DUBLIN CITY UNIVERSITY
SCHOOL OF ELECTRONIC ENGINEERING

Smart Home Water Flow Sensing and Control with ESPHome and Home Assistant

Adelin Denis Diac

April 2024

BACHELOR OF ENGINEERING

IN

MECHATRONIC ENGINEERING

Supervised by Dr Derek Molloy

Acknowledgements

I want to thank my supervisor, Dr. Derek Molloy, for his guidance, enthusiasm, and commitment to this project and for preparing the template of the final report. I also want to thank Robert Clare and Dean McLaughlin for supporting me during this project when I needed any electronic components or support with the mechanical aspects of the project.

Declaration

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work. I understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. I have read and understood the Assignment Regulations set out in the module documentation. I have identified and included the source of all facts, ideas, opinions, and viewpoints of others in the assignment references. Direct quotations from books, journal articles, internet sources, module text, or any other source whatsoever are acknowledged and the source cited in the assignment references. This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study.

I have read and understood the DCU Academic Integrity and Plagiarism at https://www.dcu.ie/system/files/2020-09/1_-_integrity_and_plagiarism_policy_ovpaa-v4.pdf and IEEE referencing guidelines found at https://loop.dcu.ie/mod/url/view.php?id=1401800.

Name: Adelin Denis Diac **Date:** 26/02/2024

Abstract

This project explains the implementation of a smart home system and a flow control and water metering device. The report starts with the network setup, which consists of Home Assistant running as a Docker container inside a virtual machine and a mobile hotspot for IP connectivity. Two ESP32-based devices are designed and programmed using ESPHome. The first is a water flow control and metering system that allows water usage to be monitored, and also allows the control of the flow rate through it. The mechanical and electronic design of this device is explained. The second device is a temperature sensor that is used to show how the network setup can be used to easily integrate different smart devices. A control system is proposed that implements flow control and temperature sensing for automatic room heating. The response of this system is found by deriving an approximate model of the heat gain/loss in a room. The flow control valve position is correlated with the heat input into the room and was controlled using a PID controller, showing that this type of controller is able to maintain a stable response for the room temperature. The accuracy of the flow sensor was also tested, and results found that the YF-B7 flow sensor may not be accurate enough or may require further tweaking.

Table of Contents

Table of Figures

Table of Tables

Chapter 1 - Introduction

The evolution of digital technologies is transforming the way humans live. Improvements in computational power have paved the way for the Internet of Things to advance, allowing the smallest of sensors to interact with other devices. This growth, alongside the availability of open-source software, has led to improvements in many sectors, including home automation. These advancements have increased the functionality, efficiency, and safety of homes, which are now labelled as “smart homes.”

The definition of a “smart home” has changed drastically over the years. In the early stages, a smart home was thought to consist of different services integrated using a standard communication system [1], but in more recent years, a smart home is thought of as one which provides its occupants with sophisticated monitoring and control over the building functions [2]. This control over the functions has improved over the years and can be done automatically or via voice commands, physical control, remote control (using devices like smartphones), and many more.

1.1. Internet of Things (IoT)

The Internet of Things (IoT) is the term for the network of devices that connect the physical world to the Internet [3]. This allows for smart automation, data collection, remote access, and many other previously unattainable functions.

IoT’s impact extends beyond individual convenience. In smart homes, Internet of Things technologies allow the integration of heating, lighting, security, entertainment systems, and many more. This enhances a home's security and energy efficiency and improves the quality of life of the residents inside it.

The advancements of IoT also help address other challenges faced by society. IoT devices can reduce unnecessary energy consumption by enabling precise control and automatic analysis of the home environment. Additionally, these devices can assist elderly and disabled individuals [4] in regaining independence by monitoring their health and safety while providing alerts in case of emergencies, such as water leaks, which the system in this paper aims to detect.

Despite all the benefits of the Internet of Things, it also has downsides. There are many concerns regarding data integrity, privacy security, etc., which will be explained later in this paper. The potential for unauthorised access to devices is also a potential problem, which grows as more devices are connected to the Internet.

In summary, the Internet of Things has been instrumental in transitioning to a data-driven world, enabling the interaction between the physical and digital worlds. As the world advances, the Internet of Things will continue to play a critical role in improving the quality of life, energy efficiency and safety of homes.

1.2. Growth of the Smart Devices Market

The smart devices market is one of the fastest-growing markets in the world and is forecasted to continue growing. The IoT market is expected to reach $8131 billion worldwide by 2030 [5]. This growth has been fuelled by technological advancements in computational power, connectivity protocols, and more, allowing devices to become more powerful and readily available. Advancements in data analysis techniques, artificial intelligence (AI) and machine learning (ML) have enabled devices to become more intelligent and autonomous, which makes it easier for humans to integrate them into daily lives.

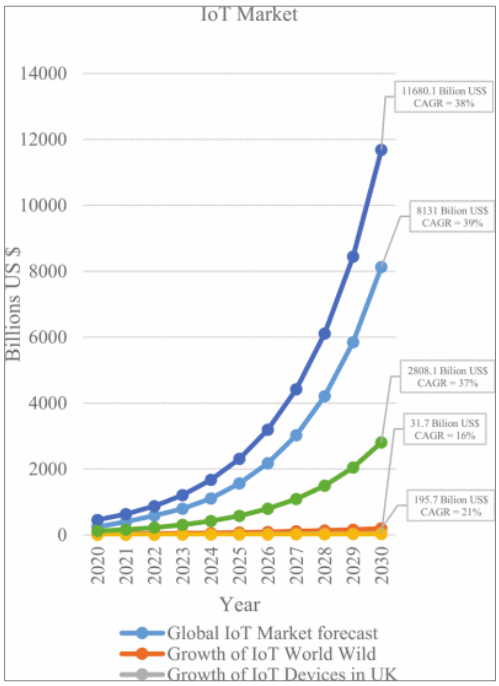


Figure 1.1 - IoT Market Forecast in USD [5]

As the market grows and technology advances, these devices will be integral to the human lifestyle. Advancements in AI/ML will make the devices more capable than ever, allowing each occupant of a smart home to have a truly personalised experience based on their preferences and previous trends.

As the market matures, a significant emphasis will be placed on a functional ecosystem that allows devices from different manufacturers and brands to communicate together [6]. One strong contender for such a system in the domestic market is “Home Assistant”, an open-source home automation platform. This ecosystem is explained thoroughly later in this report, as it is fundamental to the functioning of the solution created for this project.

Despite solid growth, the IoT market only shows signs of continued growth as technology advances. The future of smart devices looks promising for economic development, improving human lives, and increasing the efficiency of home systems.

1.3. Home Automation

Home automation is the interconnection of IoT devices into one ecosystem to allow the systems in a home to function automatically without much human interaction. The devices can communicate via Wi-Fi/IP over a network [7]. However, many more protocols are available, such as Bluetooth, Zigbee, and Z-Wave [8]. Devices that communicate over different protocols require some central hub to process the “protocol translation”. This can be a regular computer with additional hardware for each protocol, or existing solutions such as EVVR, Hubitat, Home Assistant Yellow, and many more can be used.

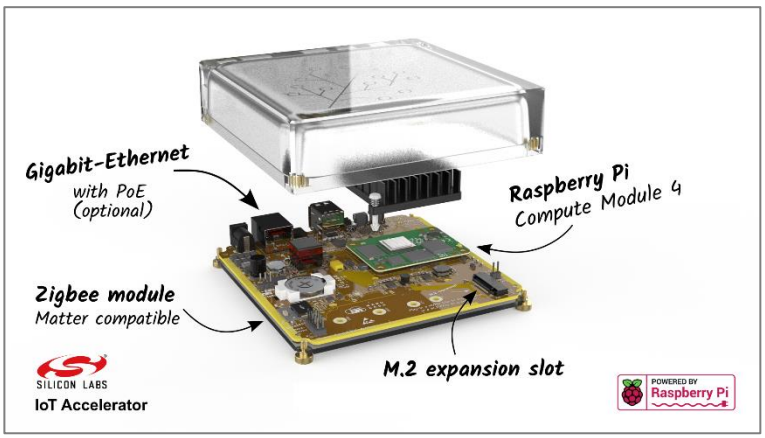


Figure 1.2 - Home Assistant Yellow [9]

The space will only expand as smart devices become more available and people become more familiar with the technologies fuelling home automation. The benefits of a smart home are appealing to more people every day, as they recognise the potential to make their homes more secure and pre-emptively address and prevent common household issues. Managing home systems to prevent problems like water leaks is a significant advantage. Leveraging smart water sensors and automation technologies can help homeowners detect leaks before much damage occurs, saving repair costs and water use.

1.4. History of Water Damage in Ireland

Water leaks are a significant problem in Ireland, not just in the residential setting, where leaks occur in a household, but also throughout the piping system. In 2019, the Irish supply system lost 43% of treated drinking water [10]. This is not counting water lost in individual households. The data is based on how much water enters a district water meter versus how much is leaving as it passes through customers' meters. Additionally, in 2015, when water charges were implemented temporarily, Irish Water found that 7% of installed water meters indicated a leak on the customer side [11]. This was around 46 million litres of water daily.

Not only are water leaks costly, but if they go undetected, they can cause many potential hazards. Small water leaks, which are only noticeable if there is a water monitoring system in place, can be very troublesome for homeowners, as these can leak for a significant amount of time, causing internal damage to the house that isn’t directly visible to the occupants until it has reached the edge of the walls. Depending on how long it took for the leak to be detected, the cost of fixing it can reach tens of thousands [12]. To make this worse, some insurance policies do not cover damage due to leaking baths or faulty showers [13]. If the leak is not detected early, the homeowner must pay a significant amount to repair the damage. Not dealing with a leak promptly, either because it was not a known problem or because of cost implications, could also lead to severe issues such as black mould [14], which will not only increase the cost of the repair but can also lead to serious health problems for occupants [15].

Based on these points, it is evident that being notified about potential leaks and being able to remotely or automatically shut off the water supply to sections of the house or the entire household can help reduce the cost of water and prevent large unforeseen expenses due to damage.

1.5. Water Metering and Flow Rate Control

The evolution of water metering and flow rate control technologies in the smart home sector, especially remote control and monitoring ability, will be critical to reducing water usage. In 2013, only one-third of UK households had a meter to monitor water consumption [16]. Monitoring water usage will play a pivotal role in reducing the effects of the climate crisis. Although the UK and Ireland do not have water charges implemented, individual households still pay for this through taxes or fixed fees for excess consumption. Implementing individual billing can reduce the cost of water [17], help the occupants understand their water use, and encourage them to implement methods to reduce their usage.

Flow rate control can be applied in many contexts, one of which is in domestic water meters. This could be used to reduce water supply to households, specifically during periods of high demand on the water grid [18], often seen during hot summer periods. Although water suppliers' ability to control water supply to their customers comes with sustainability, cost-saving, and operational benefits, there are also ethical concerns to be considered, as water is a fundamental human right [19]. This is explored in more detail in the Ethics section of this report.

Another potential implementation of intelligent flow rate control is in temperature control of rooms, where a single temperature sensor could control the amount of hot water flowing through a radiator. Other implementations could include intelligent irrigation systems, which might also be part of a larger-scale smart home system. Either way, having a system that allows sensors to communicate with each other or with a central processing hub is essential for a smart home to work effectively, especially if trying to automate the control of flow rates using multiple, non-related devices. This is explored in this report and is one of the main objectives of this project.

1.6. Open Source Software and Local Smart Home

The growth of open source projects has significantly impacted the development of technologies primarily due to community support, which allows new features to be implemented faster while being much more customisable to users. One benefit is that users can download the software and edit it themselves however they wish. They can also suggest

that the changes they made be implemented into the primary source of the code, usually hosted on an online platform such as GitHub.

Large commercial players such as Microsoft or Google develop or maintain most open-source projects. In the top 10 open-source projects (ordered by the number of contributors), Home Assistant Core is the only project not backed by a large company [20]. It is maintained by the community and its founder, Paulus Schoutsen. Home Assistant is also the fastest-growing open-source project [20], and as it continues growing, it is solidifying its place as the "go-to" home automation platform. This growth is mainly due to the "user-focused" development, meaning that user/community feedback is the main driver behind the implemented features. Another benefit of Home Assistant is its ability to integrate with almost any other smart device. To build on this, if the device has a "local API" (i.e. the device can be monitored/controlled locally on the smart home network without connection to a vendor-specific cloud platform [21], see Figure 1.3), then Home Assistant can control the device directly without having to go through a separate server. This vastly increases the system's security, reduces the latency of an action after it is triggered, and enables devices to be controlled without requiring an internet connection.

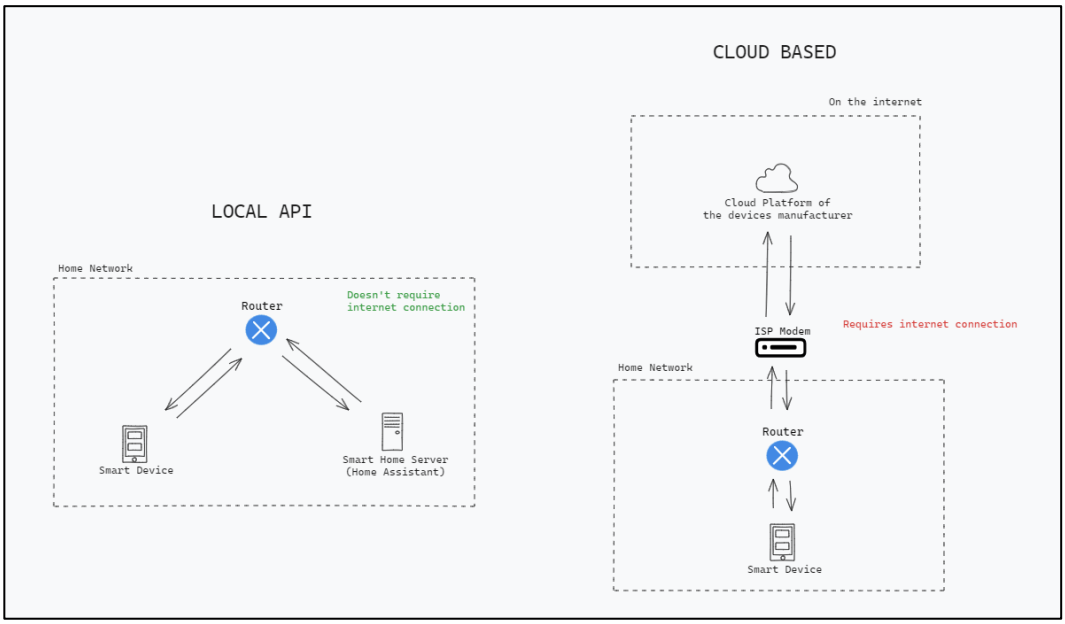


Figure 1.3 - Diagram of data flow for devices with a local API vs devices that work via a cloud platform

The open-source movement's transformative impact on home automation, mainly through platforms like Home Assistant, highlights an era of user-driven innovation and customisation. This community-centric approach to development has democratised technology and

accelerated the integration of diverse smart devices, enhancing security and user experience. Looking to the future, the ability to integrate devices using an open-source platform and control them locally will not only allow for a secure smart home but also a user-friendly one that does not depend on external servers to function smoothly.

1.7. Problem Definition

This project aimed to overcome several problems associated with smart water metering and leak detection systems. The main problem is the lack of water metering systems capable of detecting leaks, shutting off automatically/remotely, and integrating into “local-only” smart home systems, specifically Home Assistant, while remaining affordable to home occupants.

Another problem identified is the intelligent control of water flow, specifically in areas like zone heating. The issue arises from the absence of devices that can control the flow of hot water into a heater based on feedback from external temperature sensors. Typically, a thermostat toggles a switch when the temperature goes outside the desired setpoint, initiating the home’s heating. Addressing this challenge requires a flow valve that can partially open/close depending on the required amount of heating.

Due to the similarities of the two problems described in this section, addressing both issues with a single device presented an additional challenge. Solving these problems addresses the immediate issues at hand and lays the foundation for further automation or implementations using additional sensors. This approach opens up avenues for other implementations requiring precise flow control, thereby broadening the scope of smart home technology to encompass a wider range of functionalities and efficiency improvements.

1.8. Objectives and Report Structure

The project's primary objective was to develop an integrated solution for intelligent water metering, leak detection, and flow control compatible with Home Assistant. One aim was to keep the cost low (relative to other existing products). The implementation should be monitored/controlled remotely and wirelessly, but should also implement a display and physical controls.

This report will outline the steps taken to achieve these objectives, beginning with examining current solutions and research in Chapter 2. Here, an overview of available systems and the findings from relevant studies will be presented. Chapter 3 will delve into implementing the smart home infrastructure and the reasoning behind setting up a more complex system rather than “out of the box” solutions. The report will then explain the design and implementation of the edge device used to measure flow rate, total water volume, water temperature and controlling flow in Chapter 4. In Chapter 5, the design of a separate temperature sensor is explained. This will be used to show how the infrastructure setup in Chapter 3 allows many sensors to be easily integrated into the system, which will be demonstrated in Chapter 6. Chapter 7 will then discuss the testing and the results and conclusions that can be drawn from this. Some of the ethical concerns related to the topics of this project will then be discussed before concluding this report.

1.9. Motivation

The motivation behind this project stems from the need for more devices that can be controlled locally on a network independent of the vendor's cloud server. This aims to give more power to the customer, as they are in control of their data and also allows them to integrate the device into their home automation system.

The specific device being designed is a water flow meter due to the increasing need for water metering in Ireland and the UK, as well as the need to detect leaks in homes to avoid water damage and other issues arising from undetected water leakages. On top of this, flow control is also a feature not implemented in many systems, and having the ability to control the flow of water into one appliance based on input from a completely separate device/system shows the extent of how important system integration in a smart home can be.

1.10. Summary

This chapter introduced the sections explored by this project and discussed some of the existing associated problems, beginning with the introduction of smart home systems and the Internet of Things, before moving on to the current and predicted growth of the smart devices market. It then took a deeper look into home automation and how it benefits occupants of a home. The history and problems of water leaks and the damages they cause were explored, shifting onto how intelligent water metering and flow rate control could mitigate these while

reducing water usage and making homes safer and less wasteful. These were then tied together by explaining the benefits of open-source software and how Home Assistant, the fastest-growing open-source project, can integrate water control and monitoring methods to tackle the problems explained while remaining a secure and user-friendly platform for enabling and automating smart homes.

This chapter then explained the problems this project aimed to overcome, highlighting the objectives that need to be met to assess the issues to a desirable extent. It also described the structure of this report, the motivation behind this project, and some of its implementations.

