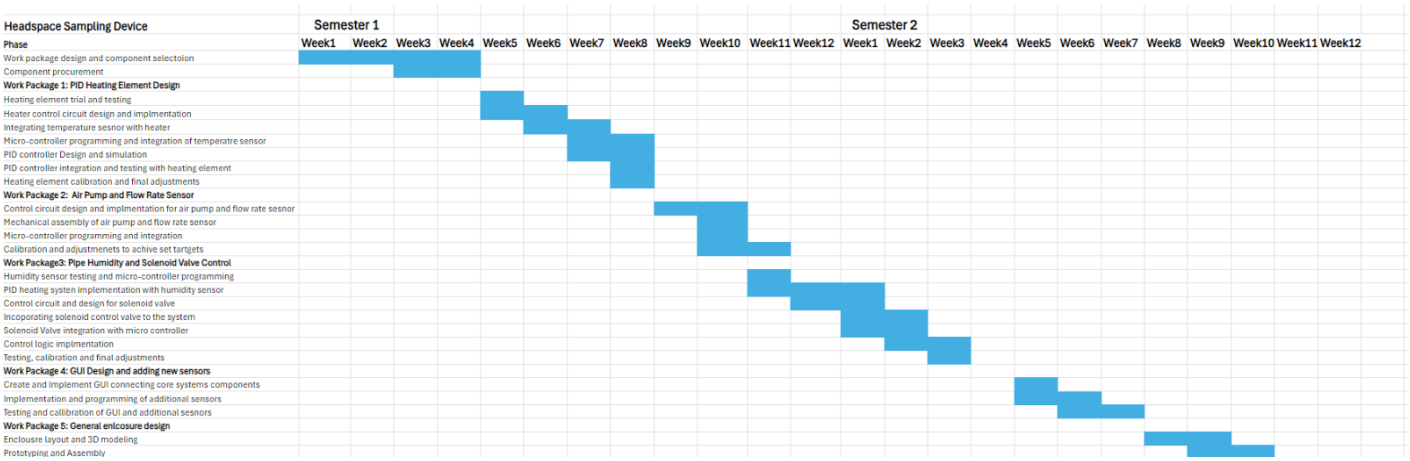


Figure 50. Gantt Chart

A Revised Gantt chart was developed at the beginning of Semester 2 to schedule immediate objectives required for the successful completion of the project.

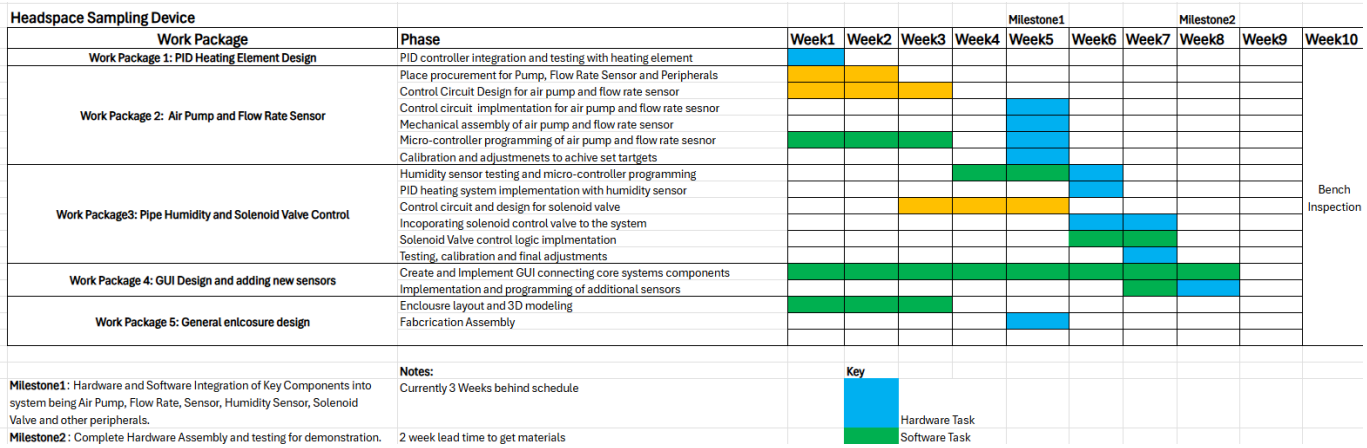


Figure 51. Gantt Chart Second Semester two only

Appendix 13: Original Milestones

Specific	Design a PID controlled heating element to uniformly alter the temperature of two

	thermal loads, the sample and the headspace distribution pipe.
Measurable	The PID controlled heating element should achieve a maximum of 120°C, with an accuracy of 1°C, additionally the heating element needs to draw less than 20w of power and achieve a temperature rise and fall time of 20 seconds. The efficiency of the heater can be determined after attaching a thermal load and using thermistor or any other temperature sensor to periodically record the temperature of the thermal mass.
Achievable	Using MATLAB, then selected heating element and an appropriate power source, the PID controlled heating element can be achieved.
Relevant	The heating element increases the efficiency at which the sample moves from the sample phase to the gas phase(headspace), making it the most vital part of the system.
Time Bound	This can be achieved by week 8 considering lead times and learning PID control, see Figure 5.

Table 8. Work package 1: Heating element for sample and headspace distribution system.

.

Specific	Assemble air pump and flow meter and integrate with micro-controller to achieve variable flow rate.
Measurable	The flow rate of headspace should have a range of 1 to 500ml/min, with an accuracy of 0.5ml/min, with the flow rate being determined by the user. The capability of the air pump will be determined by the data acquired from the flow rate meter.
Achievable	This would require a feedback system where the air pump's flow rate changes based on the data acquired from the flow meter in relation to what the user set.

Relevant	The flow rate determines the effects of the time taken for the headspace to migrate to the MS for analysis.
Time bound	This is second priority and will be completed by towards the start of week 12.

Table 9. Work package 2: Air pump and flowrate control system

Specific	Implement previous PID heating element as well as a relative humidity sensor to reduce condensation. Implement a solenoid valve.
Measurable	The PID system will function similar to that use to heat the sample vial, with a range of ambient to 120°C an accuracy of 1°C and a rise and fall time of 20 accounting for a thermal load. This system will additionally make use of a relative humidity sensor where when the humidity of the pipe reaches a relative humidity in the range of 60 to 100% the heating element would go on. The solenoid valve attached to the system should have an action time of within 1 minute.
Achievable	The PID heater element is re-used, lessening the workload, the only attachment is a humidity sensor to the feedback system, lastly integrating the solenoid valve requires the least amount of effort compared to the other system components.
Relevant	The heating element and humidity sensor for the pipe is not integral to the operation of the system and does not contribute to the operation of the sampling, however the solenoid valve is necessary to the sampling process is not optional when assembling the system
Time bound	This will be completed around week 3 of semester 2.

Table 10. Work package 3: Headspace distribution system humidity and solenoid valve control.

Specific	Implement a GUI interface to control key parts of the system as well as add additional sensors such as a carbon dioxide and or oxygen sensor to the transfer pipe.
Measurable	The GUI interface should combine all the key parameters of the system including temperature of the sample and pipe, relative humidity, system flow rate and solenoid valve control, with an update time of 30 seconds to ensure responsiveness. The effectiveness of the additional sensors will be measured by their performance to accurately collect their respective topics.
Achievable	Time will be needed to learn how to create a user interface in MATLAB, however the other components only require basic hardware and software skills.
Relevant	The Micro-controller GUI control is an integral part of the system as it completes the automation aims of the project. The addition of the two sensors is not essential in the operation of the device and should only be done given time towards the end of the project.
Time bound	This will be completed around week 8 of semester 2.

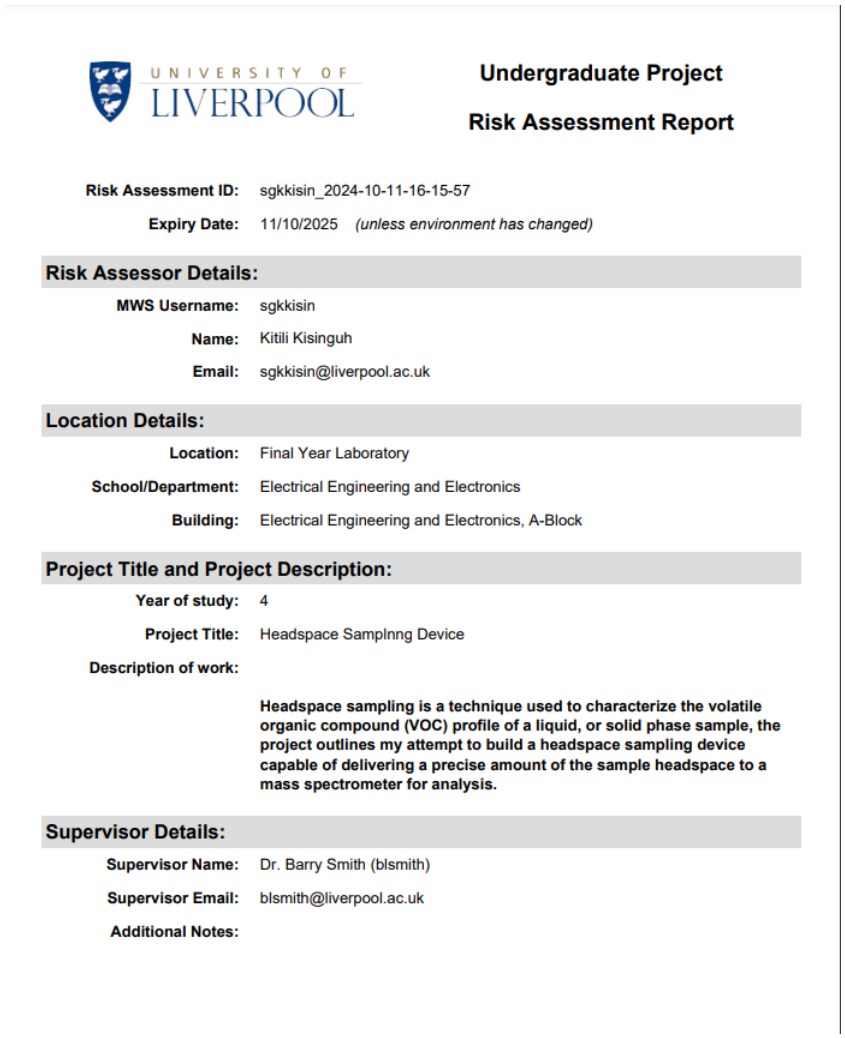
Table 11. Work package 4: GUI design and implementation of additional transfer line sensors.

Specific	Design and print an enclosure for the PID heating element as well as an enclosure to house all elements of the design.
Measurable	The enclosure should be able to house the heating elements as well as the sample vial of dimensions as well as house the various elements involved in making the device function, this includes pipes, heating elements, solenoid valve, flow meter, pump and power management systems.

Achievable	Using Creo or a similar computer aided design (CAD) software, the enclosure could be designed and fabricated.
Relevant	The enclosure for the heating elements as well as the housing for all system components creates an organized environment, making troubleshooting and maintenance easier as well as provides added safety shielding dangerous components from the user and or bystanders present. It is noted that this is not relevant to the functionality of the system as a whole and is placed as the lowest priority.
Time Bound	Being the lowest priority means this should be completed around week 10 of semester 2, as seen in figure 5.

Table 12. Work package 5: General enclosure design.