Chapter 2 - Literature Survey and Existing Solutions

2.1. Academic Literature

2.1.1. Design and implementation of a low-power ultrasonic water meter [22]

Y.-S. Hong and C.-H. Lee, “A design and implementation of low-power ultrasonic water meter,” *Smart Water*, vol. 4, no. 1, p. 6, Nov. 2019, doi: 10.1186/s40713-019-0018-9.

In this paper, the authors aim to design a low-power ultrasonic water metering system. They model a system that should run for at least ten years before the batteries need replacement. The project takes advantage of recent developments in low-power and high-performance ultrasonic microcontrollers to allow for the fast processing of signals with low-power usage. The most significant advantage of ultrasonic water meters, unlike traditional water meters, is that they can be used to measure flow rate accurately, not just the volume of water flowing through, and they also don’t contain any mechanical parts. Despite these advantages, ultrasonic sensors are much more expensive than other water meters or flow rate sensors.

This research paper was explored and studied in the early stages of the project when comparing the different methods and technologies used to measure flow rate. An ultrasonic flow sensor would have been an ideal choice for this type of project. However, the cost constraints were a significant barrier to implementation. As technology advances, if these sensors can become cheaper, then they will be the standard water metering solution, allowing super-accurate measurements with very little power consumption, as proved in this paper by Y. S. Hong and C. H. Lee.

2.1.2. Designing a Cost-Effective and Reliable Pipeline Leak Detection System [23]

J. Zhang, "Designing a Cost Effective and Reliable Pipeline Leak Detection System," in *Proc. Pipeline Reliability Conference*, Houston, USA, Nov. 1996, pp. 1-11.

This paper explores various methods of detecting pipe leaks, including biological, hardware-based, and software-based approaches. It explains that all methods have advantages and disadvantages and compares them based on seven key attributes**: leak sensitivity**, **location estimate capability**, **operational change**, **availability**, **false alarm rate**, **maintenance requirement**, and **cost**.

The paper goes on to explain specific methods used to detect leaks based on each heading and some examples of where they were applied successfully. The paper finds that "False Alarm" is a problem associated with all methods except "biological," which cannot be done on a large scale or continuously. Therefore, this paper proves that water leaks and being able to detect them successfully have been a problem identified for a long while, and with technological advancements, it is getting easier to do so successfully.

Unfortunately, none of the leak detection methods mentioned could be implemented due to hardware constraints. However, this paper was an excellent introduction to the history of how leak detection might've been accomplished.

2.1.3. Sensitivity of water meters to small leakage [24]

A. Pietrosanto, M. Carratù, and C. Liguori, "Sensitivity of water meters to small leakage," *Measurement*, vol. 168, p. 108479, Jan. 2021, doi: 10.1016/j.measurement.2020.108479.

This paper explains how water leakages at household levels are a problem that is becoming more visible to the public, especially in the current state of the world where water must be saved. The author proposes that the generation of leak alerts could be processed at the device level, even if battery-powered, rather than the more traditional method of sending the data to a central database. This differs from the solution proposed in this project, where the water flow data is saved in a local database and processed accordingly. Both methods come with pros and cons, and one advantage of the implementation proposed in my report is that battery devices might not be as reliable in the long term, especially for heavier processing and data analysis.

The paper then looks at two methods of implementing leak detection algorithms, with the first being based on the total water used in a day. The graphs are split into zones 1 when water

usage does not increase (usually during the night) and 2 when water usage increases (usually during the day). The second method requires the instantaneous flow rate, which shows the usage, in terms of litres per second, with the time on the x-axis. The author explains how this method is preferred because one can easily identify “Periods With Null Consumption (PWNC)” in the graph.

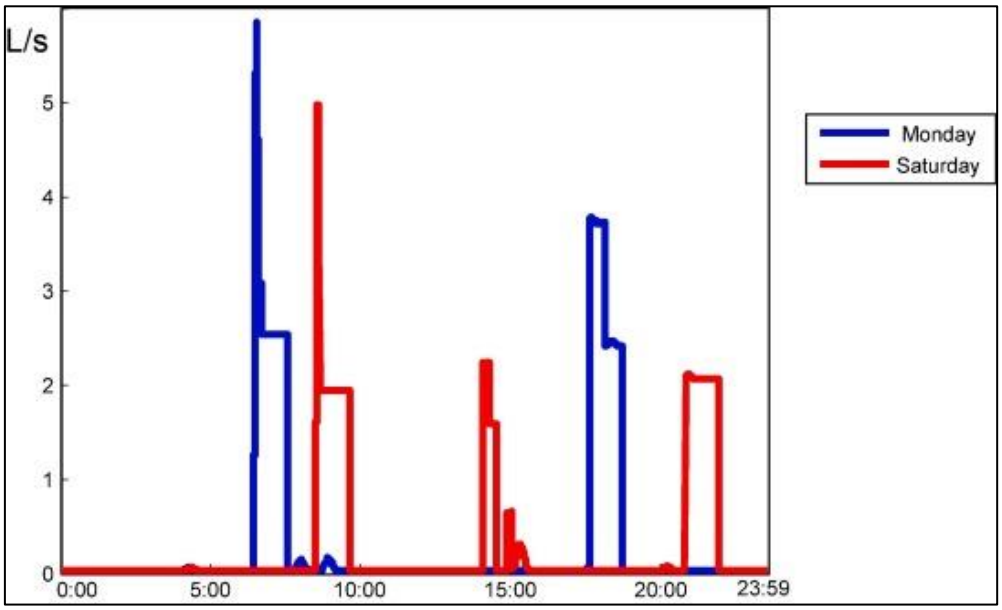


Figure 2.1 - Graph of instantaneous water usage vs. time from [24]. Note PWNC.

The author then tests the sensitivity of various types of water meters, concluding that there is a requirement for high-resolution measurements. This is not fully satisfied because of the low resolution of water meters and the limitations of battery-powered electronics, which this project aims to improve on.

2.1.4. Development of Motor-Operated Ball Valve [25]

R. Ghube, K. Kap, and M. G. Ghogare, "Development Of Motor Operated Ball Valve," in *Journal of Instrumentation & Control Engineering Department*, vol. 1, no. 19, pp. 23, Apr. 2012.

This paper explains in great detail the components used in their implementation of a motorised ball valve. The rotation of the ball valve is controlled by the amount of current flowing into an analogue-to-digital converter (ADC). The current is between 4-20 mA, and the size of the current corresponds to a rotation between 0 and 90 degrees. The implementation uses a servo

motor, which can monitor its rotation using the built-in potentiometer. This implementation was a significant motivator behind the design implemented in this project; however, with slight changes due to hardware restrictions.

2.2. Existing Products

2.2.1. StreamLabs Control

The StreamLabs Control is a smart water meter with an integrated shut-off valve. This product was a significant motivator for this project, as it implements many of the desired features, such as intelligent shut-off when a leak is detected, an ultrasonic water meter, and built-in WiFi for remote monitoring. Although it is a great product that meets almost all the criteria of a smart water flow meter, it does come at a high price of $999.

Figure 2.2 - StreamLabs Control Water Meter [26]

2.2.2. Water Hero

The Water Hero is a similar implementation to the StreamLabs Control product. However, this does not seem like a thoroughly engineered product. Rather than an all-in-one solution such as StreamLabs, seen in Figure 2.2, the Water Hero seems more like a combination of separate products sold as a “package”. This “package” consists of a water meter, a motorised shut-off valve, and a processing unit, which may also include the product's WiFi capability. Despite the product not looking as professionally done as the StreamLabs Control, it still comes at a hefty price, starting from €838.95. Figure 2.3. shows the Water Hero as seen on

their website. This product further proves that more affordable intelligent water meters with shut-off and flow control capabilities are needed.

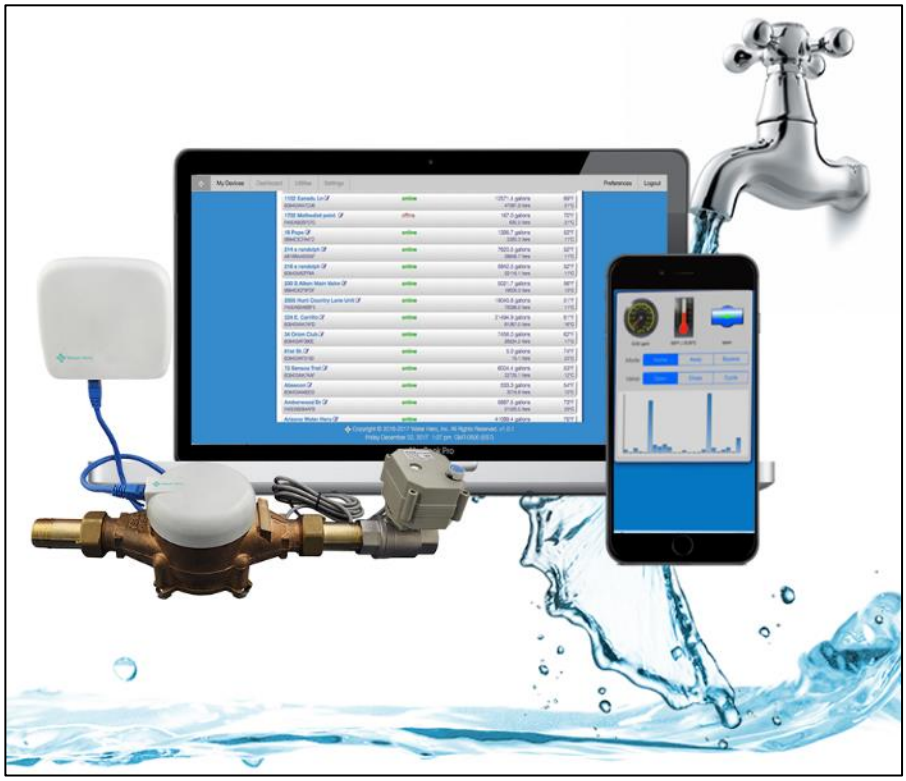


Figure 2.3 - Water Hero [27]

2.2.3. Flo Smart Water Monitor and Shutoff – Moen

This is another fully-fledged product, similar to StreamLabs Control. However, the water monitoring system does not use an ultrasonic sensor but rather a turbine with magnets that can measure water flow. Based on some of their online reviews, the product is not the most reliable, with an average of 3.1 stars from 107 reviews. Most negative reviews mention that it is not reliable in the long term, especially for an expensive product. The cheapest smart water meter on the Moen website is $499.98 (reduced from $767.70). Customers also mention that the shutoff valve will automatically shut itself off, without warning the homeowner, if it thinks it has detected a leak, which may not always be accurate. They also mention that it would be preferred for the meter to warn the occupant, who then can decide whether the water should be shut off. This will be implemented in this project and further backs the need for reliable and affordable smart water monitoring and shut-off.

Figure 2.4 - Flo by Moen Smart Water Monitor [28]

2.3. Summary

This section explored some of the existing literature published in smart water management, specifically leak detection, ultrasonic flow measurement, and water flow control. It explained how specific papers motivated sections of this project and the reasons for opting for different solutions compared to those in the research papers.

This chapter then examined existing products in the market and identified gaps in them. Some implementations offer much control and data; however, they also come at a cost. This shows that an affordable solution for such a common problem is required to help reduce homeowners' costs and wasted water due to leaks.

Chapter 3 - Smart Home Infrastructure

Before working on the device itself, the foundation of the "smart home" must be set up. This includes network connections and software used as the central smart home platform.

3.1. Network Setup

3.1.1. Project vs. Smart Home Structure

For the network, internet protocol (IP) networking [29] was used due to its reliability, comprehensive implementation in homes, and the fact that microcontrollers with Wi-Fi capability are widely available. Other protocols commonly used in smart homes are Zigbee and Z-Wave. However, these are newer protocols, with microcontrollers not as available or affordable. IP networking was used to keep up with the aim of implementing a low-cost device. This also helps users of such a device avoid purchasing additional hardware to implement other protocols like Zigbee/Z-Wave.

In a typical home network, a router is the centre of the network. Nowadays, a single device will act as the router while also doing the functions of many other devices. It assigns IP addresses to the connected devices so they can be identified on the network, routes traffic between devices, allows wireless connectivity between devices (Wireless Access Point (WAP)), and much more [30]. The built-in "hotspot" function on a Windows laptop was used for this project to make the network portable. This creates the network, acts as the router, and allows the devices to connect wirelessly.

The smart home hub in this project was a Home Assistant. In typical setups, this is installed on a separate computer and may be able to run additional programmes on it (depending on the Home Assistant setup – explained in section 3.3). For this project, again, to maintain portability, a separate computer was set up on the same Windows laptop via a virtual machine (VM). A virtual machine is essentially a computer that runs inside another computer [31]. A hypervisor is the software that coordinates this and is usually software that runs on an existing operating system (called a Type 2 Hypervisor [32]). It accesses the underlying hardware to run the virtual computer [32]. This project used VirtualBox [33] to run a computer with the Ubuntu operating system [34], an open-source, free-to-use OS. This Ubuntu server will be the central hub of the "smart home" in this project.

The edge devices then connect to the router via a wireless or cabled connection, depending on the device. In this setup, the devices connected to the laptop's hotspot wirelessly. The internet connection in a home is provided by a modem supplied by the internet service provider (ISP) [30]. In this project, the host laptop was connected to either the home network or the Eduroam network in DCU, allowing internet access. However, the edge device was designed so that an internet connection is not needed. Figure 3.1 shows what this high-level network infrastructure would look like in a typical smart home vs. how it was replicated in this project.

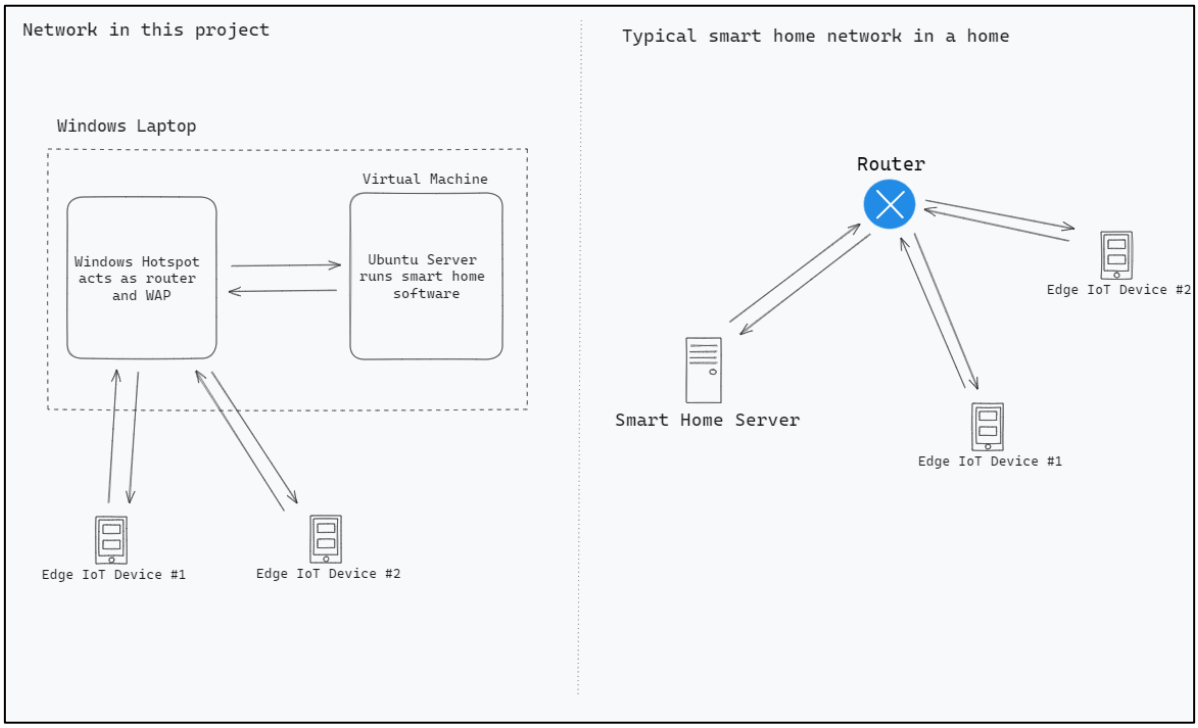


Figure 3.1 - Network setup in this project compared to how it would be set up in a home

3.1.2. Challenges

One challenge in this setup was that edge IoT devices connected to the hotspot would be given a randomly assigned IP address. This is not preferred as the smart home server communicates with the devices via the IP address, and having it change each time a device disconnects from the network would cause problems in communication. This was resolved by assigning a fixed IP address to the devices from their software, which is explained more thoroughly in the corresponding sections of the edge devices (Chapters 4 and 5).

3.1.3. Reason for Complicated Setup

Such a complicated setup was established to allow for a scalable smart home network. In the current setup, the central smart home server can be anything from a small computer running just Home Assistant to an entire server running many other software applications. It also allows for hundreds of devices to be connected, which will all be able to communicate with the server and one another to provide a seamless smart home experience.

3.2. Containerisation – Docker and Portainer

3.2.1. Containerisation

Before delving into the software used in this project, explaining the concept of "containerisation" is essential. This method, similar to virtualisation, allows the separation of applications and services on a computer. The most significant difference is that containers run inside a "Container Engine" on the host operating system. It is not required to use different "machines" to separate applications as you would when using virtualisation methods [35]. Hence, it is a lot lighter and easier to maintain.

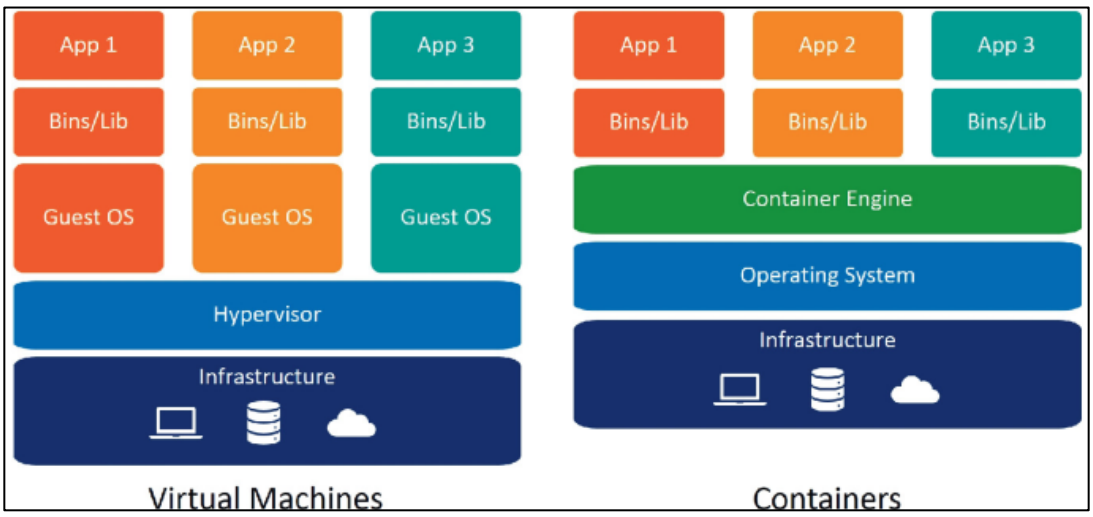


Figure 3.2 - Difference between Virtualisation and Containerisation [35]

3.2.2. Docker Engine

Docker is one example of a container engine that can be used to manage containers on a machine. Containers are executable components that contain application source code with the required libraries and dependencies [36]. Docker allows users to build, test, and deploy applications quickly. It is an open-source application that can be run on any computer or

server. Docker helps teams maintain a consistent environment for development and production, reducing errors associated with different versions of packages or runtimes, etc. It can be maintained from the command line on the server, or various GUI (Graphical User Interface) tools exist, like Docker Desktop or Portainer.

3.2.3. Portainer

Portainer is a management user interface (UI) that allows users to manage different Docker environments and containers directly from a web browser [37]. The Docker engine physically separates the containers, and Portainer will enable you to view all the information about them in a web browser. This is specifically useful when installed on a "headless" [38] (no display, keyboard, mouse, etc.) server because viewing the containers on the command line can be difficult and confusing.

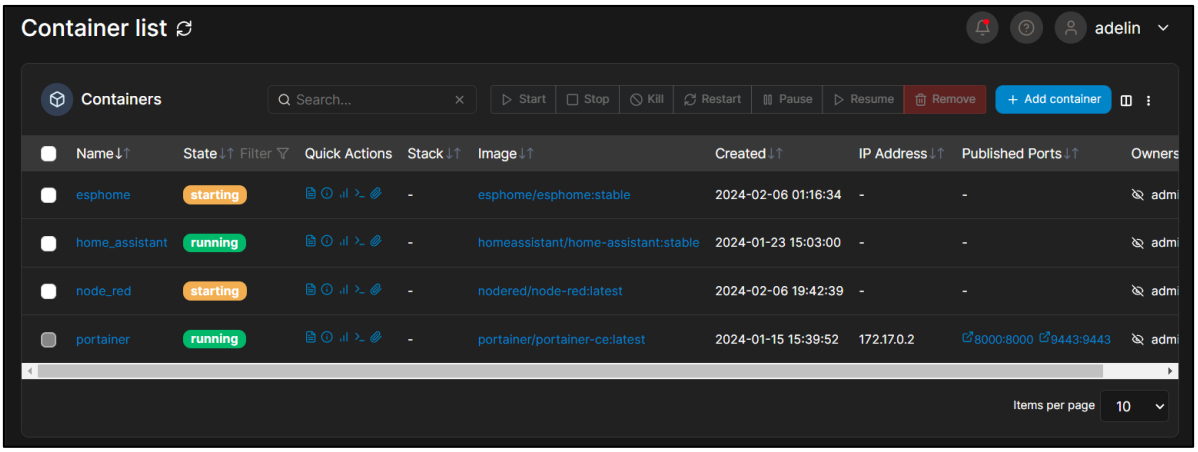


Figure 3.3 - View of the Portainer Dashboard, looking at the currently installed containers

In this project, Docker and Portainer were installed on the virtual Ubuntu machine. All the software applications used in the "smart home" are run inside Docker containers so that they can run concurrently without a problem. Portainer was used to monitor these containers from a web browser.

3.3. Home Assistant

3.3.1. Introduction

Home Assistant [39] is an open-source home automation platform that works on the basis of "local control" with the option for cloud connectivity. This means that the software can be installed on a local machine/computer in one's home network, and it will function without

having to connect to a third-party server for anything to work. The most significant advantages that make this platform so popular is that all data remains within the local network, so other parties cannot access it, and it works without needing an internet connection. It only requires a router for communication between IP devices (and any other hardware necessary for different protocols, such as Zigbee, Z-Wave, etc). Another main attraction to Home Assistant and open source software, in general, is the escape from "vendor lock-in", hence giving devices from different brands (Google, Amazon, Govee, Tuya, Apple and more) the ability to communicate with each other, something which is not always possible on the individual platforms.

3.3.2. Setting up Home Assistant

Home Assistant can be set up in many ways [40]. The main methods are:

- **Home Assistant Operating System (HAOS),** a standalone operating system that can be installed on a mini PC, Raspberry Pi [41], Virtual Machine, or other devices. It is the easiest way to get started without much technical knowledge other than installing an operating system on a computer.
- **Home Assistant Container**, which can run as a Docker container. This slightly more complicated installation method offers more flexibility over the applications installed on the machine and allows Home Assistant to run alongside other applications.
- **Home Assistant Core**, a manual install using a Python virtual environment. Although the user has much more control over the software, this method removes the benefits of Docker Containers and requires much more advanced technical knowledge to set up.

3.3.3. Chosen Setup

Initially, Home Assistant was set up by installing the HAOS on an SSD drive and connecting this to a Raspberry Pi using a USB connection. This hardware was readily available and did not require any upfront costs. SSD provides longer life and faster performance [42] than a standard SD card generally used on a Pi. This allowed a quick setup of Home Assistant to get familiar with it. The container approach was then implemented by setting up the Home Assistant image from Docker Hub onto the virtual machine running inside the Windows laptop. This was done from the Portainer web UI. The benefit of this approach is that now the virtual machine runs Home Assistant in its own environment, basically isolated from

everything else on the machine, but still allows other applications to be deployed, allowing us to take full advantage of the machine's hardware. Typically, this setup would be done on a computer on the home network, which the virtual machine replicates for this project.

Expanding on Figure 3.1, which shows the network layout, a deeper look at the virtual machine can be taken. The software and applications architecture on this machine look as per Figure 3.4, which allows for horizontal expansion of the services running on this machine, represented by the three dots. Ubuntu is installed on the VM hardware, with Docker running as the containerisation engine.

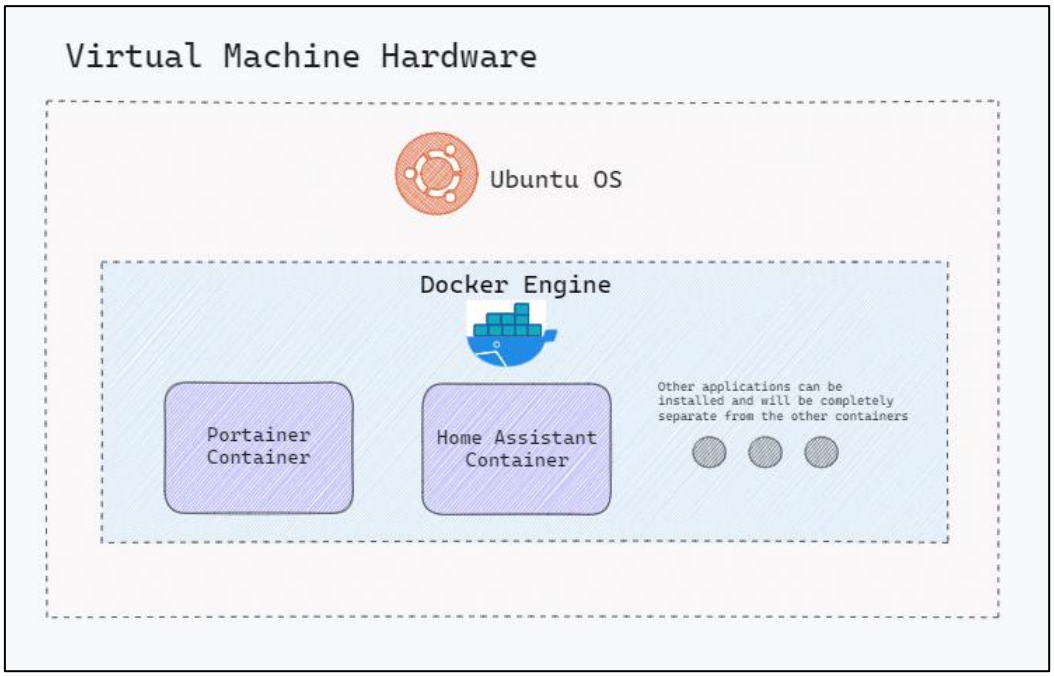


Figure 3.4 - Software Architecture on the Virtual Machine

3.4. ESPHome

3.4.1. How it Works and Setup

ESPHome [43] is an automation platform that simplifies the programming of Wi-Fi microcontrollers, such as ESP8266 and ESP32. It can also flash smart devices that implement an ESP-based chip [44]. It allows microcontrollers to be programmed using YAML [45] configuration files with built-in components that exist in the codebase. The ESPHome compiler turns the YAML file into C++ when installing the code. This is then compiled again into a binary file, which can be uploaded to the microcontroller. An example of what a YAML to C++ translation looks like is shown below for a basic GPIO component.

Firstly, inside the devices configuration file, named [DEVICE_NAME].yaml, the following code is typed:

Figure 3.5 - Example ESPHome YAML configuration

This enables GPIO4 to be a binary sensor, which monitors the state (High or low) of a GPIO pin on the ESP. With this configuration, some options automatically go to their default values; hence, ESPHome converts it to the following configuration (present in the main.cpp file):

Figure 3.6 - ESPHome configuration for binary sensor with all default values included

This configuration then gets converted to this C++ code:

```
gpio_gpiobinarysensor = new gpio::GPIOBinarySensor();
App.register_binary_sensor(gpio_gpiobinarysensor);
gpio_gpiobinarysensor->set_name("Test GPIO Binary Sensor");
gpio_gpiobinarysensor->set_object_id("test_gpio_binary_sensor");
gpio_gpiobinarysensor->set_disabled_by_default(false);
gpio_gpiobinarysensor->set_component_source("gpio.binary_sensor");
App.register_component(gpio_gpiobinarysensor);
esphome_esp8266_esp8266gpiopin = new esphome::esp8266::ESP8266GPIOPin();
esphome_esp8266_esp8266gpiopin->set_pin(4);
esphome_esp8266_esp8266gpiopin->set_inverted(false);
esphome_esp8266_esp8266gpiopin->set_flags(gpio::Flags::FLAG_INPUT);
gpio_gpiobinarysensor->set_pin(esphome_esp8266_esp8266gpiopin);
```

Figure 3.7 - C++ Code generated by ESPHome in main.cpp

This generated C++ code gets much deeper, each class and function called in the *main.cpp* file was created and implemented. The benefit of ESPHome is clear from this example, as much more functionality was implemented with just four lines of code. A lot of time is saved during the development stage. Unfortunately, the downside is that if a particular sensor or component is unavailable in the ESPHome source code, it must be added in. Thankfully, thanks to a growing community, there are over 500 components available, and the contributors are adding in many more. A big attraction to ESPHome is its native compatibility with Home Assistant. Components initialised in the YAML configuration file can be controlled in multiple ways, some of which include:

- Connect to the access point hotspot of the ESP device and go to its main page on a web browser (must be enabled in the YAML configuration).
- By going to its main page on a web browser when the device accessing the browser and the ESP are connected to the same network (access is via the ESP IP address).
- Control it via Home Assistant using the ESPHome integration (it must be enabled in YAML, but by default, it is on when creating a new device configuration).

In this project, ESPHome was installed as a Docker container inside the Ubuntu virtual machine. The image was pulled from the Docker hub and deployed to run parallel to Home Assistant and Portainer. This now ran as a separate application and was accessed via a browser at *http://[Ubuntu VM IP address]:6052*.

Physically uploading the generated code onto the device itself only needs to be done the first time. ESPHome also generates the binary file from the generated C++ code. This can be downloaded from a browser onto whatever computer is accessing ESPHome. The ESP device must be connected to this computer, and the code can be uploaded from ESPHome web. If this is done correctly, the device appears as "online" on the ESPHome dashboard, and now it can be updated "over the air", i.e. without connecting the device to the computer (as long as both devices are connected to the same network).

3.4.2. Special Components

General Components

General components are all those integrated into the main ESPHome codebase and can be included in the YAML configuration file. They can be anything from the simple "binary_sensor" example from before to more complicated components like the "web_server," which enables the device to be monitored/controlled from a built-in web page on port 80 of the device. There are over 500 components. The source code for these can be viewed at https://github.com/esphome/esphome/tree/dev/esphome/components.

Lambda Functions

One powerful ability of ESPHome is the ability to easily add custom C++ code, called "Lambdas", almost anywhere in the configuration file [46]. This allows for the inline definition of complex logic while still maintaining the ease of use of the built-in components in ESPHome, and it can be particularly useful when a custom calculation must be performed when processing a sensor reading. An example of a *lambda* function that selects which page is displayed on an LCD screen using a "*switch*" statement in C++ can be seen in Figure 3.8.

```
display:
  - platform: lcd_pcf8574
    dimensions: 20x4
    address: 0x27
    id: lcd
    lambda: |-
          switch (id(page)){
            case 1:
              it.print(0, 1, "Page1");
              break;
            case 2:
              it.print(0, 1, "Page2");
              break;
            case 3:
              it.print(0, 1, "Page3");
              break;
          }
```

Figure 3.8 - Example of lambda function in use [46]

External Components

Another powerful example of ESPHome's flexibility is its ability to use external components. These personal or community components can be used without being merged into the main ESPHome codebase [47]. The advantage of this is that development can be sped up and devices/components tested without the code being merged into the ESPHome releases. The downside, however, is that these components may not have been thoroughly tested and can cause unforeseen errors.

3.4.3. Summary

This section explored the open-source project ESPHome, which allows Wi-Fi-based microcontrollers to be programmed using simple YAML configuration files. It explained the benefits and downsides, with a deeper look into some key components. It also explained how it works and how it was set up for this project.

3.5. Node-RED

3.5.1. How it Works and Setup

Node-RED [48] is a flow-based development tool for visual programming. It is ideal for connecting hardware devices, APIs, MQTT topics, online services, etc. It is an open-source application started by two IBM engineers in early 2013 [49]. Similar to the previous applications in this project, it is deployed as a Docker container inside the Ubuntu virtual machine. Node-RED is accessible in a browser at port 1880. There are many integrations available, one of which is Home Assistant. It can read and automatically run “flows” and “nodes” of logic based on values from devices connected to Home Assistant. In this project, it will be used as a “processing” hub to process the data from various sensors, creating a seamless “smart home” integration between different devices and the central server.

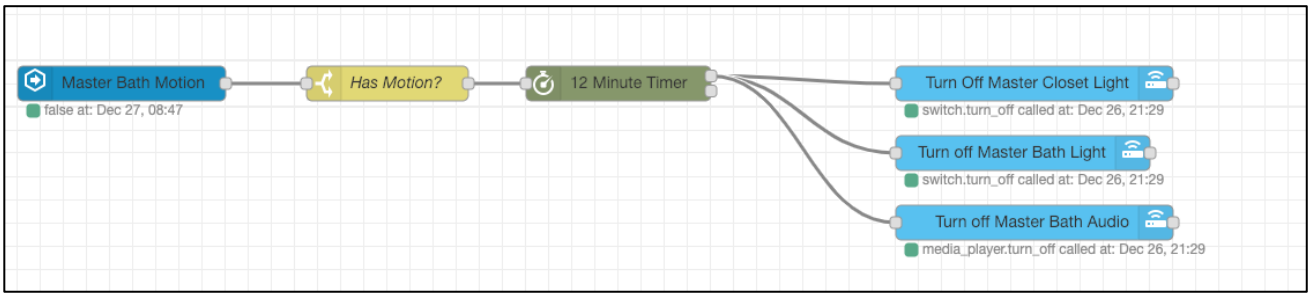


Figure 3.9 - An example of a simple "flow" in Node-Red that calls Home Assistant services.

Integrating Node-RED with Home Assistant required some setup. The Home Assistant (HA) package was first added to Node-RED using the palette manager. A HA server had to be configured by including the base URL of the HA instance. To allow secure communication between the HA and Node-RED instances, a long-lived access token had to be generated from the HA dashboard; this was included in the “Access Token” box.

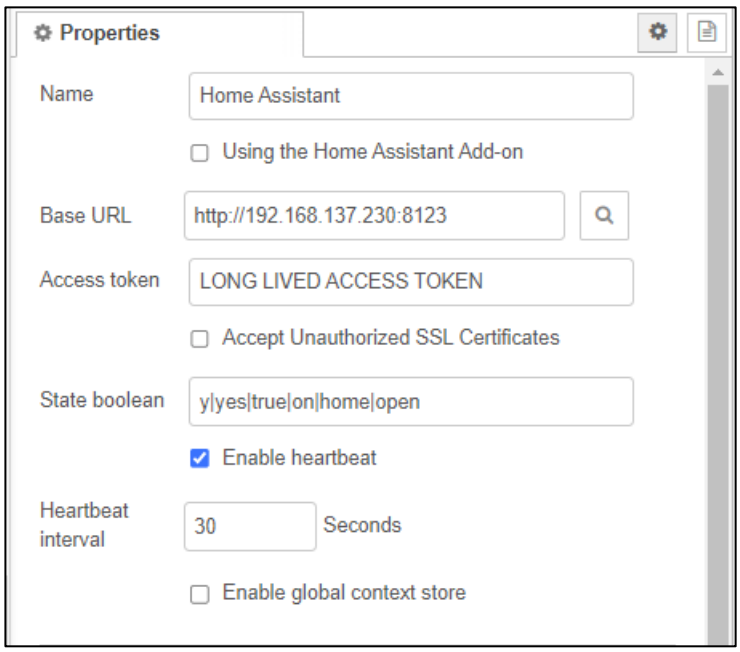


Figure 3.10 - Home Assistant Server setup in Node-RED

3.5.2. Comparison to Home Assistant Automations

Unlike Home Assistant’s built-in automations, Node-Red offers a more intuitive graphical user interface that enhances visibility and makes interactions between components easier to read. Its drag-and-drop approach facilitates the creation and management of automations in a smart home.

3.5.3. Summary

Node-RED is an advantageous replacement for Home Assistant automations, particularly for those prioritising ease of use and visual clarity in their automation setups. Its Docker-based deployment ensures compatibility and ease of access, marking it an invaluable tool in the smart home automation landscape.

3.6. Leak Detection

3.6.1. Setup and Implementation

The leak detection functionality was implemented inside Node-RED using the Home Assistant nodes. The “event: state” node is used, which monitors state changes to a particular variable. This was chosen as the platform of choice to show the power and ease of use of Node-RED. Another way to set this up could have been by deploying another container in Docker, which would host the analysis software. This could be done in a language such as Python [50], which is a commonly used high-level programming language when doing data analysis. However, setting up in Node-RED required less time and, due to time constraints, allowed for a baseline solution to be implemented.