Appendix 14: Risk Assessment Form



UNIVERSITY OF LIVERPOOL

Undergraduate Project

Risk Assessment Report

Risk Assessment ID: sgkkisin_2024-10-11-16-15-57

Expiry Date: 11/10/2025 *(unless environment has changed)*

Risk Assessor Details:

MWS Username: sgkkisin

Name: Kitili Kisinguh

Email: sgkkisin@liverpool.ac.uk

Location Details:

Location: Final Year Laboratory

School/Department: Electrical Engineering and Electronics

Building: Electrical Engineering and Electronics, A-Block

Project Title and Project Description:

Year of study: 4

Project Title: Headspace Samplnng Device

Description of work:

Headspace sampling is a technique used to characterize the volatile organic compound (VOC) profile of a liquid, or solid phase sample, the project outlines my attempt to build a headspace sampling device capable of delivering a precise amount of the sample headspace to a mass spectrometer for analysis.

Supervisor Details:

Supervisor Name: Dr. Barry Smith (blsmith)

Supervisor Email: blsmith@liverpool.ac.uk

Additional Notes:

Figure 52. Risk Assessment form page 1.

Category:

Category 1 – Projects based on specialist equipment:

Projects requiring equipment available in the electronics laboratories (such as power supplies, multimeters, oscilloscopes, etc.) or any other specialist equipment that requires specific health and safety considerations (such as drones, etc.) that students would not normally be allowed to take home.

Voltage and Current:

Any power supplies that can generate current and voltages of **>10mA AND >20V** respectively, can be regarded as potentially extremely hazardous:

Voltage: 24

Current: 5

Identified Hazards and Current Controls:

Likelihood(L) x Consequence(C) = Risk(R)

Identified Hazard	Who can be Harmed	Current Controls	L	C	R
Moving solenoid in the solenoid can cause risks of pinching or crushing.	Operator	Solenoids enclosed in solenoid valve housing frame. Moving parts of the solenoid are not strong enough to crush an individual's limbs.	1	2	2
Risk of touching exposed moving parts of the Pump	Operator	Moving parts are enclosed in the pump frame Moving parts of the pump are not strong enough to crush the limbs of an individual.	1	2	2
Sample vial breakage from significant vapor pressure.	Operator	A sample vial with appropriate pressure specification is used. A plastic screen can be further employed to shield both the operator and other bystanders.	2	2	4
Overheating of electrical components.	Operator	Operating electrical components as specified by respective data sheets.	1	2	2
Touching exposed heating elements.	Operator and bystanders	Exposed parts of heating elements to the user will be insulated. Warning indicators such as labels, indicating hot surfaces and indicator LEDs will be used to warn users of any hot areas The device can be placed in a designated marked area as a warning not to touch.	2	2	4
Careless placement of the device near the ledge causing injury to the foot.	Operator and bystanders	The device will be placed at a reasonable length from the ledge of any table surface.	2	2	4
Exposed high voltage/current elements, leading to shock.	Operator and bystanders	Electrical components used, operating under high voltage and currents, are encased in above adequate insulation. Additionally, proper warning signs such as keep-off areas and labels will be used to warn of high voltage/current areas.	2	3	6

Figure 53. Risk Assessment form page 1.

Appendix 15: ESP32Webserver.INO

#include <Arduino.h> #include <WiFi.h> #include <AsyncTCP.h> #include <ESPAsyncWebServer.h> #include "LittleFS.h" #include <Arduino_JSON.h>

```
// LM35 Pin (ADC1) const int lm35_pin = 34; const int pwm_pin = 4; const int pump_pin =
13; const int solenoid_valve = 14; const int motor_pin = 27; const int flow_sensor = 32;

// Temperature sensor and PID control variables float temp_val = 0; double dt, last_time;
double integral = 0, previous = 0; double kp, ki, kd; double temp_set = 25; // Temperature
setpoint in °C double dutycycle = 0; double airflow = 0;

// Anti-windup limit for the integral term const double INTEGRAL_MAX = 100.0;

#define ADC_VREF_mV 3300.0 // ADC reference voltage in millivolts #define
ADC_RESOLUTION 4095.0

// Access Point credentials const char *ssid = "ESP32"; const char *password = "123";

// Create AsyncWebServer object on port 80 AsyncWebServer server(80);

// Create an Event Source on /events AsyncEventSource events("/events");

// JSON variable to hold sensor readings JSONVar readings;

// Timer variables; readings every 1 second unsigned long lastTime = 0; unsigned long
timerDelay = 1000;

// Get sensor readings and return a JSON string String getSensorReadings()
{ readings["sensor"] = readTemperature(); return JSON.stringify(readings); };

// Initialize LittleFS void initLittleFS() { if (!LittleFS.begin()) { Serial.println("An error has
occurred while mounting LittleFS"); } else { Serial.println("LittleFS mounted
successfully"); } };

// Initialize WiFi in Access Point mode void initWiFi() { WiFi.mode(WIFI_AP);
WiFi.softAP(ssid, password); Serial.print("AP IP address: "); Serial.println(WiFi.softAPIP()); }

double readTemperature() { int adcVal = analogRead(lm35_pin); float milliVolt = adcVal *
(ADC_VREF_mV / ADC_RESOLUTION); float tempC = milliVolt / 10.0; return tempC; };

double computePID(double error) { double proportional = error; integral += error * dt;

//Clamp to prevent windup if (integral > INTEGRAL_MAX) { integral = INTEGRAL_MAX; } else
if (integral < -INTEGRAL_MAX) { integral = -INTEGRAL_MAX; };
```

double derivative = (error - previous) / dt; previous = error; double outputVal = (kp * proportional) + (ki * integral) + (kd * derivative); return outputVal; };

void setup() {

// Serial port for debugging Serial.begin(115200); initWiFi(); initLittleFS();

analogWriteFrequency(pump_pin, 10000); analogWriteFrequency(motor_pin, 5000);

// Serve index.html from LittleFS at root URL server.on("/", HTTP_GET, [](AsyncWebServerRequest *request) { request->send(LittleFS, "/index.html", "text/html"); });