The leak detection functionality in this project works based on “periods of null flow rate” mentioned in section 2.1.3 of this report. It checks if the flow rate during periods of 30 minutes is greater than 0. If it is, it then moves into the next node of the automation, which checks if the current time is inside a specific time frame – selected as between 2 am and 5 am in this project. This period was chosen as it is inside the period of least water usage [51], but it can easily be changed based on the occupant's schedules or potentially by implementing algorithms that compare to previous water usage data. A “Notify” node was then used to send a critical notification to a mobile device registered in HA.

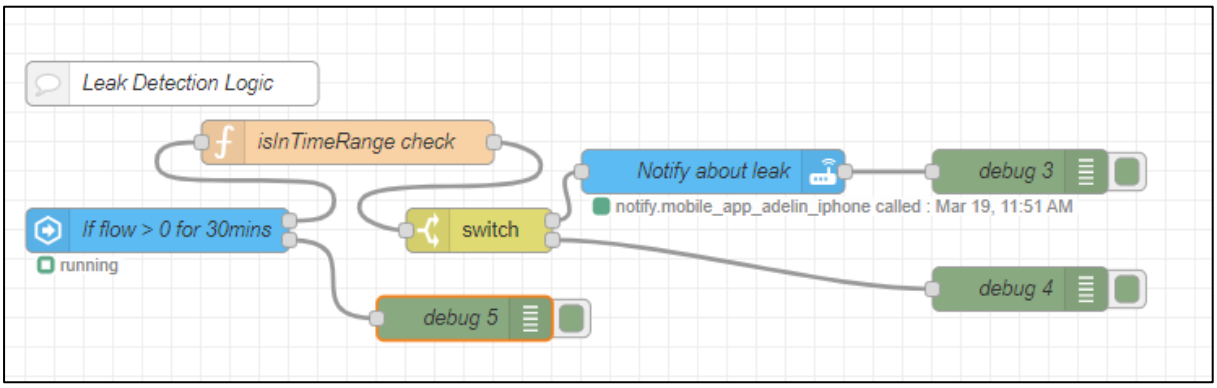


Figure 3.11 - Full Leak Detection Flow in Node-RED

The ”*event: state*” Home Assistant block is the entry point into the leak detection flow. It contains a lot of functionality in just a simple node. It was configured to check the flow rate reported to Home Assistant and apply the logic explained in the previous paragraph.

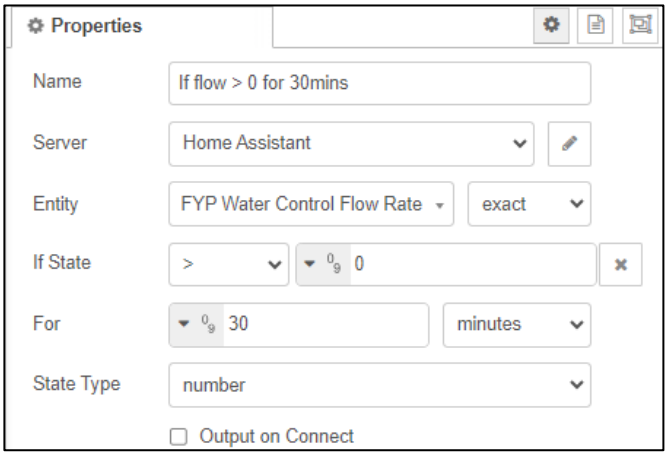


Figure 3.12 - Flow rate node configuration

The next section checks that the current time is within the set hours. This was implemented using a “function” block, which lets you write custom code using the JavaScript programming language. The “*msg*” variable is what gets passed between nodes inside Node-RED.

```
1
2   let currentHour = new Date().getHours();
3   msg.isInTimeRange = (currentHour >= 2) && (currentHour < 5);
4
5   return msg;
```

Figure 3.13 - Logic to check if current time is withn required hours

The value of “*msg.isInTimeRange*” was set to either true or false based on the current hour. This was then passed into the “switch” statement, which allows the flow to progress into separate sections based on the value of a variable. So, in this scenario, if “*msg.isInTimeRange*” was true, it would trigger the “Notify” node; otherwise, it would go into the “debug” node, which in this case is just a way of closing off the logic by logging the “*msg*” variable.

The notification to the phone was configured to send as a “critical” alert, which makes a noise even if the phone is on silent. The phone must allow HA to send critical alerts for this to work. Otherwise, it is just sent as a regular notification.

```
{
    "title": "Leak Detected",
    "message": "A leak has been detected in HA",
    "data": {
        "push": {
            "sound": {
                "name": "default",
                "critical": 1,
                "volume": 1
            }
        }
    }
}
```

Figure 3.14 - Leak Detected Alert Configuration

Notifications are one of the only parts of Home Assistant that depend on an internet connection to work. This is because they use either Google or Apple notification services (depending on what device the app is installed on) [52]. This functionality is beneficial; however, it does concern users about the privacy of the notification data as these are typically saved in plain text on the company's servers (explored in the Ethics section of this report).

This simple but effective flow shows how advancements in open-source software allow users to create powerful functionality quickly and cheaply. Previously, this would have required much more technical knowledge or purchasing of an entire product.

3.6.2. Advantages, Disadvantages and Future Work

This implementation shows how easy it is to get set up on creating and hosting complicated algorithms without needing external servers to process the data or to purchase expensive hardware. Once this foundation is laid, complex leak detection algorithms can be implemented using the built-in options and custom components, like the "function" block used in this implementation.

The most significant disadvantage is the requirement for a very sensitive sensor to detect small leaks. Future work involves the creation of a more thorough leak-detection algorithm and the implementation of a separate pressure sensor, which can work with the flow sensor to check if the pressure in the pipe changes when the valve is closed. This can increase the system's redundancy while reducing false alarms and increasing the accuracy of alerts.

3.7. Entire Infrastructure

This section shows the entire diagram of the communication between all the services in this project, the chosen architecture, and the connections between devices. This is seen in Figure 3.15 below:

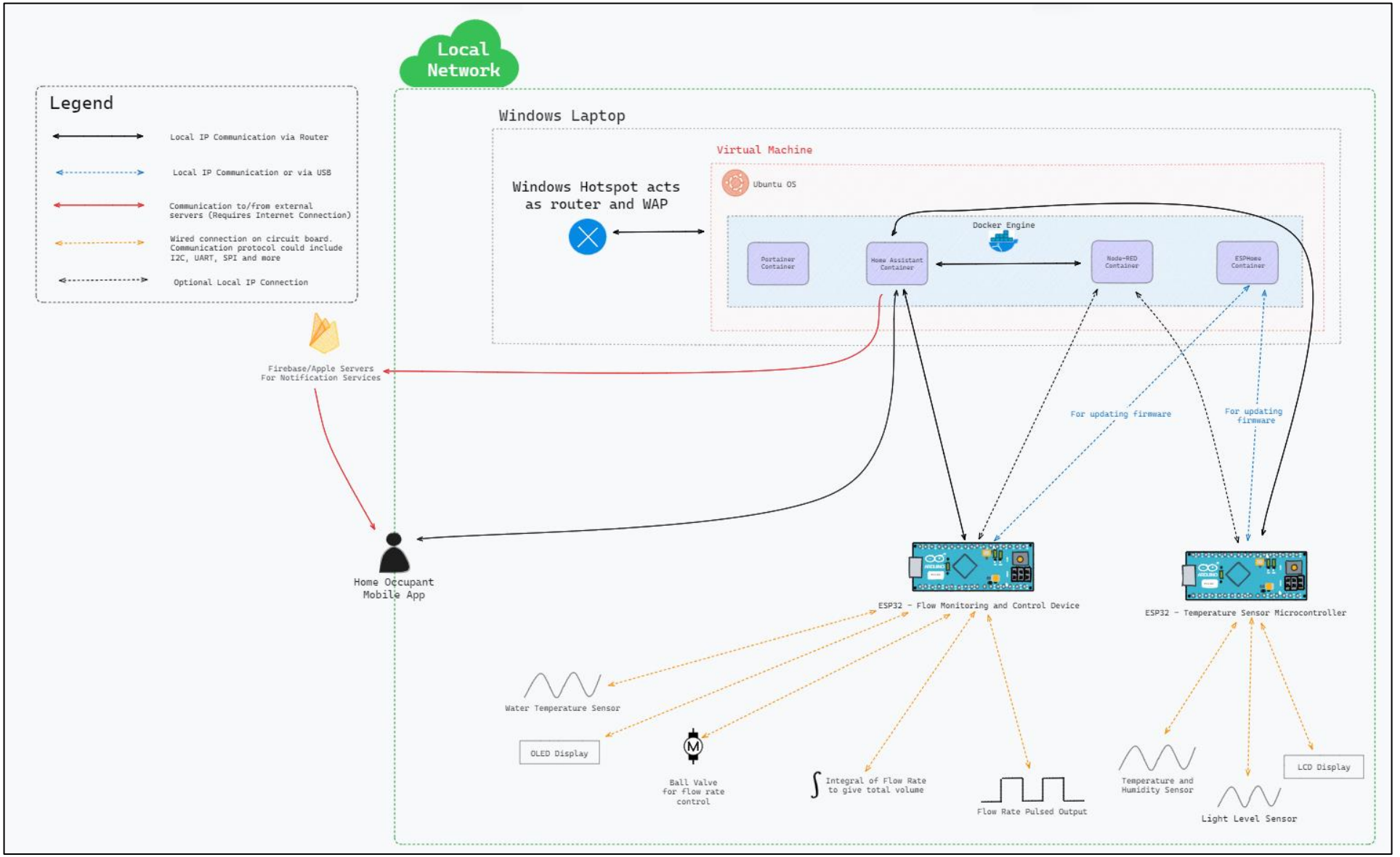


Figure 3.15 - Architecture showing communication lines between services and devices

3.8. Summary

This section outlined the foundational setup for a scalable and customisable smart home system. It discusses the network configuration utilising IP networking due to its availability in homes, enabling easy integration with widely available microcontrollers. A virtualised Home Assistant hub forms the central control platform, running on a portable Windows laptop hotspot network for this project, ensuring flexibility and mimicking the network setup in a typical home. Docker and Portainer are used for containerisation, allowing multiple applications to run in isolated environments and enhancing maintainability and stability. This complex setup, though intricate, ensures that the infrastructure can expand to accommodate a growing number of devices and software applications, providing a robust framework for a comprehensive and interconnected smart home ecosystem.

Chapter 4 - Edge Device Design

4.1. Microcontroller

In the IoT realm, various microcontrollers that offer wireless capability, specifically Wi-Fi, are available. These include the Arduino series with Wi-Fi modules, smaller Raspberry Pis, ESP8266s, and ESP32s, to name a few. The ESP8266 and ESP32 are very similar in price and availability; however, the ESP32 is faster, more powerful, and comes with more GPIO pins [52], eliminating the ESP8266 as a contender. The choice was between the remaining three options.

The decision was to go for the ESP32 because it can be programmed using ESPHome, so it can easily integrate into Home Assistant and allows quicker prototyping compared to the Arduino. It also does not require an operating system to be installed, compared to the Raspberry Pi. Its presence in industry and commercial products is also growing, specifically in building automation systems [53], showing that it is an excellent microcontroller for small projects and prototyping and becoming a respected choice for scalable commercial applications.

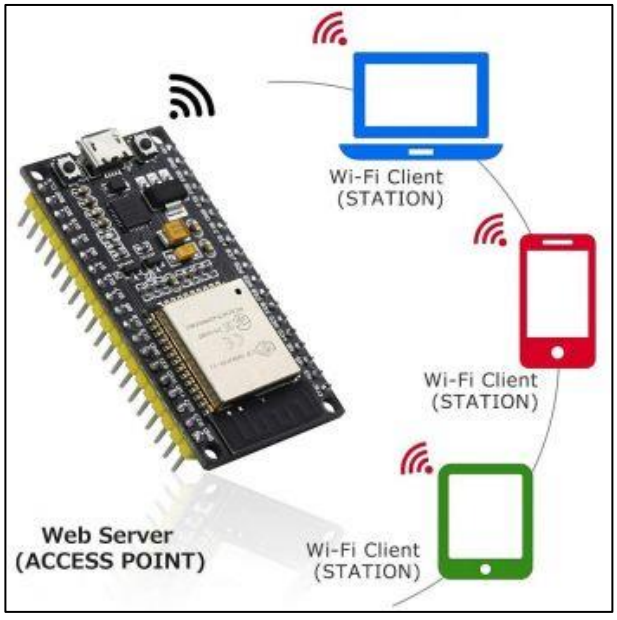


Figure 4.1 - ESP32 Development Board and some of its functionalities [54]

4.2. Programming

As explained in section 3.4, ESPHome was used to program the ESP-based devices. The ability to program devices "over-the-air" (OTA) was used to deploy new software easily. To do this, the device must first be programmed via a USB. This is done from the ESPHome dashboard, which can be opened from any computer on the same network as the device running the ESPHome container.

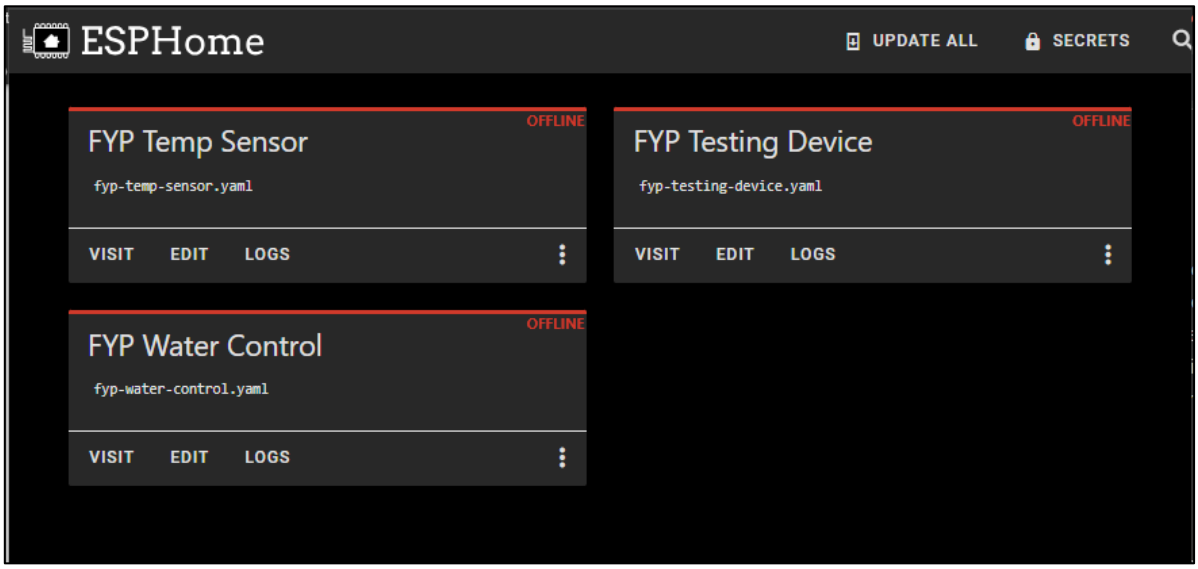
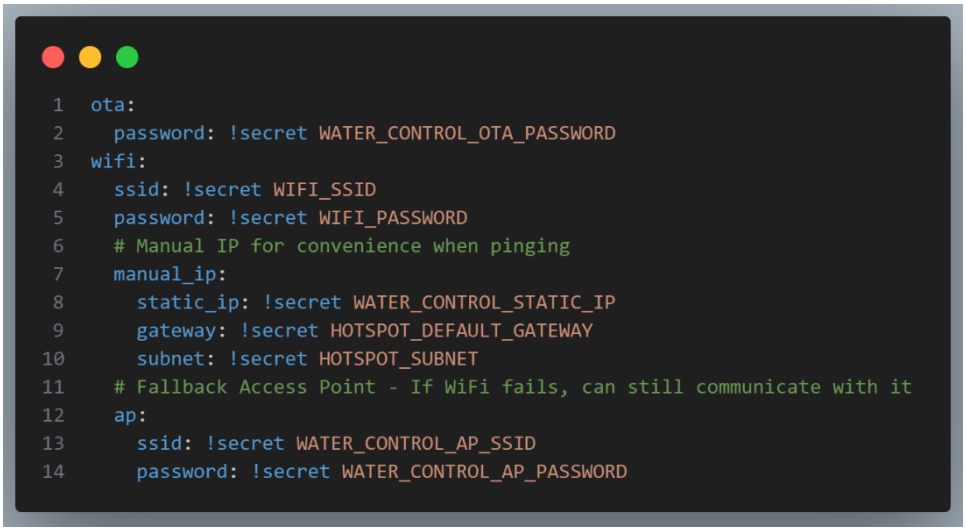


Figure 4.2 - ESPHome Dashboard viewed from a web browser

The initial binary file was downloaded by pressing the three dots in the bottom right corner of the rectangle and selecting "Manual Download". Once downloaded, the page opened "web.esphome.io", which can view the devices connected to the computer via the USB cable and allows the file to be uploaded onto the ESP. Once the first installation was complete and the device was powered, it appeared "Online" on the dashboard and could be updated wirelessly. This feature is secure because when first setting up the core configuration of a device in ESPHome, it generates a password that must be supplied by the server trying to update the microcontroller. This password is only known by the device and the ESPHome container. The WiFi connection configuration and OTA updates were implemented as per Figure 4.3. A manual IP address was set because the Windows hotspot acting as the router would change the dynamic address every time the device re-connected, causing problems with communications. An "access-point" (ap) mode was also configured. This allows the device to create its own "hotspot" when it is not connected to the WiFi network. This helps if the WiFi network connection fails and communication is needed to the device.



```yaml
ota:
  password: !secret WATER_CONTROL_OTA_PASSWORD
wifi:
  ssid: !secret WIFI_SSID
  password: !secret WIFI_PASSWORD
  # Manual IP for convenience when pinging
  manual_ip:
    static_ip: !secret WATER_CONTROL_STATIC_IP
    gateway: !secret HOTSPOT_DEFAULT_GATEWAY
    subnet: !secret HOTSPOT_SUBNET
  # Fallback Access Point - If WiFi fails, can still communicate with it
  ap:
    ssid: !secret WATER_CONTROL_AP_SSID
    password: !secret WATER_CONTROL_AP_PASSWORD
```

Figure 4.3 - WiFi and OTA YAML configuration in ESPHome

All sensitive variables were stored in a secrets file built into ESPHome, so they could not be viewed in plain text on the configuration if, for example, the file was shared with others (such as in this report).

4.3. Home Assistant Integration and Individual Control

ESPHome devices have a built-in integration with Home Assistant. This can be configured using just three lines of YAML code, as seen in Figure 4.4. This contains an API key

```yaml
api:
  encryption:
    key: !secret WATER_CONTROL_API_KEY
web_server:
  port: "80"
```

Figure 4.4 - Home Assistant integration and web server configuration

automatically generated by ESPHome and is used to integrate the device into Home Assistant. To integrate the device, inside the Home Assistant ”Devices and Services” dashboard, “Add Integration” was selected. The “ESPHome” integration was found, which required a ”Host” setting to be entered. This was the manual IP set in Figure 4.3. It also required an API key, which was auto-generated when creating the device configuration.