// Serve static files from LittleFS server.serveStatic("/", LittleFS, "/");

// Endpoint for sensor readings server.on("/readings", HTTP_GET, [](AsyncWebServerRequest *request) { String json = getSensorReadings(); request->send(200, "application/json", json); });

// Set up event source for clients events.onConnect([](AsyncEventSourceClient *client) { if (client->lastId()) { // Serial.printf("Client reconnected! Last message ID that it got is: %u\n", client->lastId()); } client->send("hello!", NULL, millis(), 10000); }); server.addHandler(&events);

// Start the web server server.begin();

// Initialize solenoid_valve (Pin 14) pinMode(solenoid_valve, OUTPUT); digitalWrite(solenoid_valve, LOW);

// PID constants and timer kp = 10; ki = 1; kd = 0.01; last_time = millis();

// Endpoint to adjust the temperature setpoint server.on("/setTemp", HTTP_GET, [](AsyncWebServerRequest *request) { if (request->hasParam("temp")) { String tempStr = request->getParam("temp")->value(); float newTempSet = tempStr.toFloat(); if (newTempSet >= 0 && newTempSet <= 150) { temp_set = newTempSet; Serial.print("New temperature setpoint: "); Serial.println(temp_set); } }

```
request->redirect("/?setpoint=" + String(temp_set, 0));
```

});

// Endpoint to adjust the flow rate server.on("/setFlow", HTTP_GET, [](AsyncWebServerRequest *request) { if (request->hasParam("airflow")) { String flowStr = request->getParam("airflow")->value(); float newairflow = flowStr.toFloat(); if (newairflow >= 0 && newairflow <= 100) { airflow = newairflow; Serial.print("New AirFlow setpoint: "); Serial.println(airflow); } }

```
request->redirect("/?flowrate=" + String(airflow, 0));
```

});

// Endpoint to update solenoid_valve state server.on("/updateSolenoid", HTTP_GET, [](AsyncWebServerRequest *request) { if (request->hasParam("state")) { String state = request->getParam("state")->value(); Serial.print("Updating solenoid_valve (Pin 14) state to: "); Serial.println(state); digitalWrite(solenoid_valve, state.toInt()); } request->send(200, "text/plain", "OK"); });

// Endpoint to return the current state of solenoid_valve server.on("/stateSolenoid", HTTP_GET, [](AsyncWebServerRequest *request) { String currentState = String(digitalRead(solenoid_valve)); request->send(200, "text/plain", currentState); });

// Endpoint to adjust the motor duty cycle server.on("/setMotor", HTTP_GET, [](AsyncWebServerRequest *request) { if (request->hasParam("motor")) { String motorStr = request->getParam("motor")->value(); float newdutycycle = motorStr.toFloat(); if (newdutycycle >= 0 && newdutycycle <= 100) { dutycycle = newdutycycle; Serial.print("NewDutyCycle: "); Serial.println(dutycycle); } }

```
request->redirect("/?dutycycle=" + String(dutycycle, 0));
```

}); }

void loop() { // Send sensor readings via events every 1 second if ((millis() - lastTime) > timerDelay) { events.send("ping", NULL, millis()); events.send(getSensorReadings().c_str(), "new_readings", millis()); lastTime = millis(); }; temp_val = readTemperature(); Serial.print("Temperature = "); Serial.print(temp_val); Serial.println(" °C");

double now = millis(); dt = (now - last_time) / 1000.0; last_time = now;

double error = temp_set - temp_val; double pidOutput = computePID(error); //Constrains PID from the range 0-255 pidOutput = constrain(pidOutput, 0, 255);

```
Serial.print("PID Output: "); Serial.println(pidOutput);

//Writing PID analogWrite(pwm_pin, pidOutput);

// Writing Vacumm Pump Air Flow int pumpout = map(airflow, 0, 100, 0, 255);
analogWrite(pump_pin, pumpout);

// Writing Motor Duty Cycle int motorout = map(dutycycle, 0, 100, 0, 255);
analogWrite(motor_pin, motorout);

Serial.print("Flow rate= "); Serial.print(flow_sensor); Serial.println("ml");

delay(100); // 100ms delay to give PID output enought time to take effect }
```