The ”*web_server*” configuration in Figure 4.4 enables the ESP to serve a web page on the selected port. This page shows the sensor values, controls, and logs and lets you interact directly with the device from any standard web browser. This is useful if one or two devices are deployed; however, an integrated system like Home Assistant is preferred for a scalable smart home.

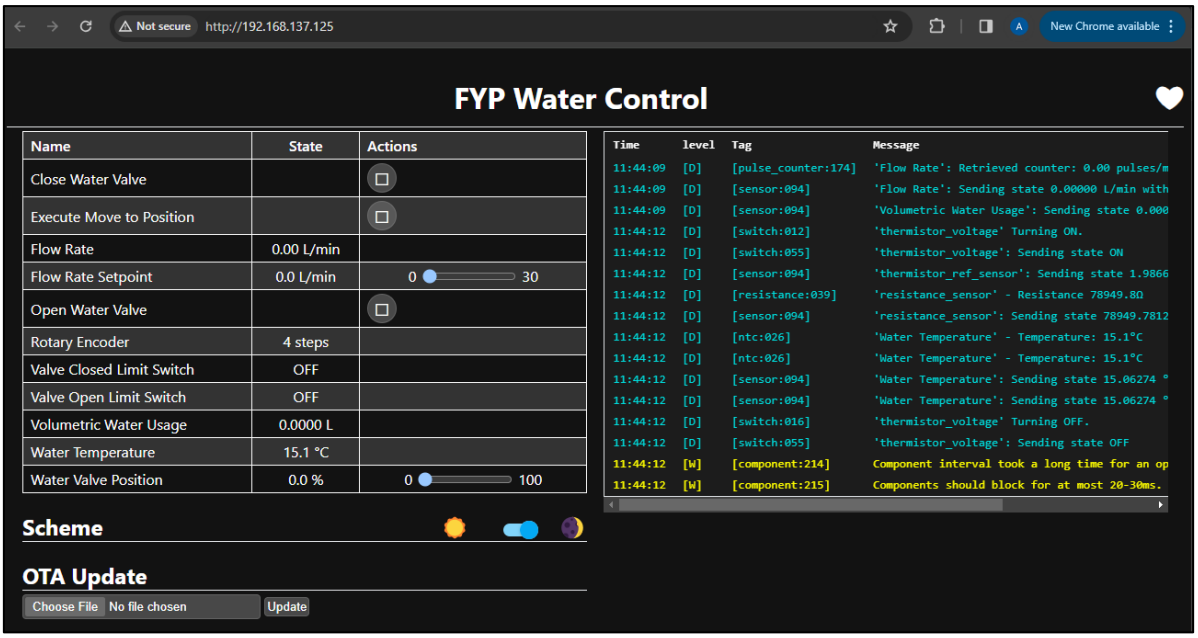


Figure 4.5 - Web page hosted by the ESP device allowing monitoring and control from browser

4.4. Flow Meter – Instantaneous + Total Volume

4.4.1. Sensor Options

Once the core configuration of the device was completed, including a WiFi connection, OTA updates, web server integration, and HA integration, the actual sensors and actuators could be implemented. The first sensor required was a flow sensor or water meter. The difference between these is that a flow sensor will generate the flow rate of the water, whereas a water meter will generate a pulse for a fixed volume measurement. This fixed volume reading could be divided by the time between outputs; however, a dedicated flow rate sensor would be preferred for a more “real-time” measurement of the flow rate.

Eliminating the water meter leaves two main types of sensors that can be used to measure water usage and flow rate: ultrasonic and Hall-effect sensors.

Hall Effect Sensors

These relatively cheap sensors use the Hall Effect, which produces a potential difference created due to a magnetic field [55]. The frequency of pulses is used to calculate the flow rate through the sensor.

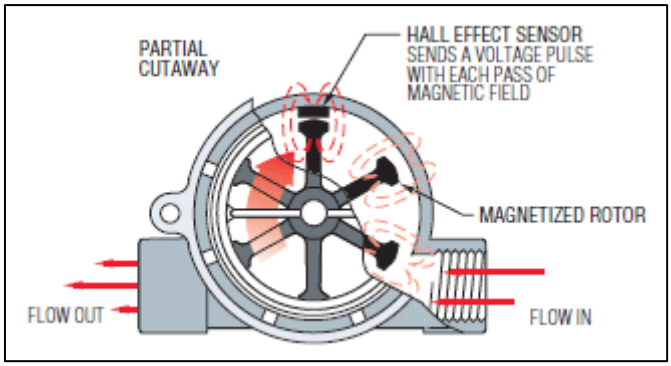


Figure 4.6 - Hall Effect Water Flow Sensor Diagram [56]

Ultrasonic Sensors

These are expensive but highly accurate sensors that work using two transducers, each of which sends a signal the other receives. Since water is flowing in one direction, the time it takes the signals to reach the other sensor will be different. The difference between the time of flight of the two signals is used to calculate the speed of water flowing through the pipe [57].

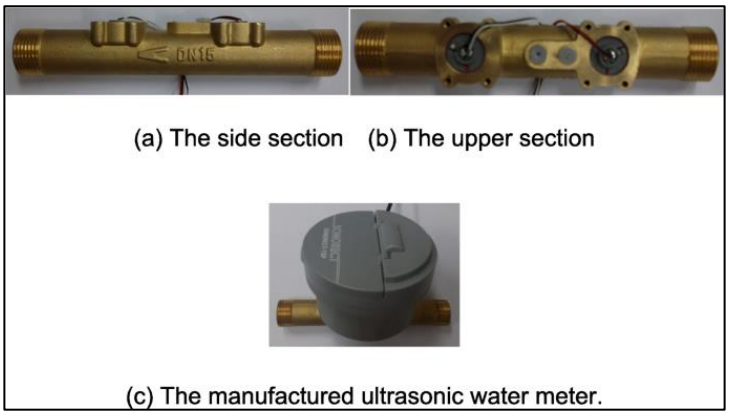


Figure 4.7 - Example of Ultrasonic Transducers on a water pipe [57]

4.4.2. Chosen Sensor and Connection to ESP

The ideal water flow sensor would be the ultrasonic sensor, which is highly accurate and does not require any moving parts to be in line with the water. Unfortunately, these are extremely expensive (as seen on the StreamLabs product, which costs around $1000). If this project

were to be commercialised, the ultrasonic sensors would be the best; however, to keep costs down, a simple Hall Effect sensor was purchased (YF-B7) [58]. This was ideal for getting some hardware set up and allowing initial testing to take place. The advantages of this sensor were its affordability, ease of setup and the extra add-on of a temperature sensor (thermistor).

Figure 4.8 - YF-B7 Water Hall Effect Sensor [58]

This sensor came with five wires. Two of these were for the thermistor, used to measure the temperature of the water (explained in section 4.4.5), and three for the flow sensor. These three wires were labelled as follows: V_{cc}, GND, and Data (with the colours red, black, and yellow, respectively). The sensor requires at least 5V to be powered, and the "Data" signal would be a pulsed output of 4.7V. This could not be wired directly to the ESP32 because the pins are only 3.3V tolerant [59]. Therefore, a voltage divider or a step-down level shifter needed to be implemented to step down the voltage. The voltage divider was not chosen because it is unreliable for high-frequency signals [60] (such as that from the flow sensor output). Therefore, a step-down level shifter was implemented, using a transistor and two 10k Ω resistors.

A transistor is an electronic component with three pins: base, collector, and emitter. It can be used as an electronic switch, allowing a large current to flow between the collector and emitter by applying a small current on the base [61]. Using it as a step-down level shifter required the following set-up: the signal to be stepped down connected to the emitter pin. The base was connected with a pull-up resistor to +3V (or 3.3V in the case of this project as that is what the ESP32 supplies), and the collector was connected to +3V (or 3.3V) with a pull-up resistor, and also into the GPIO of the microcontroller (see figure 4.9)

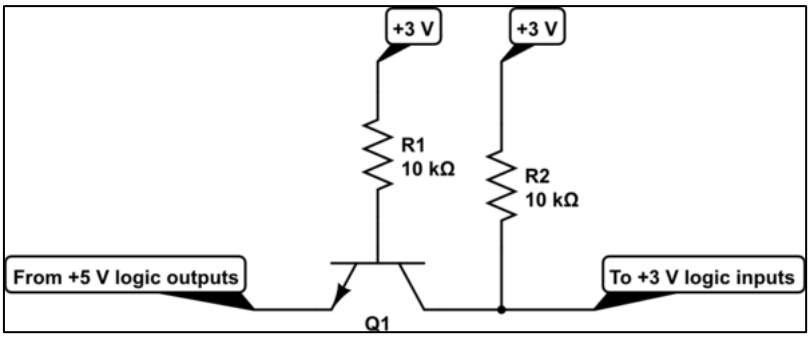


Figure 4.9 - Circuit for step-down logic shifter using a single transistor [62]

The transistor used for this was a TIP29C, as it was readily available, allowing the circuit to be built without purchasing new parts. Its current and voltage ratings are above the requirements for this project (the maximum base voltage is 5V, and the maximum pulsed collector current is 3A [63]).

The logic shifter was then connected to the flow sensor, and the pulsed output that would be used to calculate the flow rate was connected to GPIO16 on the ESP32.

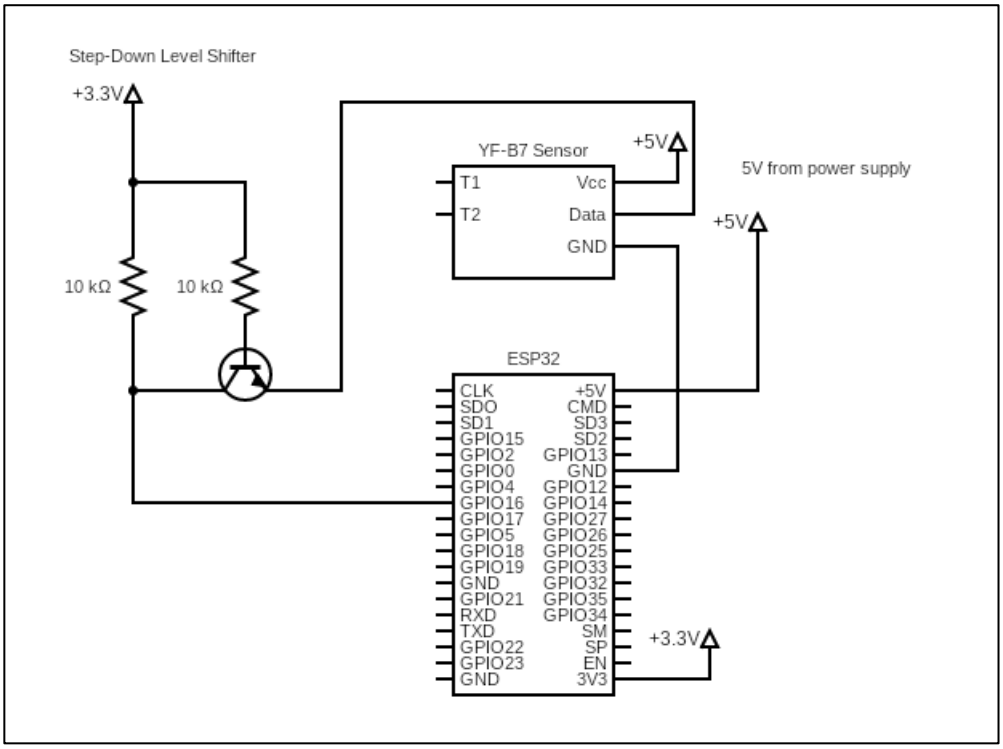


Figure 4.10 - Circuit Diagram for connecting Hall Effect flow sensor with step-down logic shifter to ESP32

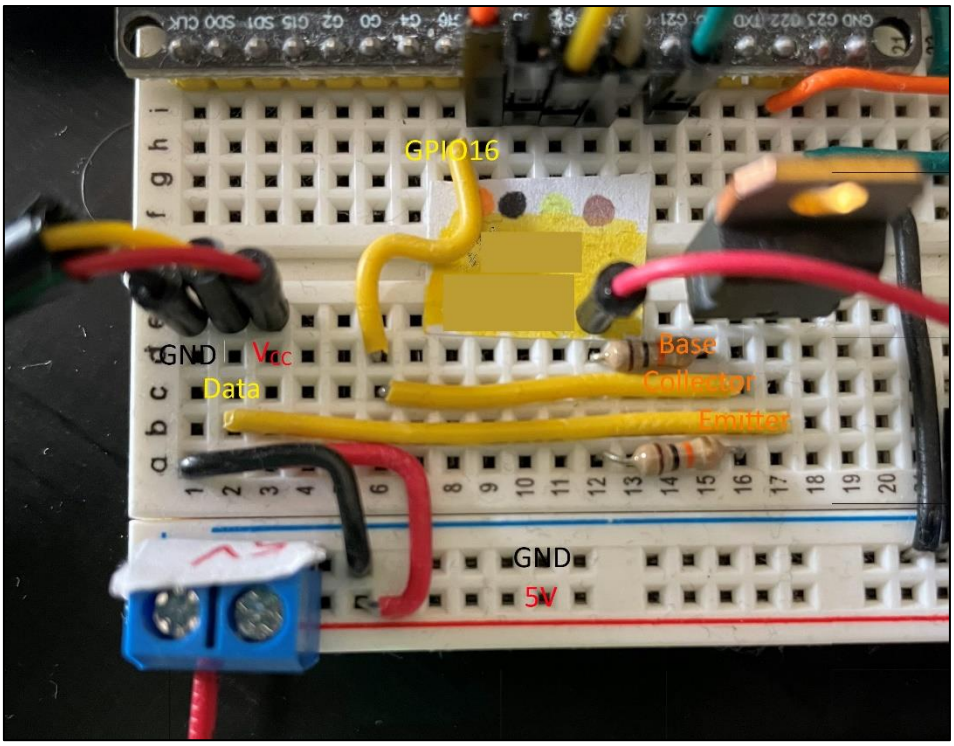


Figure 4.11 - Actual wiring for YF-B7 and step-down logic shifter

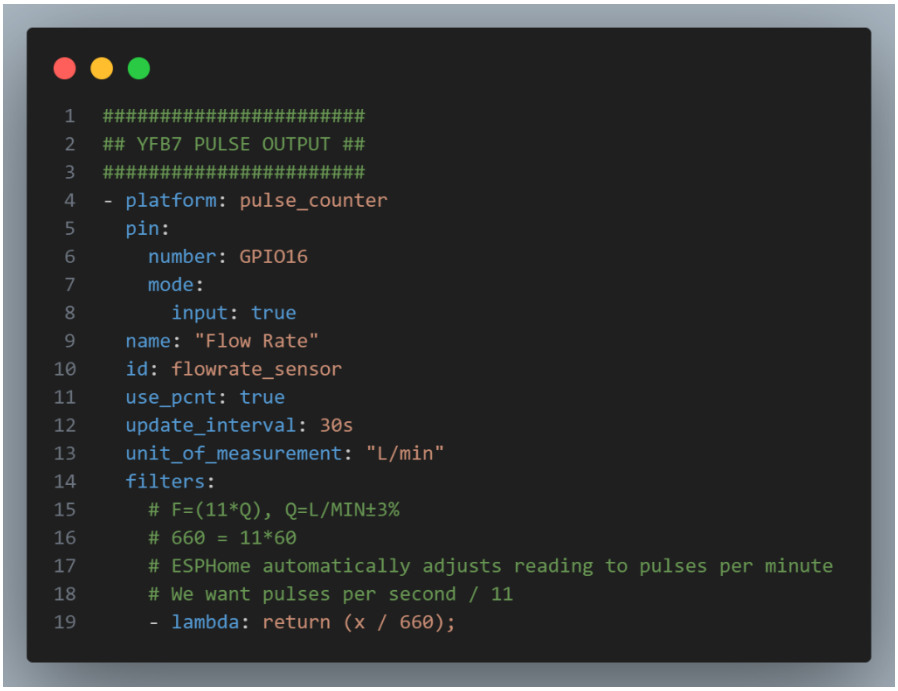
4.4.3. Pulse Counter

Once wired, the device had to be programmed. The first thing was turning a pulse frequency going to GPIO16 into a flow rate. The datasheet of the YF-B7 gives the following formula:

$$F = 11 * Q \quad (1)$$

F is the frequency of the pulses, and Q is the flow rate in litres per minute. For programming, ESPHome has a built-in component for a “pulse_counter”, which will automatically count the number of pulses on a pin. The number of pulses is counted during the “update_interval” duration and converted into a “pulses per minute” reading. So, if the update interval is 30 seconds, and 30 pulses are counted during this time, ESPHome converts this into 60 pulses/minute. A “lambda” function is applied to convert the pulses per minute into litres/minute by dividing by 60 * 11.

For this project, the update interval was set to 30 seconds to avoid causing too much load on the ESP32, but future work could include testing the accuracy at different update intervals to see which gives the best results.



```yaml
########################
## YFB7 PULSE OUTPUT ##
########################
- platform: pulse_counter
  pin:
    number: GPIO16
    mode:
      input: true
  name: "Flow Rate"
  id: flowrate_sensor
  use_pcnt: true
  update_interval: 30s
  unit_of_measurement: "L/min"
  filters:
    # F=(11*Q), Q=L/MIN±3%
    # 660 = 11*60
    # ESPHome automatically adjusts reading to pulses per minute
    # We want pulses per second / 11
    - lambda: return (x / 660);
```

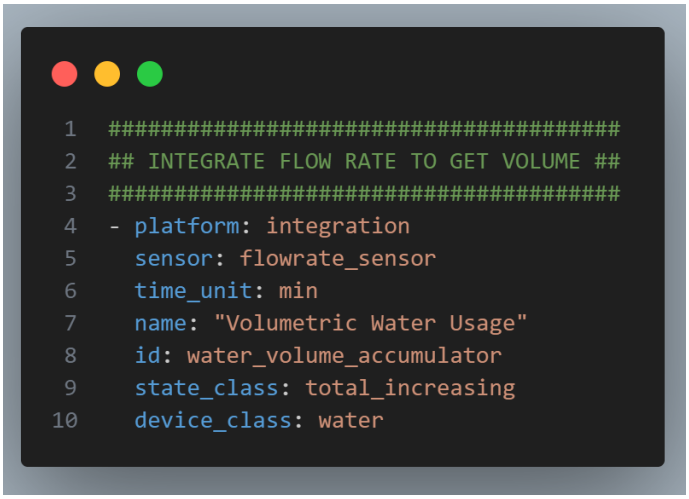
Figure 4.12 - YAML configuration for pulse counter to flow rate

4.4.4. Water Volume Accumulator

The next step was to turn the water flow reading into a total volumetric measurement in litres. To do this, the "integration" component from ESPHome was used. Since the sensor data is litres per minute, which is just the rate of change of volume per time, integrating it with respect to time (minutes) will result in a value measured in litres:

$$\int \frac{d(litres)}{dt} dt = litres \tag{2}$$

This is easily implemented in ESPHome as:



```yaml
########################################
## INTEGRATE FLOW RATE TO GET VOLUME ##
########################################
- platform: integration
  sensor: flowrate_sensor
  time_unit: min
  name: "Volumetric Water Usage"
  id: water_volume_accumulator
  state_class: total_increasing
  device_class: water
```

Figure 4.13 - ESPHome configuration for an integration data point

4.4.5. Water Temperature Sensor

The YF-B7 water flow sensor also came with a built-in thermistor. A thermistor is a variable resistor whose resistance changes with the surrounding temperature [64]. To implement this with the ESP32, it must be connected in series with a different resistor. As the temperature changes, the voltage between the thermistor and ground will change, which will allow the resistance and the temperature to be calculated. The general formula for a thermistor is:

$$B = \frac{T_2 * T_1}{T_2 - T_1} * \ln\left(\frac{R_1}{R_2}\right) \tag{3}$$

B is the constant for the thermistor. T_1 and R_1 are reference temperatures and resistance. All these values were given in the formula sheet. B is 3950, and R_1 is 50kΩ at 25 °C. T_2 and R_2 are the resistance and corresponding temperature. Since the ESP would measure R2, this formula could be re-arranged to get T_2. The temperature values must be in Kelvin when calculating the formula, so to find T_2 in degrees Celsius, the formula becomes:

$$T_2 = \frac{B}{\frac{B}{T_1 + 273.15} - \ln\left(\frac{R_1}{R_2}\right)} - 273.15 \tag{4}$$

The YF-B7 has two wires for the thermistor. These have no polarity. The resistor chosen to be in series with the thermistor was 51kΩ, as this was very close to the reference resistance, allowing measurements to be more accurate. The power to this was supplied via GPIO23.

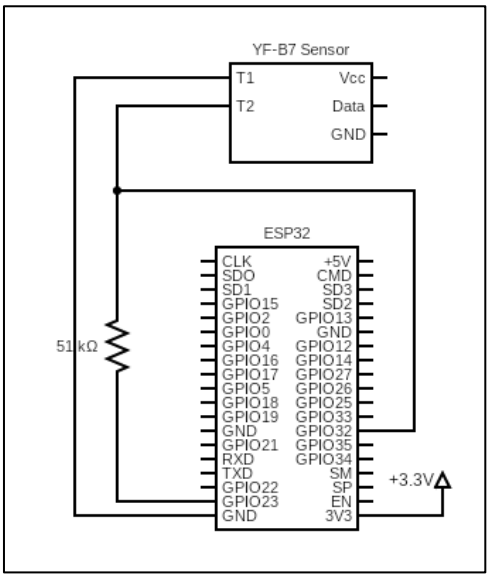


Figure 4.14 - Wiring diagram for the thermistor

The regular 3.3V pin was not used because constantly applying a current through the thermistor would cause it to heat up, causing the measurements to be slightly inaccurate. By using a regular GPIO pin, it can be toggled on only when a measurement has to be taken. The voltage was read using the “Analog to Digital (ADC)” converter on GPIO32.

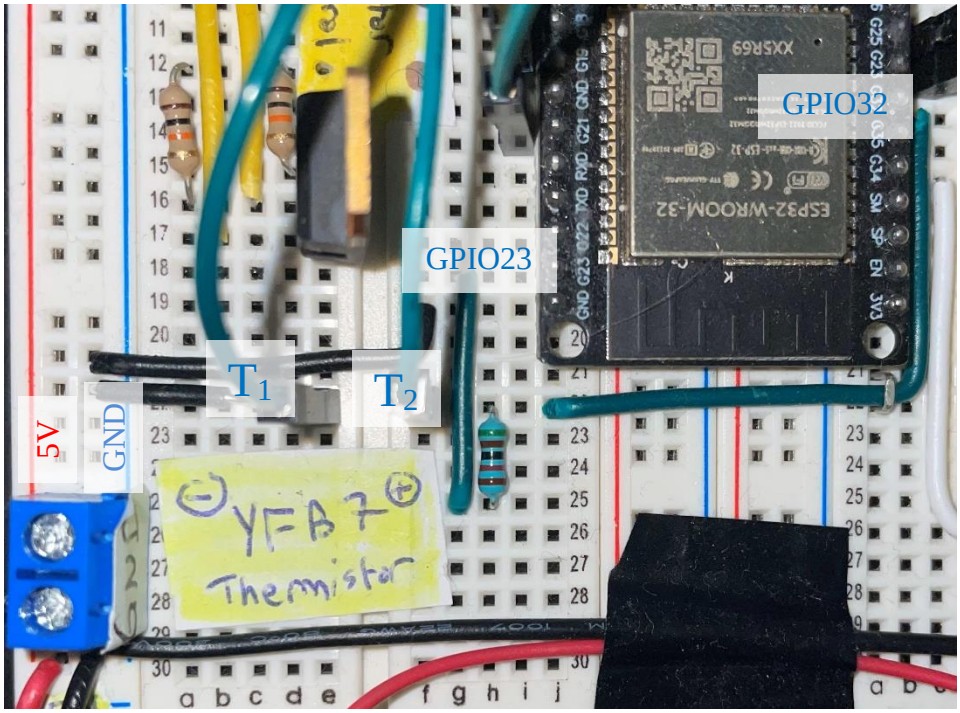


Figure 4.15 - Thermistor wiring

To implement the software, firstly, GPIO23 and an “interval” component were declared. The interval activates every 30 seconds and runs a short list of commands. When activated, GPIO23 is set high, meaning a voltage is applied across the thermistor. The command then prompts the thermistor sensor to update its value. Once updated, the voltage across the thermistor is turned off.

```
switch:
  ##############################################
  ## THERMISTOR SWITCH - PROMPT TO READ TEMP ##
  ##############################################
  # Keeping the pin that thermistor is connected to powered at all times,
  ## can cause the temperature reading to be slightly off, so only power it
  ### at short intervals, and read the temperature then
  - platform: gpio
    pin: GPIO23
    id: thermistor_voltage

interval:
  - interval: 30s
    then:
      - switch.turn_on: thermistor_voltage
      - component.update: thermistor_ref_sensor
      - switch.turn_off: thermistor_voltage
```

Figure 4.16 - Declaring GPIO23 as a usable pin and the "interval" component.

To read the actual voltage and turn it into a resistor value, firstly, the “adc” (Analog to Digital Converter) component is used to read the analog value on GPIO32 and turn it into a digital value, which the ESP32 can work with. The “resistance” component automatically turns that reading into a resistance. This component can be configured to work with a voltage divider automatically. The resistor that was placed in series is configured in this component. A reference voltage is also passed into it. 3.27V is used as this was the measured value on GPIO32 when it was set to high. The voltage divider configuration is declared using the ”configuration” option. The “DOWNSTREAM” option was passed into this, as this implies that the variable resistor (thermistor) is down past where the voltage is being read (i.e. it is the second resistor in the two-resistor voltage divider). Finally, using the “ntc” component of ESPHome, the thermistor resistance can be used to calculate the temperature. This required the value of the “b constant”, as well as the reference temperature and resistance from the datasheet. Once complete, the “Water Temperature” sensor value will become available.

```
1  ##############################
2  ## YFB7 THERMISTOR SENSOR  ##
3  ##############################
4  - platform: ntc
5    sensor: resistance_sensor
6    calibration:
7      b_constant: 3950
8      reference_temperature: 25°C
9      reference_resistance: 50kOhm
10   id: supply_water_temperature_sensor
11   name: Water Temperature
12
13 - platform: resistance
14   id: resistance_sensor
15   sensor: thermistor_ref_sensor
16   configuration: DOWNSTREAM
17   resistor: 51kOhm
18   reference_voltage: 3.27
19
20 - platform: adc
21   id: thermistor_ref_sensor
22   pin: GPIO32
23   attenuation: auto
24   update_interval: never
```

Figure 4.17 - ESPHome configuration for the Thermistor

4.5. Flow Control and Mechanical Design

4.5.1. Shutoff Valve

Some existing consumer products, such as Figure 4.18 below, can be used to control the flow of water through the sensor and pipe. However, these can only be fully open or fully closed. They also come at a high price when bought from recognised resellers (the valve below costs €665.79 without VAT).

Figure 4.18 - Motorised ball valve from RS [65]

Therefore, it was decided to purchase a much more affordable manual ball valve and construct a casing connected to a motor. The casing would be constructed using 3-D printing, and a rotary encoder would also be used for position feedback.

The chosen ball valve is seen below. This was preferred as the cap used to turn the valve points upwards and has a larger surface area, so the casing that would be designed could be slid right on, and the force could be distributed evenly to either side of the cap when turning.

Figure 4.19 - Chosen Ball Valve

4.5.2. Motor Control

The next step of the valve control was to select a motor and successfully control it with the ESP32. Two readily available options were a 28BYJ-48 Stepper motor and an Astrosyn MY355 Stepper motor.

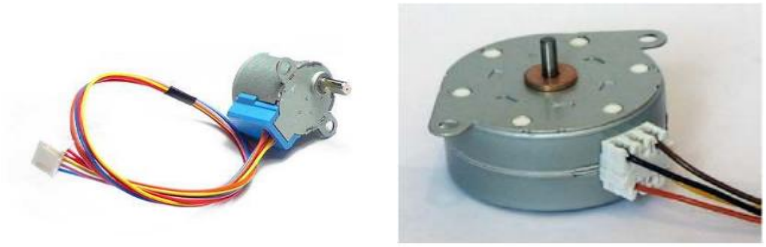


Figure 4.20 - 28BYJ-48 (Left) and Astrosyn MY355 (Right)

The advantage of the 28BYJ-48 stepper motor is that it is powered by 5V, so it could be powered by the same power source as the ESP32 (which is also powered by 5V via a USB-C or the 5V pin). This motor also came with its own motor controller chip, which has a component built into ESPHome. The downside is that this motor has a relatively low torque output, so it would need a really high gear ratio to *possibly* be able to turn the ball valve. The MY355 is powered via 12V and provides a much higher torque output. Despite requiring more setup work, the motor chosen was the MY355, which would provide much more torque and more reliable control of the ball valve.

The MY355 is a permanent magnet stepper motor. These consist of two key parts: stators and the rotor. It usually consists of multiple stators, which cause the rotor to turn when a current is passed through individual stators in a specific sequence [66]. The number of wires going into the motor depends on the number of stators present. For the MY355, there are four wires, and the sequence for powering these to get a rotation is given in the datasheet.

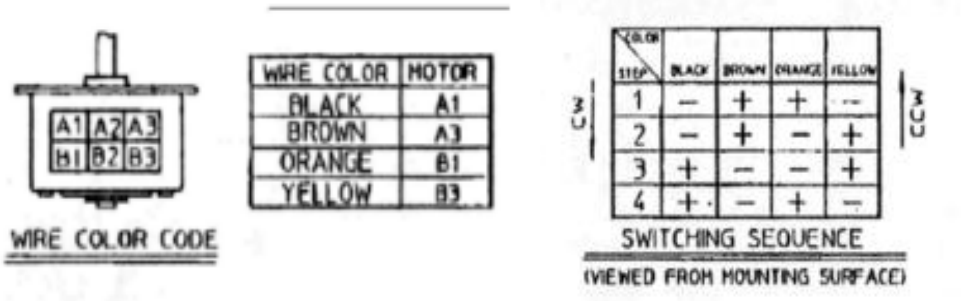


Figure 4.21 - Switching sequence for Clockwise and Anticlockwise rotation of MY355

The first challenge for controlling this motor is that the ESP32 digital outputs are 3.3V or 0V, but the motor requires 12V (when the table shows "+") and 0V (when the table shows "-"). Since no motor controller chip was readily available, this motor was controlled using a dual H-Bridge. This was used instead of mechanical relay switches because the electronic switches can handle a much higher switching frequency. An H-Bridge is a circuit that allows voltage to be applied across a load in either direction. It is typically used to power DC and stepper motors and make them rotate in either direction [67]. In this project, the L293D H-Bridge from Texas Instruments was used, allowing four outputs to be controlled, which is the same as the amount of wires in the MY355.

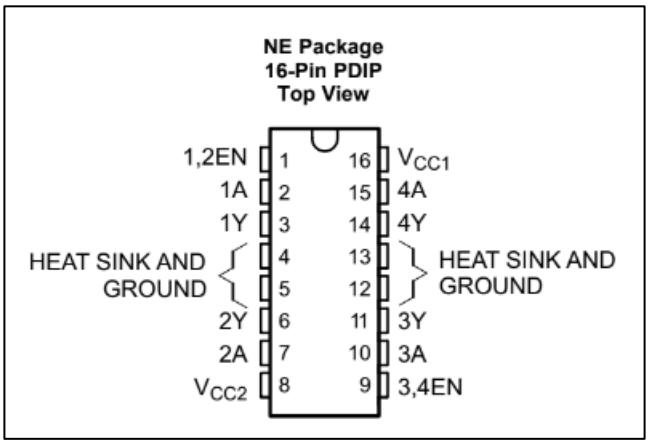


Figure 4.22 - Pin configuration of L293D [68]

The L293D consists of four input pins, four output pins, 2 enable pins (which must be turned on for power to be supplied to the motor), a 5V pin for powering the chip, and a pin for the voltage that will be applied to the stepper motor. Y represents the outputs, and U the inputs. Hence, input 1A controls output 1Y. The input pins were connected to the ESP32, and the corresponding outputs were connected to the motor wires. There are two enable pins on the L293D in case only one-half of the outputs are used. Since all outputs were used in this scenario, the enable pins were connected together so a single GPIO pin from the ESP could control them. The wire colours were kept consistent for easy debugging and understanding of the circuit. For example, the orange cable from the stepper motor was connected to the L293D pin 1Y, and then the wire going from 1Y to 1A and then to the ESP was orange. Two status LEDs were also included, one for the 12V supply and one for the motor enable pins. This allowed for visual debugging in case of any problems with the ESP or power supply. The circuit diagram and actual circuit can be seen below.

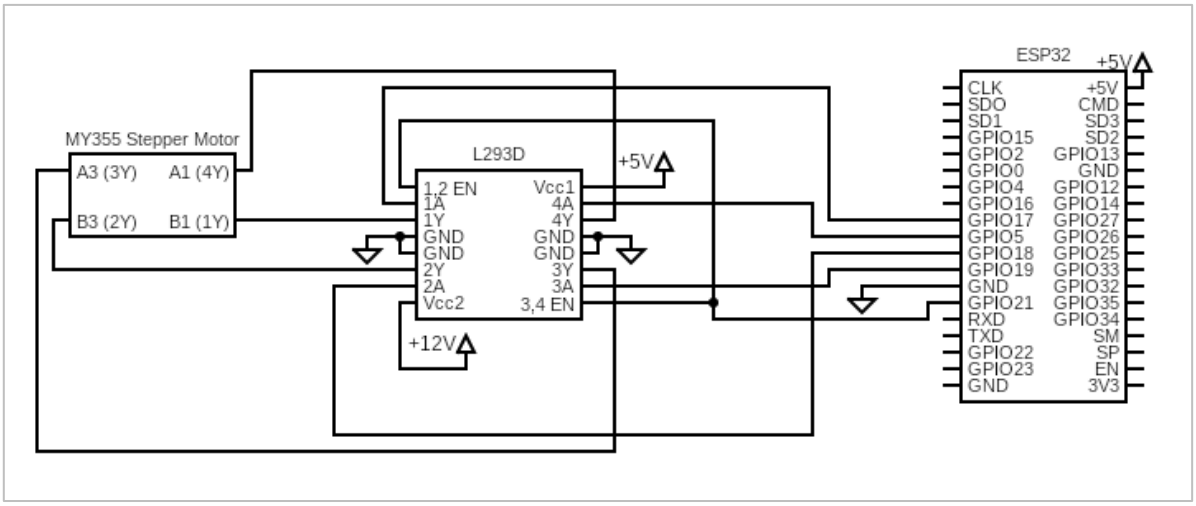


Figure 4.24 - Circuit Diagram for Motor Control

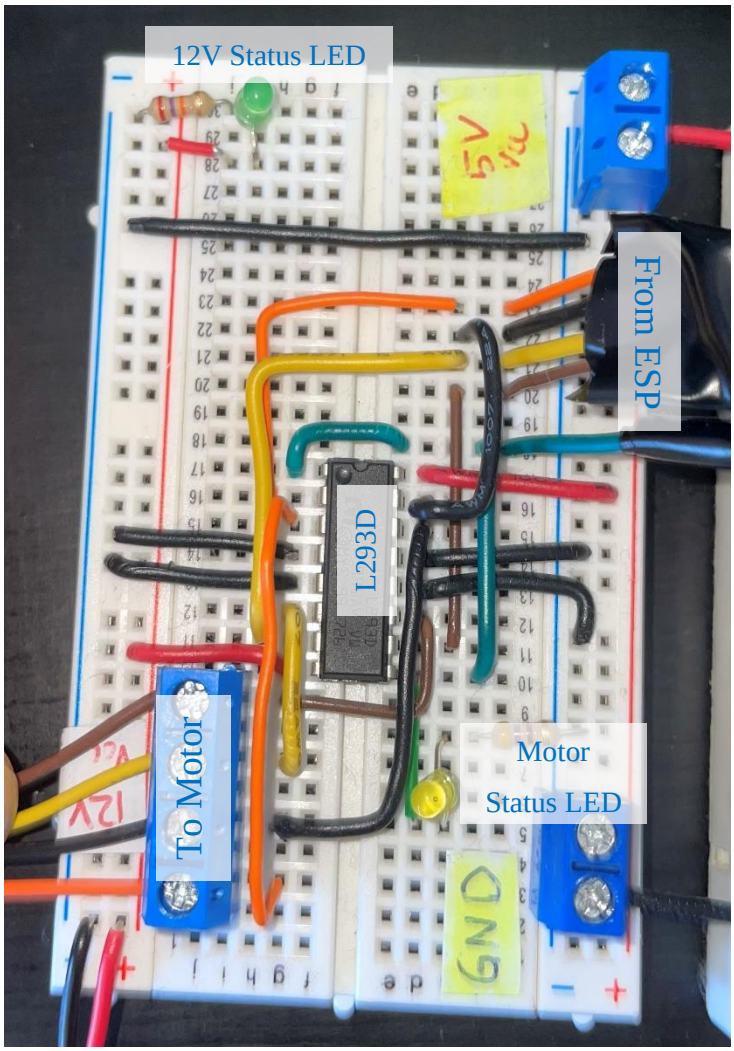


Figure 4.23 - Circuit Setup for Motor Control - note wire colours

Once the wiring was complete, the ESP had to be programmed to turn the motor clockwise and anti-clockwise. Firstly, the five pins (four for the motor and one for the enable) were initialised. These were named using the colour of the wire going into the pin (Figure 4.26).

```yaml
##################################
## STEPPER MOTOR DRIVER OUTPUTS ##
##################################
output:
  # ENABLE PIN FOR MOTOR COILS
  - platform: gpio
    pin: GPIO21
    id: stepper_enable_pin

  # PIN 1A on L293D H-Bridge
  # CONTROLS 1Y
  - platform: gpio
    pin: GPIO17
    id: stepper_orange_cable

  # PIN 3A on L293D H-Bridge
  # CONTROLS 3Y
  - platform: gpio
    pin: GPIO5
    id: stepper_black_cable

  # PIN 2A on L293D H-Bridge
  # CONTROLS 2Y
  - platform: gpio
    pin: GPIO18
    id: stepper_yellow_cable

  # PIN 4A on L293D H-Bridge
  # CONTROLS 4Y
  - platform: gpio
    pin: GPIO19
    id: stepper_brown_cable
```