Appendix 16: index.html

```
<!DOCTYPE html>
<html>
 <head>
  <title>ESP32 Controller</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" type="text/css" href="style.css">
  <style>
   #graphCanvas {
    border: 1px solid #000;
    background: #fff;
    margin: 20px auto;
    display: block;
   }
   .controls {
    text-align: center;
```

```
    margin: 20px;
  }
  .controls p {
    font-size: 1.2rem;
  }
  .slider {
    width: 80%;
  }
</style>
<!-- Toggle switch styling -->
<style>
  .toggle-switch {
    position: relative;
    display: inline-block;
    width: 120px;
    height: 68px;
  }
  .toggle-switch input {
    display: none;
  }
  .toggle-slider {
    position: absolute;
    top: 0;
    left: 0;
    right: 0;
    bottom: 0;
```

```
    background-color: #ccc;
    border-radius: 34px;
   }
   .toggle-slider:before {
    position: absolute;
    content: "";
    height: 52px;
    width: 52px;
    left: 8px;
    bottom: 8px;
    background-color: #fff;
    transition: .4s;
    border-radius: 68px;
   }
   input:checked + .toggle-slider {
    background-color: #2196F3;
   }
   input:checked + .toggle-slider:before {
    transform: translateX(52px);
   }
  </style>
</head>
<body>
  <div class="topnav">
   <h1>ESP32 Controller</h1>
  </div>
```

```
  <div class="content">
   <div class="controls">
    <p>Setpoint: <span id="setpoint">25</span> &deg;C</p>
    <form id="tempForm" action="/setTemp" method="get">
     <input type="range" min="25" max="150" value="25" class="slider" id="tempSlider"
name="temp" oninput="document.getElementById('setpoint').innerText = this.value">
     <br><br>
     <input type="submit" value="Set Temperature">
    </form>
   </div>

   <h2>Temperature Graph</h2>
   <!-- Canvas element where the graph will be drawn -->
   <canvas id="graphCanvas" width="800" height="400"></canvas>
  </div>
  <script src="script.js"></script>
  <script>
   // Helper function to retrieve a query parameter's value
   function getQueryParam(param) {
    let params = new URLSearchParams(window.location.search);
    return params.get(param);
   }
   // On page load, update the setpoint display and slider if provided in the URL
   window.addEventListener('load', function() {
    let sp = getQueryParam('setpoint');
    if (sp) {
```

```
      document.getElementById('setpoint').innerText = sp;
      document.getElementById('tempSlider').value = sp;
    }
  });
</script>

<!-- Pump Control (Pin 13) -->
  <h3>Pump Control</h3>
  <div class="controls">
    <p>FlowRate: <span id="flowrate">0</span>%</p>
    <form id="flowForm" action="/setFlow" method="get">
      <input type="range" min="0" max="100" value="0" class="slider" id="FlowSlider" name="airflow" oninput="document.getElementById('flowrate').innerText = this.value">
      <br><br>
      <input type="submit" value="Set Flow Rate">
  </form>
  </div>
</script>

<!-- Toggle switch for Solenoid Valve Control (Pin 14) -->
<div class="controls">
  <h3>Solenoid Valve Control </h3>
  <label class="toggle-switch">
    <input type="checkbox" onchange="toggleSolenoid(this)" id="solenoidToggle">
    <span class="toggle-slider"></span>
  </label>
```

```
  <p>State: <span id="solenoidState">Off</span></p>
 </div>
 <script>
  function toggleSolenoid(element){
   var xhr = new XMLHttpRequest();
   if(element.checked){
    xhr.open("GET", "/updateSolenoid?state=1", true);
   } else {
    xhr.open("GET", "/updateSolenoid?state=0", true);
   }
   xhr.send();
  }

  // Poll for the current state of the Solenoid Valve every second
  setInterval(function () {
   var xhttp = new XMLHttpRequest();
   xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
     var solenoidStateText = (this.responseText.trim() == "1") ? "On" : "Off";
     document.getElementById("solenoidState").innerText = solenoidStateText;
     document.getElementById("solenoidToggle").checked = (this.responseText.trim() ==
"1");
    }
   };
   xhttp.open("GET", "/stateSolenoid", true);
   xhttp.send();
```

```
  }, 1000);
 </script>

 <h3>Motor Control</h3>
 <div class="controls">
  <p>DutyCycle: <span id="dutycycle">0</span>%</p>
  <!-- Added id "motorForm" -->
  <form id="motorForm" action="/setMotor" method="get">
   <input type="range" min="0" max="100" value="0" class="slider" id="MotorSlider"
name="motor" oninput="document.getElementById('dutycycle').innerText = this.value">
   <br><br>
   <input type="submit" value="Set Duty Cycle">
  </form>
 </div>

 <!--Intercepts form submissions and send AJAX requests -->
 <script>
  // Temperature form submit handler
  document.getElementById("tempForm").addEventListener("submit", function(e) {
   e.preventDefault();
   var tempVal = document.getElementById("tempSlider").value;
   var xhr = new XMLHttpRequest();
   xhr.open("GET", "/setTemp?temp=" + tempVal, true);
   xhr.send();
  });
```

```
    // Flow rate form submit handler
    document.getElementById("flowForm").addEventListener("submit", function(e) {
     e.preventDefault();
     var flowVal = document.getElementById("FlowSlider").value;
     var xhr = new XMLHttpRequest();
     xhr.open("GET", "/setFlow?airflow=" + flowVal, true);
     xhr.send();
    });

    // duty cycle form submit handler
    document.getElementById("motorForm").addEventListener("submit", function(e) {
    e.preventDefault();
    var dutyVal = document.getElementById("MotorSlider").value;
    var xhr = new XMLHttpRequest();
    xhr.open("GET", "/setMotor?motor=" + dutyVal, true);
    xhr.send();
    });
   </script>
  </body>
</html>
```

Appendix 17: script.Js

```
// Global variables to store data points and set maximum number of points
var dataPoints = [];
var maxPoints = 40;
```

```
// Get reference to the canvas and its context
var canvas = document.getElementById("graphCanvas");
var ctx = canvas.getContext("2d");

// Function to fetch sensor readings from the ESP32 /readings endpoint
function fetchData() {
  var xhr = new XMLHttpRequest();
  xhr.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      // Expecting a JSON object with the temperature reading under "sensor"
      var reading = JSON.parse(this.responseText);
      var temperature = parseFloat(reading.sensor);
      addDataPoint(temperature);
      drawGraph();
    }
  };
  xhr.open("GET", "/readings", true);
  xhr.send();
}

// Adds a new data point to the dataPoints array and removes the oldest if necessary
function addDataPoint(value) {
  dataPoints.push(value);
  if (dataPoints.length > maxPoints) {
    dataPoints.shift();
  }
```

```
}

// Draws a simple line graph on the canvas using the dataPoints array
function drawGraph() {
  // Clear the canvas
  ctx.clearRect(0, 0, canvas.width, canvas.height);

  // Draw axes
  // y-axis from (30, 10) to (30, canvas.height - 10)
  // x-axis from (30, canvas.height - 10) to (canvas.width - 10, canvas.height - 10)
  ctx.beginPath();
  ctx.moveTo(30, 10);
  ctx.lineTo(30, canvas.height - 10);
  ctx.lineTo(canvas.width - 10, canvas.height - 10);
  ctx.strokeStyle = "#000";
  ctx.stroke();

  //  compute min and max for scaling
  if (dataPoints.length > 0) {
    var maxVal = Math.max(...dataPoints);
    var minVal = Math.min(...dataPoints);
    if (maxVal === minVal) {
      maxVal = minVal + 1; // Avoid division by zero
    }

    // Draw temperature axis ticks and labels along the y-axis
```

```
  var tickCount = 5; // number of ticks
  ctx.font = "10px Arial";
  ctx.fillStyle = "#000";
  ctx.textAlign = "right";
  for (var i = 0; i <= tickCount; i++) {
   var tickValue = minVal + ((maxVal - minVal) * i / tickCount);
   // Map tickValue to y-coordinate (inverted axis)
   var tickY = canvas.height - 10 - ((tickValue - minVal) / (maxVal - minVal)) * (canvas.height
- 20);
   // Draw a small tick mark from x=25 to x=30
   ctx.beginPath();
   ctx.moveTo(25, tickY);
   ctx.lineTo(30, tickY);
   ctx.stroke();
   // Draw the label just to the left of the tick mark
   ctx.fillText(tickValue.toFixed(1), 25, tickY + 3);
  }

  // Draw the data as a line graph
  if (dataPoints.length > 1) {
   var step = (canvas.width - 40) / (maxPoints - 1);
   ctx.beginPath();
   for (var i = 0; i < dataPoints.length; i++) {
    var x = 30 + i * step;
    // Map data point to canvas y coordinate
```

```
      var y = canvas.height - 10 - ((dataPoints[i] - minVal) / (maxVal - minVal)) *
(canvas.height - 20);
      if (i === 0) {
        ctx.moveTo(x, y);
      } else {
        ctx.lineTo(x, y);
      }
    }
    ctx.strokeStyle = "#101D42";
    ctx.stroke();
  }
 }
}

// Fetch new data every 1 second
setInterval(fetchData, 1000);

// Optional: fetch initial reading immediately on load
fetchData();
```

Appendix 18 MATLAB Code; Response Analysis 3.3mL

```
function response_analysis()
% Input data
data = [22.47, 20.27, 21.25, 20.63, 19.54, 19.9, 19.54, 19.29, 21.37, 19.54, ...
17.58, 20.76, 20.39, 19.54, 21.49, 18.8, 21.86, 23.81, 23.81, 23.81, ...
25.15, 25.76, 27.59, 30.16, 32.97, 33.21, 33.58, 35.16, 35.9, 40.17, ...
39.68, 41.39, 43.71, 44.32, 47.13, 48.23, 52.01, 50.55, 54.33, 56.29, ...
57.88, 59.1, 59.83, 58.73, 64.47, 65.57, 64.59, 66.79, 69.96, 70.09, ...
73.63, 73.63, 74.97, 75.09, 76.19, 76.19, 76.31, 78.75, 78.27, 81.32, ...
81.07, 83.27, 81.93, 84, 84, 83.52, 83.76, 84.74, 84.25, 84.25, ...
```

```
83.88, 83.76, 84, 83.52, 83.76, 84.74, 84.25, 84.25, 83.88, 83.76, ...
86.08, 87.55, 90.72, 90.48, 87.42, 87.91, 88.16, 89.5, 88.64, 86.81, ...
88.52, 85.96, 86.69, 86.2, 86.69, 86.2, 87.55, 84.86, 85.71, 84.98, ...
83.27, 82.66, 82.66, 84, 82.3, 85.35, 84.13, 85.1, 84.74, 86.2, ...
86.69, 86.08, 84.62, 85.35, 86.08, 84.74, 86.32, 87.55, 87.79, 86.45, ...
87.55, 88.03, 91.45, 87.67, 89.87, 89.13, 87.91, 89.87, 89.13, 91.21, ...
90.6, 90.96, 90.72, 91.21, 88.03, 89.01, 91.7, 91.45, 90.6, 90.23, ...
90.11, 91.21, 87.91, 89.87, 91.33, 88.77, 90.23, 91.82, 90.48, 91.09, ...
88.77, 90.11, 88.64, 89.62, 89.99, 90.96, 90.72];
% Parameters
setpoint = 90;
settling_percentage = 2; % 2% settling band
sample_time = 5; % seconds per sample
% Create time vector
time = (0:length(data)-1) * sample_time;
temperature = data;
% Calculate response characteristics
[results] = calculate_response_parameters(time, temperature, setpoint,
settling_percentage);
% Display results (now includes steady-state error)
display_results(results, setpoint);
% Plot response