Figure 4.26 - Initialise GPIO pins

```yaml
script:
  ##############################################
  ###### PM STEPPER MOTOR SEQUENCE OUTPUTS ######
  ##############################################
  - id: sequence_reset
    then:
      - output.turn_off: stepper_enable_pin
      - output.turn_off: stepper_black_cable
      - output.turn_off: stepper_brown_cable
      - output.turn_off: stepper_orange_cable
      - output.turn_off: stepper_yellow_cable

  ### Sequences taken from datasheet of stepper motor
  ### CW       = 1 -> 2 -> 3 -> 4
  ### Anti-CW  = 4 -> 3 -> 2 -> 1
  - id: sequence_row_1
    then:
      - output.turn_off: stepper_black_cable
      - output.turn_on: stepper_brown_cable
      - output.turn_on: stepper_orange_cable
      - output.turn_off: stepper_yellow_cable

  - id: sequence_row_2
    then:
      - output.turn_off: stepper_black_cable
      - output.turn_on: stepper_brown_cable
      - output.turn_off: stepper_orange_cable
      - output.turn_on: stepper_yellow_cable

  - id: sequence_row_3
    then:
      - output.turn_on: stepper_black_cable
      - output.turn_off: stepper_brown_cable
      - output.turn_off: stepper_orange_cable
      - output.turn_on: stepper_yellow_cable

  - id: sequence_row_4
    then:
      - output.turn_on: stepper_black_cable
      - output.turn_off: stepper_brown_cable
      - output.turn_on: stepper_orange_cable
      - output.turn_off: stepper_yellow_cable
```

Figure 4.25 - Motor pins sequences scripts

As seen in Figure 4.21, to rotate the motor, the pins must be turned on or off in a specific configuration. These sequences were implemented using the “scripts” option in ESPHome, so each row can easily be called from anywhere in the file. A “Reset” script was also set up, which disables the “Enable” pin and turns all outputs off.

A slight delay between each sequence was needed to prevent the motor from rotating too fast or causing the ESP to switch sequences faster than the motor could turn. A global variable was created in the file to remove the need for hardcoding this each time it was needed. This was set to 10ms.

```yaml
globals:
  - id: stepper_delay
    # The delay between execution of each step on stepper motor
    type: int
    initial_value: "10"
```

Figure 4.27 - Stepper delay variable

Before implementing the scripts to rotate the motor, it was noted that when the motor rotated the actual ball valve, it needed to know when it reached the fully opened or closed position. To implement this, two end-stop switches were connected to the ESP. These would trigger that the start/end position has been reached.

Figure 4.28 - End-stop switch used

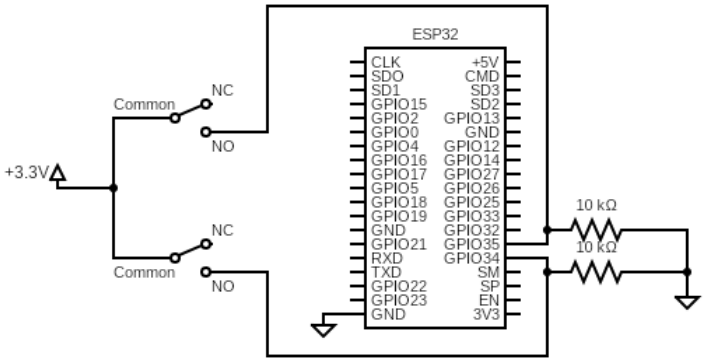


Figure 4.29 - Wiring diagram for end-stop switches

These switches contain three pins: normally open (NO), normally closed (NC), and common. The common pin was connected to +3.3V, and the NO pin was connected to one of the pins on the ESP32 via a pull-down resistor (to reduce noise and false triggers).

These were then implemented into the software using the “binary_sensor.gpio” component.

```yaml
binary_sensor:
  - platform: gpio
    pin: GPIO35
    name: "Valve Open Limit Switch"
    id: valve_open_limit_switch
  - platform: gpio
    pin: GPIO34
    name: "Valve Closed Limit Switch"
    id: valve_closed_limit_switch
```

Figure 4.30 - End switches software implementation

The scripts to open or close the valve (i.e. to rotate the motor clockwise or anticlockwise) could now be implemented. These scripts used a “while” condition command, meaning a block inside the script only ran while the condition was true. In this case, the condition would be whether or not the switch is off. So, for closing the valve, it is checked that the “valve_closed_limit_switch” is off, and if so, the script to rotate the motor in that direction will run until the switch is triggered.

```yaml
##################################
### OPEN / CLOSE VALVE SCRIPTS ###
##################################
- id: open_water_valve
  then:
    - output.turn_on: stepper_enable_pin
    - while:
        condition:
          #### ROTATE ANTICLOCKWISE TO CLOSE THE VALVE ####
          ### IF valve_open_limit_switch IS ON, THIS WON'T RUN ###
          - binary_sensor.is_off: valve_open_limit_switch
        then:
          - script.execute: sequence_row_4
          - delay: !lambda |-
              return id(stepper_delay);
          - script.execute: sequence_row_3
          - delay: !lambda |-
              return id(stepper_delay);
          - script.execute: sequence_row_2
          - delay: !lambda |-
              return id(stepper_delay);
          - script.execute: sequence_row_1
          - delay: !lambda |-
              return id(stepper_delay);
    - output.turn_off: stepper_enable_pin

```

Figure 4.31 - Script to open water valve (rotate motor anticlockwise)

```
26
27  - id: close_water_valve
28    then:
29      - output.turn_on: stepper_enable_pin
30      - while:
31          condition:
32            #### ROTATE CLOCKWISE TO CLOSE THE VALVE ####
33            ### IF valve_closed_limit_switch IS ON, THIS WON'T RUN ###
34            - binary_sensor.is_off: valve_closed_limit_switch
35          then:
36            - script.execute: sequence_row_1
37            - delay: !lambda |-
38                return id(stepper_delay);
39            - script.execute: sequence_row_2
40            - delay: !lambda |-
41                return id(stepper_delay);
42            - script.execute: sequence_row_3
43            - delay: !lambda |-
44                return id(stepper_delay);
45            - script.execute: sequence_row_4
46            - delay: !lambda |-
47                return id(stepper_delay);
48      - output.turn_off: stepper_enable_pin
```

Figure 4.32 - Script to close water valve (rotate motor clockwise)

4.5.3. Parts for Turning the Ball Valve using the Motor

SolidWorks was used as the CAD (computer-aided design) software to design the mechanical aspects of this project. Firstly, to design parts that would accurately fit the system, the existing items, such as the ball valve, motor, flow sensor, etc., were designed in SolidWorks by taking accurate measurements and using any measurements on the data sheets. Firstly, a small gear was made and placed on the motor's shaft. This gear was given ten teeth and a module of 1.5.

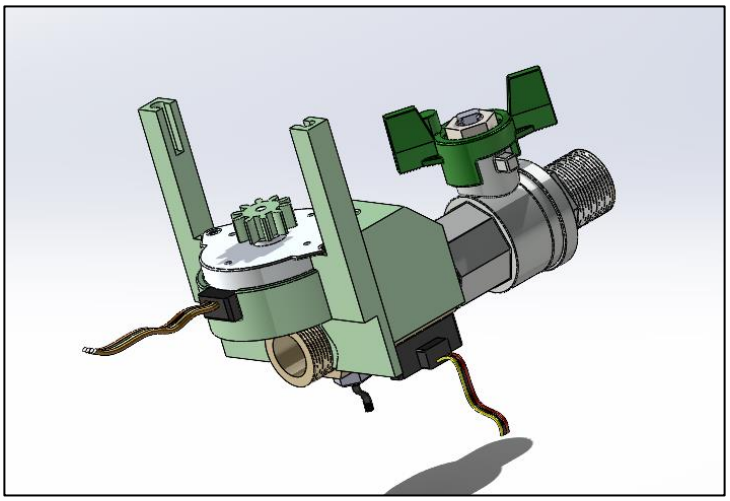


Figure 4.33 - Flow sensor, ball valve, motor, gear and body to hold the motor

Next, the body that could hold the motor in place was designed. A round opening, with a thin wall, was made, and a small opening had to be left in the wall for the small rubber body that is part of the motor (at the point where wires are attached to it). This worked out favourably as the rubber body would cause an opposing force if the motor slipped and rotated itself. The underside of this body had to be accurately designed as it needed to fit tightly onto the flow sensor, without slipping. In reality, the flow sensor would be connected to a pipe via its threaded end, fixing the body fully into position. The two arms on either side of the motor were used to hold a roof component, which will hold the two end switches. This roof will also contain a small opening for the rotary encoder (used for position feedback, which will be explained thoroughly in the next section). In Figure 4.33, behind the first arm is a small hole in the body holding the motor. This was used to hold a pin, around which an intermediary gear rotated. This is used because the distance from the motor to the ball valve is relatively large, so having just two gears would cause them to be too wide. It also helps to increase the torque further and reduce the speed. Speed is not crucial to this implementation, so reducing this to increase torque is a fair trade-off. This gear was designed as a compound gear. The motor gear rotated a large gear with 30 teeth, and a small gear on top of it rotated the ball valve. This allowed for a large increase in torque without taking up too much space. Both gears had a module of 1.5.

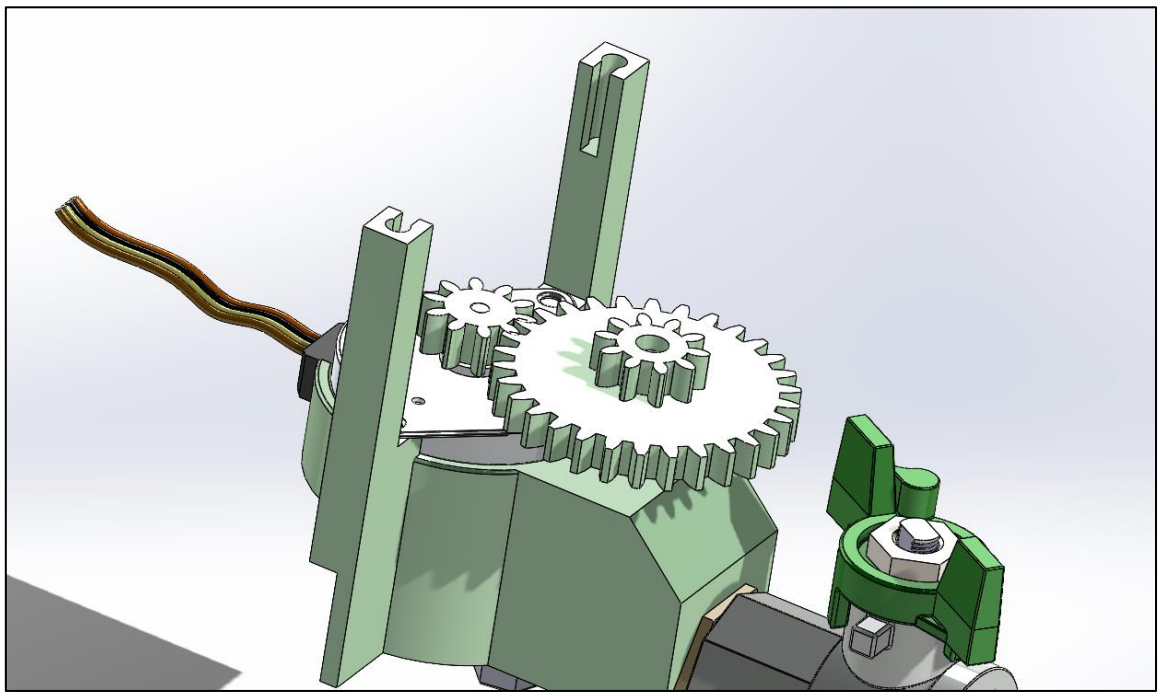


Figure 4.34 - Intermediary compound gear coupled with motor gear

Next, the cap for the ball valve was designed. Firstly, it was designed to slide onto the green twisting cap and stay firm. It also required a gear component which could couple with the small part of the intermediary compound gear. To calculate the teeth on the gear component, the distance between the intermediary gear's reference diameter and the green cap's centre was measured. This was found to be around 42mm. This would be the radius of the gear, so the value 84mm was used. The formula below was then used to find that the gear would need to have 56 teeth.

$$reference\ diamater = (teeth) * (module) => 84 = N * 1.5 \quad (5)$$

This means that the torque from the motor is increased by 16.8 times and applied to the ball valve:

$$Torque_{out} = Torque_{in} * \frac{N_2}{N_1} * \frac{N_4}{N_3} = T_{in} * \frac{30}{10} * \frac{56}{10} = T_{in} * 16.8 \quad (6)$$

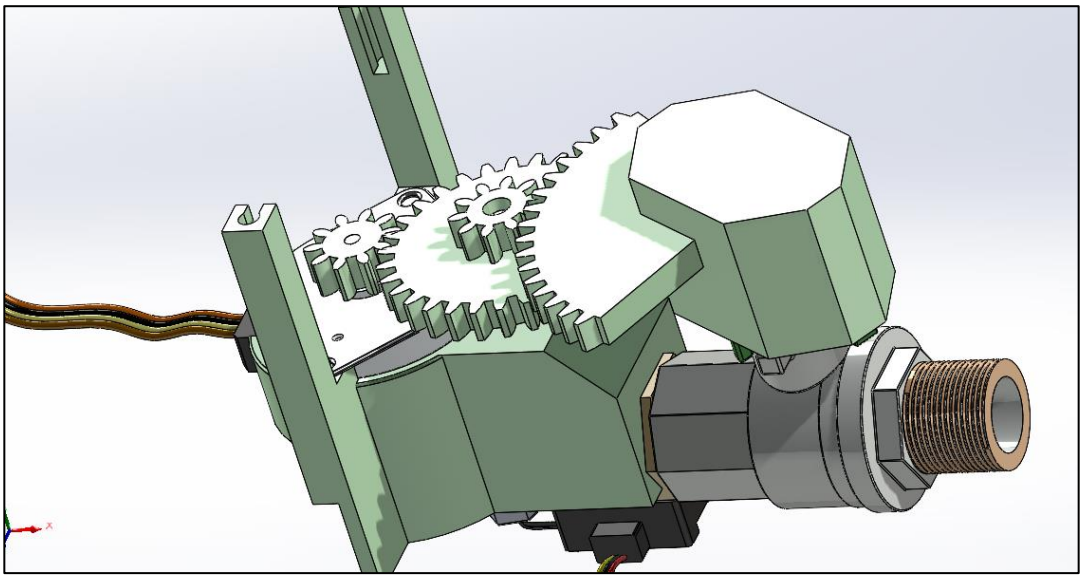


Figure 4.35 - Gear coupling from motor to ball valve

Since the ball valve only required a quarter-turn to open, a large section of the gear on the cap was removed, leaving only the usable section. This saved on materials and space.

From here, the duration it would take for the all valve to go from fully open to closed can be estimated. Since the final gear requires a quarter turn, $\frac{56}{4} = 14$ teeth must be moved. This means the intermediary gear must make 1.4 rotations ($\frac{14\ teeth}{10\ teeth}$), and hence the motor gear

must make $\frac{30*1.4}{10} = 4.2$ rotations. The data sheet of the motor mentions that a full step of the motor is 7.5°, which means the number of steps in 4.2 rotations is:

$$\frac{360}{7.5} * 4.2 \approx 202\ steps \tag{7}$$

Since a delay of 10ms is added between steps, the full opening/closing should take around 2.02 seconds. This is well within what would be desired for such a system.

The final part to be attached to the system was the rotary encoder. This encoder contains a shaft, which would be rotated by another gear. To simplify the process, the centre of the encoder was placed directly above the centre of the motor. A gear with 30 teeth was connected to this since it could couple directly with the small part of the compound gear. The chosen encoder does 20 steps per rotation. Using similar calculations as above, it was estimated that the gear connected to the encoder would do $\frac{10*1.4}{30} = 0.4667$ rotations, which equates to 9 positions on the rotary encoder. If position 0 is the starting point, then position 8 is the final one. Hence, every step in between means the ball valve has turned an additional 11.25°.