plot_response(time, temperature, setpoint, results, settling_percentage);
end
function [results] = calculate_response_parameters(time, temperature, setpoint,
settling_percentage)
% Initialize results structure
results = struct();
% Basic characteristics
results.min_value = min(temperature);
results.max_value = max(temperature);
% Calculate steady-state value (average of last 20 samples for better accuracy)
steady_state_window = min(20, length(temperature)); % Use last 20 samples or all if
<20
results.steady_state = mean(temperature(end-steady_state_window+1:end));
% Rise Time (10% to 90%)
v10 = results.min_value + 0.1 * (results.max_value - results.min_value);
v90 = results.min_value + 0.9 * (results.max_value - results.min_value);
t10_idx = find(temperature >= v10, 1, 'first');
t90_idx = find(temperature >= v90, 1, 'first');
results.rise_time = time(t90_idx) - time(t10_idx);
results.t10 = time(t10_idx);
results.t90 = time(t90_idx);
% Overshoot
results.overshoot_percentage = ((results.max_value - results.steady_state) / ...
results.steady_state) * 100;
% Steady-State Error (always positive)
```

```
results.steady_state_error = abs(setpoint - results.steady_state);
results.steady_state_error_percentage = (results.steady_state_error / setpoint) *
100;
% Settling Time
settling_band = settling_percentage / 100 * results.steady_state;
lower_bound = results.steady_state - settling_band;
upper_bound = results.steady_state + settling_band;
% Find last point outside the band
outside_band = (temperature < lower_bound) | (temperature > upper_bound);
settling_idx = find(outside_band, 1, 'last');
if ~isempty(settling_idx) && settling_idx < length(time)
results.settling_time = time(settling_idx + 1);
else
results.settling_time = 0;
end
% Store bounds for plotting
results.lower_bound = lower_bound;
results.upper_bound = upper_bound;
end
function display_results(results, setpoint)
fprintf('=== System Response Analysis ===\n');
fprintf('Rise Time (10%%-90%%): %.2f seconds\n', results.rise_time);
fprintf('Settling Time (2%% band): %.2f seconds\n', results.settling_time);
fprintf('Overshoot: %.2f%%\n', results.overshoot_percentage);
fprintf('Steady-State Value: %.2f°C\n', results.steady_state);
fprintf('Steady-State Error: %.2f°C (%.2f%% of setpoint)\n', ...
results.steady_state_error, results.steady_state_error_percentage);
fprintf('Setpoint: %.2f°C\n', setpoint);
end
function plot_response(time, temperature, setpoint, results, settling_percentage)
figure('Position', [100, 100, 800, 600]);
% Main plot
plot(time, temperature, 'b-', 'LineWidth', 1.5, 'DisplayName', 'Temperature');
hold on;
grid on;
% Setpoint line
plot([time(1), time(end)], [setpoint, setpoint], 'k--', 'LineWidth', 1.5,
'DisplayName', 'Setpoint');
% Steady-state line with error annotation
steady_state_line = plot([time(1), time(end)], [results.steady_state,
results.steady_state], ...
'g:', 'LineWidth', 1.5, 'DisplayName', sprintf('Steady-State (%.2f°C)',
results.steady_state));
% Annotate steady-state error
text(time(end), results.steady_state, ...
sprintf('Error: %.2f°C', results.steady_state_error), ...
'VerticalAlignment', 'bottom', 'HorizontalAlignment', 'right', ...
```

```
'Color', 'green', 'FontWeight', 'bold');
% Rise time indicators
plot([results.t10, results.t10], [min(temperature), max(temperature)], 'b--',
'LineWidth', 0.5);
plot([results.t90, results.t90], [min(temperature), max(temperature)], 'b--',
'LineWidth', 0.5);
patch([results.t10, results.t90, results.t90, results.t10], ...
[results.min_value+0.1*(results.max_value-results.min_value), ...
results.min_value+0.1*(results.max_value-results.min_value), ...
results.min_value+0.9*(results.max_value-results.min_value), ...
results.min_value+0.9*(results.max_value-results.min_value)], ...
'b', 'FaceAlpha', 0.1, 'EdgeColor', 'none', 'DisplayName', 'Rise Time');
% Settling time indicators
if results.settling_time > 0
settling_line = plot([results.settling_time, results.settling_time],
[min(temperature), max(temperature)], ...
'r--', 'LineWidth', 1.5, 'DisplayName', sprintf('Settling Time (%.1fs)',
results.settling_time));
end
% Settling band
settling_band = patch([time(1), time(end), time(end), time(1)], ...
[results.lower_bound, results.lower_bound, ...
results.upper_bound, results.upper_bound], ...
'r', 'FaceAlpha', 0.1, 'EdgeColor', 'none', 'DisplayName', sprintf('±%d%% Band',
settling_percentage));
% Labels and title
xlabel('Time (seconds)');
ylabel('Temperature (°C)');
title(sprintf('Temperature Response; 3.3ml\nRise Time: %.1fs, Settling Time: %.1fs,
Overshoot: %.1f%%, Steady-State Error: %.2f°C', ...
results.rise_time, results.settling_time, results.overshoot_percentage,
results.steady_state_error));
% Legend
legend('Location', 'best');
% Adjust axes
ylim([min(temperature)*0.9, max(temperature)*1.1]);
xlim([time(1), time(end)]);
hold off;
End
```

Appendix 19 MATLAB Code; Response Analysis 6mL

```
function response_analysis()
% Input data
data = [24.18, 23.32, 25.15, 23.93, 22.1, 23.57, 24.05, 23.08, 23.69, 24.42, ...
26.25, 25.89, 27.96, 27.96, 30.4, 32.72, 33.94, 32.97, 37.48, 35.65, ...
```

```
41.03, 42.12, 43.59, 45.79, 49.57, 49.57, 52.01, 53.6, 54.82, 58.73, ...
58.36, 60.93, 62.39, 62.39, 66.91, 66.79, 69.72, 69.11, 72.28, 74.36, ...
74.11, 76.31, 78.27, 78.88, 80.83, 82.3, 84, 86.45, 85.59, 87.06, ...
86.69, 87.91, 88.77, 89.13, 89.26, 90.35, 90.6, 89.87, 91.94, 91.45, ...
92.43, 91.58, 92.55, 91.94, 91.82, 90.96, 92.19, 92.92, 93.41, 93.04, ...
92.67, 92.43, 92.55, 92.67, 91.82, 93.16, 92.55, 91.82, 92.67, 91.33, ...
91.82, 90.96, 92.19, 91.7, 91.45, 93.41, 93.04, 84, 91.58, 91.82, ...
91.82, 92.19, 90.11, 90.72, 92.92, 92.06, 90.48, 91.94, 93.16, 91.94, ...
93.65, 93.77, 91.21, 92.67, 92.19, 91.45, 92.67, 94.63, 90.84, 90.48, ...
92.67, 91.82, 90.23, 91.94, 91.94, 91.58, 91.82, 91.09, 90.48, 89.99, ...
91.09, 90.84, 91.82, 91.82, 91.09, 90.23, 91.82, 92.43, 90.72, 91.82, ...
91.09, 90.23, 91.82, 92.43, 90.72, 93.04, 92.67, 92.19, 91.94, 93.89, ...
91.94, 91.82, 92.55, 89.5, 91.21, 89.99, 90.96, 92.19, 90.35, 91.82, ...
89.13, 89.01, 88.16, 90.48];
% Parameters
setpoint = 90;
settling_percentage = 2; % 2% settling band
sample_time = 5; % seconds per sample
% Create time vector
time = (0:length(data)-1) * sample_time;
temperature = data;
% Calculate response characteristics
[results] = calculate_response_parameters(time, temperature, setpoint,
settling_percentage);
% Display results (now includes steady-state error)
display_results(results, setpoint);
% Plot response
plot_response(time, temperature, setpoint, results, settling_percentage);
end
function [results] = calculate_response_parameters(time, temperature, setpoint,
settling_percentage)
% Initialize results structure
results = struct();
% Basic characteristics
results.min_value = min(temperature);
results.max_value = max(temperature);
% Calculate steady-state value (average of last 20 samples for better accuracy)
steady_state_window = min(20, length(temperature)); % Use last 20 samples or all if
<20
results.steady_state = mean(temperature(end-steady_state_window+1:end));
% Rise Time (10% to 90%)
v10 = results.min_value + 0.1 * (results.max_value - results.min_value);
v90 = results.min_value + 0.9 * (results.max_value - results.min_value);
t10_idx = find(temperature >= v10, 1, 'first');
t90_idx = find(temperature >= v90, 1, 'first');
results.rise_time = time(t90_idx) - time(t10_idx);
results.t10 = time(t10_idx);
```

```
results.t90 = time(t90_idx);
% Overshoot
results.overshoot_percentage = ((results.max_value - results.steady_state) / ...
results.steady_state) * 100;
% Steady-State Error (always positive)
results.steady_state_error = abs(setpoint - results.steady_state);
results.steady_state_error_percentage = (results.steady_state_error / setpoint) *
100;
% Settling Time
settling_band = settling_percentage / 100 * results.steady_state;
lower_bound = results.steady_state - settling_band;
upper_bound = results.steady_state + settling_band;
% Find last point outside the band
outside_band = (temperature < lower_bound) | (temperature > upper_bound);
settling_idx = find(outside_band, 1, 'last');
if ~isempty(settling_idx) && settling_idx < length(time)
results.settling_time = time(settling_idx + 1);
else
results.settling_time = 0;
end
% Store bounds for plotting
results.lower_bound = lower_bound;
results.upper_bound = upper_bound;
end
function display_results(results, setpoint)
fprintf('=== System Response Analysis ===\n');
fprintf('Rise Time (10%%-90%%): %.2f seconds\n', results.rise_time);
fprintf('Settling Time (2%% band): %.2f seconds\n', results.settling_time);
fprintf('Overshoot: %.2f%%\n', results.overshoot_percentage);
fprintf('Steady-State Value: %.2f°C\n', results.steady_state);
fprintf('Steady-State Error: %.2f°C (%.2f%% of setpoint)\n', ...
results.steady_state_error, results.steady_state_error_percentage);
fprintf('Setpoint: %.2f°C\n', setpoint);
end
function plot_response(time, temperature, setpoint, results, settling_percentage)
figure('Position', [100, 100, 800, 600]);
% Main plot
plot(time, temperature, 'b-', 'LineWidth', 1.5, 'DisplayName', 'Temperature');
hold on;
grid on;
% Setpoint line
plot([time(1), time(end)], [setpoint, setpoint], 'k--', 'LineWidth', 1.5,
'DisplayName', 'Setpoint');
% Steady-state line with error annotation
steady_state_line = plot([time(1), time(end)], [results.steady_state,
results.steady_state], ...
```

```
'g:', 'LineWidth', 1.5, 'DisplayName', sprintf('Steady-State (%.2f°C)',
results.steady_state));
% Annotate steady-state error
text(time(end), results.steady_state, ...
sprintf('Error: %.2f°C', results.steady_state_error), ...
'VerticalAlignment', 'bottom', 'HorizontalAlignment', 'right', ...
'Color', 'green', 'FontWeight', 'bold');
% Rise time indicators
plot([results.t10, results.t10], [min(temperature), max(temperature)], 'b--',
'LineWidth', 0.5);
plot([results.t90, results.t90], [min(temperature), max(temperature)], 'b--',
'LineWidth', 0.5);
patch([results.t10, results.t90, results.t90, results.t10], ...
[results.min_value+0.1*(results.max_value-results.min_value), ...
results.min_value+0.1*(results.max_value-results.min_value), ...
results.min_value+0.9*(results.max_value-results.min_value), ...
results.min_value+0.9*(results.max_value-results.min_value)], ...
'b', 'FaceAlpha', 0.1, 'EdgeColor', 'none', 'DisplayName', 'Rise Time');
% Settling time indicators
if results.settling_time > 0
settling_line = plot([results.settling_time, results.settling_time],
[min(temperature), max(temperature)], ...
'r--', 'LineWidth', 1.5, 'DisplayName', sprintf('Settling Time (%.1fs)',
results.settling_time));
end
% Settling band
settling_band = patch([time(1), time(end), time(end), time(1)], ...
[results.lower_bound, results.lower_bound, ...
results.upper_bound, results.upper_bound], ...
'r', 'FaceAlpha', 0.1, 'EdgeColor', 'none', 'DisplayName', sprintf('±%d%% Band',
settling_percentage));
% Labels and title
xlabel('Time (seconds)');
ylabel('Temperature (°C)');
title(sprintf('Temperature Response; 6.6ml\nRise Time: %.1fs, Settling Time: %.1fs,
Overshoot: %.1f%%, Steady-State Error: %.2f°C', ...
results.rise_time, results.settling_time, results.overshoot_percentage,
results.steady_state_error));
% Legend
legend('Location', 'best');
% Adjust axes
ylim([min(temperature)*0.9, max(temperature)*1.1]);
xlim([time(1), time(end)]);
hold off;
end
```