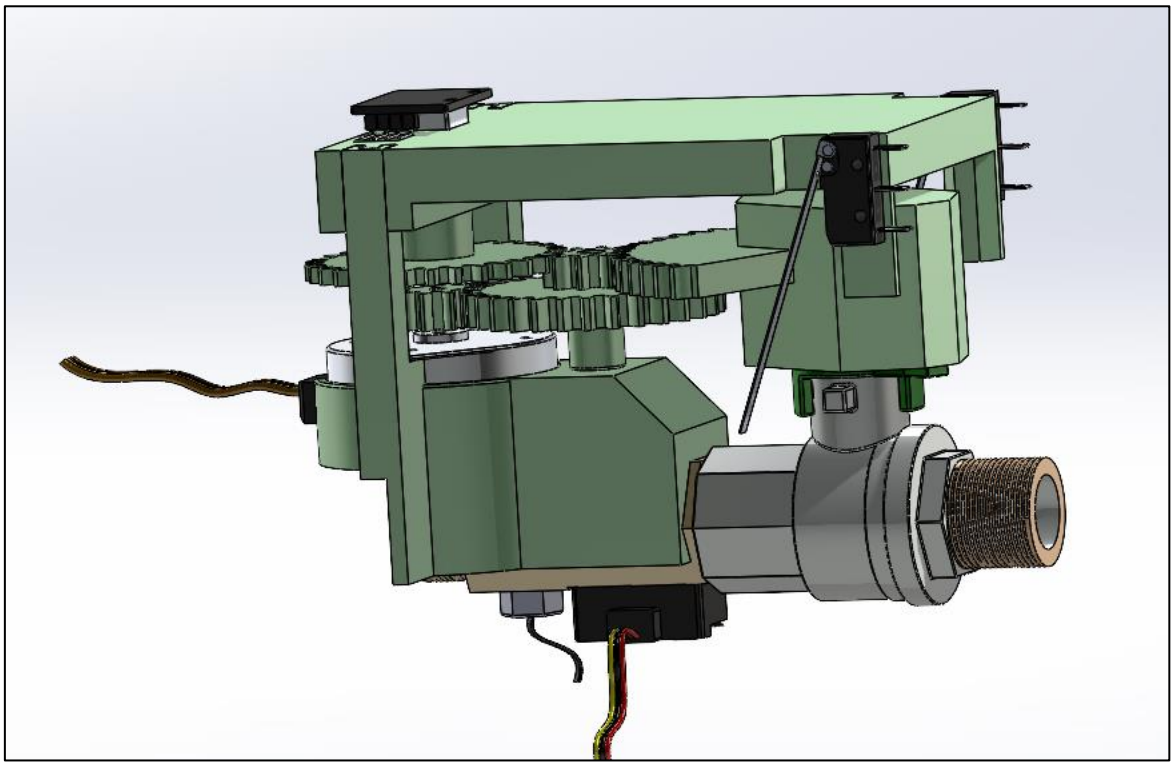


Figure 4.36 - Final SolidWorks assembly

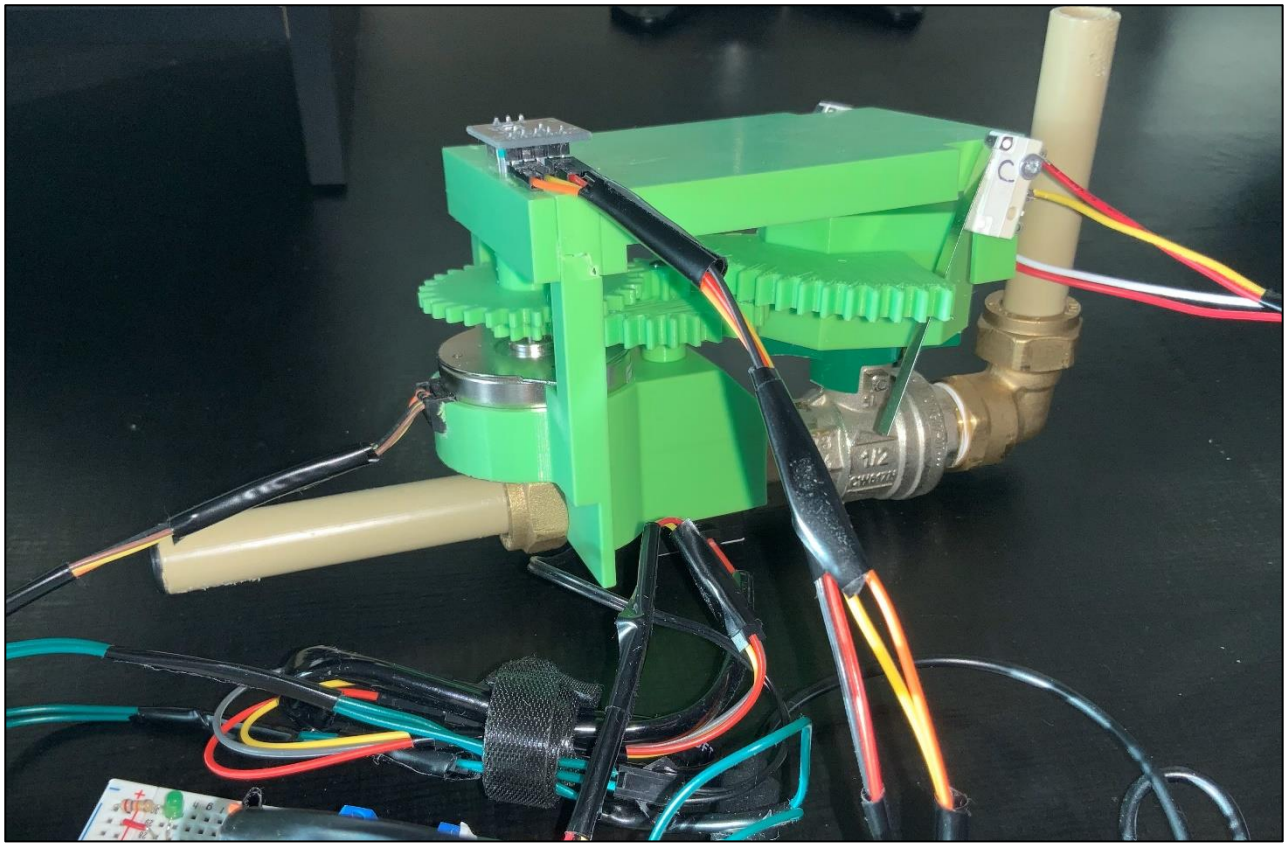


Figure 4.37 - Actual assembled mechanical system

The drawings for the parts designed in this can be seen in the Appendices section.

4.5.4. Rotary Encoder - Position Feedback and Partial Control

The final step of implementing partial rotations of the ball valve required positional feedback. Although a counter that was increased/decreased as the stepper motor moved could have been implemented, this would not have been fully accurate because the motor may miss a step, so the counter would have been incremented, but the motor wouldn't have rotated.

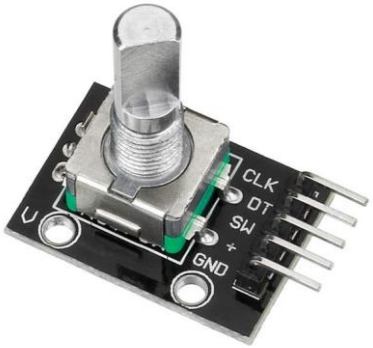


Figure 4.38 - Rotary encoder used in this project

The rotary encoder was more accurate due to how it works. The selected encoder contains three usable pins. CLK, DT and SW. SW is a built-in push button switch not implemented in this project. The two other pins each create a square wave pulse as the encoder rotates. These pulses are 90° out of phase, and the order in which the rising edges occur on these two pins determines the direction in which the encoder is rotating [69]. This encoder was connected to the ESP as follows:

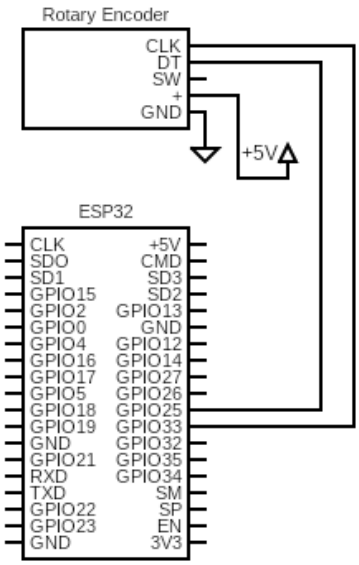


Figure 4.39 - Rotary Encoder wiring diagram

The rotary encoder is built into ESPHome and can easily be initialised in the codebase as follows:

```
sensor:
  ##################################################
  ## ROTARY ENCODER - FOR KEEPING TRACK OF POSITION ##
  ##################################################
  - platform: rotary_encoder
    name: "Rotary Encoder"
    id: ball_valve_rotary_encoder
    min_value: 0
    publish_initial_value: true
    pin_a:
      number: GPIO33
      inverted: true
      mode:
        input: true
        pullup: true
    pin_b:
      number: GPIO25
      inverted: true
      mode:
        input: true
        pullup: true
```

Figure 4.40 - Rotary encoder ESPHome setup

Next, a global variable was declared, which would be the maximum position the encoder could reach. This was used in some calculations (explained below), and declaring it like this saves having to hard-code it in multiple places. This was set as the integer 8, which was calculated in section 4.5.3.

```yaml
- id: max_rotary_encoder_position
  # Based on testing and calculations, max value is 8 when valve closes
  # IF CHANGING THIS, UPDATE water_valve_position
  type: int
  initial_value: "8"
```

Figure 4.41 - Max rotary encoder position

A short script was also created to reset the value of the rotary encoder. It was decided that the encoder would be in position 0 when the valve was fully open. Thus, if any errors occurred or the encoder missed any steps, activating the "open" switch resulted in the encoder being reset.

```yaml
############################
## ROTARY ENCODER COUNTER ##
############################
- id: reset_rotary_encoder_count
  then:
    - sensor.rotary_encoder.set_value:
        id: ball_valve_rotary_encoder
        value: 0
```

Figure 4.42 - Reset rotary encoder counter script

The next step was implementing the ability to move the ball valve to a position anywhere between fully open and fully closed. Firstly, a “number” component was set up. This would be between 0 and 100, indicating what percent the valve is open. (0 means the valve is closed, 100 means the valve is open). The step was set at 12.5 since the maximum value of the encoder was 8 (100/8 = 12.5). This was set up as such because it would be easier for a user to understand that the valve is “X% open” rather than just displaying the position as a number between 0 and 8.

```yaml
number:
  ################################################
  ## SET THE POSITION THE VALVE SHOULD MOVE TO ##
  ################################################
  - platform: template
    id: water_valve_position
    name: "Water Valve Position"
    min_value: 0
    max_value: 100
    # 100 / max_rotary_encoder_position
    step: 12.5
    unit_of_measurement: "%"
    optimistic: true
```

Figure 4.43 - Initialising the variable which will provide positional feedback

This number was then used in a script to complete the movement to the desired position. Before that, though, the code for the two end-stop switches was updated to set this number to the correct value when activated. This was done using the "on_press" configuration of the "binary_sensor" component:

```yaml
####################################
## VALVE OPEN/CLOSED LIMIT SWITCHES ##
####################################
binary_sensor:
  - platform: gpio
    pin: GPIO35
    name: "Valve Open Limit Switch"
    id: valve_open_limit_switch
  - platform: gpio
    pin: GPIO34
    name: "Valve Closed Limit Switch"
    id: valve_closed_limit_switch
    on_press:
      then:
        - script.execute: reset_rotary_encoder_count
        - number.set:
            id: water_valve_position
            value: 100
        - component.update: water_valve_position
        - script.execute: stop_valve_rotation
```

Figure 4.44 - Full code for "fully-open" limit switch

```yaml
  - platform: gpio
    pin: GPIO34
    name: "Valve Closed Limit Switch"
    id: valve_closed_limit_switch
    on_press:
      then:
        - number.set:
            id: water_valve_position
            value: 0
        - component.update: water_valve_position
        - script.execute: stop_valve_rotation
```

Figure 4.45 - Full code for "fully-closed" limit switch

The number was updated using the "number.set" command, and then the "component.update" command was called to update it not just on the backend but anywhere that it is visible (such as the Home Assistant Dashboard). Also, note the execution of the scripts to stop the valve rotation and reset the rotary encoder when the "fully open" switch was pressed.

The script to stop the valve rotation is relatively simple. It disables the H-bridge's enable pin, stopping the motor from functioning.

```yaml
- id: stop_valve_rotation
  then:
    - output.turn_off: stepper_enable_pin
```

Figure 4.46 - Stop valve rotation script

The “water_valve_position” variable does not yet move the valve to the desired position. Initially, the aim was to make the valve move when the number was set. However, some problems occurred by updating the value of the number and moving the valve simultaneously. So a separate button was implemented that, when pressed, executes the move to the desired position. This simple software button is visible in Home Assistant and the devices portal page. When pressed, the button executes the “move_valve_to_positon” script, which takes a value between 0 and 8 (0 being fully open, 8 being fully closed). These numbers are the rotary encoder values. However, since the “water_valve_positon” is a percentage value between 0 and 100, a “lambda” function was used to convert it accordingly.

```yaml
button:
  ########################################
  ### BUTTON TO EXECUTE MOVE TO POSITION ###
  ########################################
  - platform: template
    id: execute_move_to_position
    name: "Execute Move to Position"
    on_press:
      - script.execute:
          id: move_valve_to_position
          # convert the percentage value back to a integer to know when the rotary encoder needs to stop
          # subtracting from maximum, because in the water_valve_position number, 100% => open, but open => rotary_position is 0
          required_position: !lambda return id(max_rotary_encoder_position) - (id(water_valve_position).state * id(max_rotary_encoder_position) / 100);
      # Logging
      - lambda: |-
          int position = id(max_rotary_encoder_position) - (id(water_valve_position).state * id(max_rotary_encoder_position) / 100);
          ESP_LOGD("custom", "Water Valve position set to %d. Open = 0, Closed = %d", position, id(max_rotary_encoder_position));
```

Figure 4.47 - Software button to execute “move_valve_to_position”

The problem with the percentage values was that 100% means the valve is fully open, however the rotary encoder was initialised in such a way that position 0 meant fully open. Therefore the “lamba” function would need to convert 100% to 0 and 0% to 8. This function was implemented using the following equation:

$$position = max\_encoder\_position - desired\_percent\_open * \frac{\max\ \_encoder\_value}{100} \quad (8)$$

This position was then passed into the script “move_valve_to_position”, which takes a single parameter, the desired position of the rotary encoder.

```yaml
- id: move_valve_to_position
    parameters:
      required_position: int
    then:
      - logger.log:
          format: "Moving to position - %d. Current Position - %d."
          args: ["required_position", "id(ball_valve_rotary_encoder).state"]

      - if:
          condition:
            # If we're told to fully open or close, just execute regular scripts which push until trip switch is hit
            - lambda: return required_position == 0 || required_position == id(max_rotary_encoder_position);
          then:
            - if:
                condition:
                  - lambda: return required_position == 0;
                then:
                  - logger.log: "OPENING VALVE FULLY"
                  - script.execute: open_water_valve
                else:
                  - logger.log: "CLOSING VALVE FULLY"
                  - script.execute: close_water_valve

          # If we're looking to move to a position in the range 1 -> (max-1), then execute this
          else:
            # BALL_VALVE_ROTARY_ENCODER = Max => Valve Closed
            # BALL_VALVE_ROTARY_ENCODER = 0 => Valve Open
            - while:
                condition:
                  # While not at the required position
                  - lambda: return required_position != id(ball_valve_rotary_encoder).state;
                then:
                  # ENABLE MOTOR IN CASE IT TURNS OFF
                  - output.turn_on: stepper_enable_pin

                  - if:
                      condition:
                        - lambda: return required_position < id(ball_valve_rotary_encoder).state;
                      then:
                        # GO OPEN VALVE DIRECTION => ANTICLOCKWISE
                        - script.execute: sequence_row_4
                        - delay: !lambda |-
                            return id(stepper_delay);
                        - script.execute: sequence_row_3
                        - delay: !lambda |-
                            return id(stepper_delay);
                        - script.execute: sequence_row_2
                        - delay: !lambda |-
                            return id(stepper_delay);
                        - script.execute: sequence_row_1
                        - delay: !lambda |-
                            return id(stepper_delay);

                  - if:
                      condition:
                        - lambda: return required_position > id(ball_valve_rotary_encoder).state;
                      then:
                        # GO CLOSE VALVE DIRECTION => CLOCKWISE
                        - script.execute: sequence_row_1
                        - delay: !lambda |-
                            return id(stepper_delay);
                        - script.execute: sequence_row_2
                        - delay: !lambda |-
                            return id(stepper_delay);
                        - script.execute: sequence_row_3
                        - delay: !lambda |-
                            return id(stepper_delay);
                        - script.execute: sequence_row_4
                        - delay: !lambda |-
                            return id(stepper_delay);

                  # DISABLE MOTOR ONCE POSITION REACHED
                  - output.turn_off: stepper_enable_pin
```

Figure 4.48 - ESPHome script that moves the ball valve to desired position

This script consists of three main sections. The first section checks if the desired position is equal to zero or if it equals the maximum rotary encoder position global variable. If it is, it executes one of: “open_water_valve” or “close_water_valve” scripts, which were explained earlier. If the value is between 1 and 7 inclusive, then it checks whether the current position is greater than or less than the desired position and rotates the motor accordingly.

4.6. Power Supply

For this project, a 12V source and a 5V source were needed. An AC-DC adapter was purchased, which converts the 230V AC from mains power to 12V DC, ideal for this project. A buck-converter was also purchased. This is an affordable and efficient step-down converter [70]. This model takes in an input of 12-24V and will output 5V at a maximum of 5A.

The Buck Converter purchased came with a terminal block on the 12V and 5V sides, making it easy to connect to the circuit. It also contained a USB-A slot, which was used to power the ESP32. This setup ensured a safe connection to mains power and an efficient supply of both 12V and 5V to the circuit.

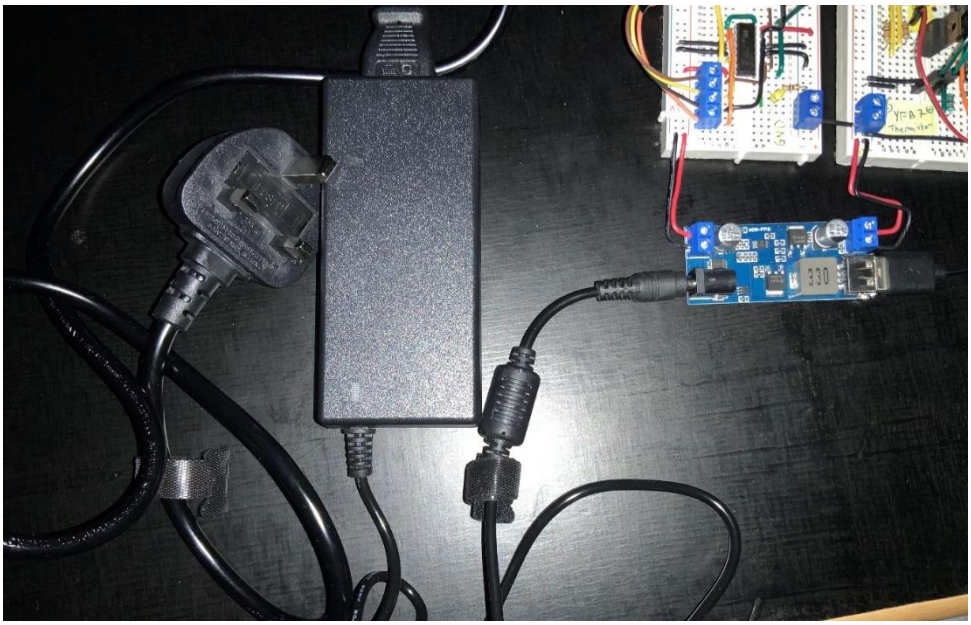


Figure 4.49 - AC-DC adapter connected to the Buck Converter

4.7. Physical Display and Control

Next, physical control and monitoring were implemented. Although seeing all the data on an online dashboard and controlling the device automatically or via a smartphone is nice, these will not always work. Sometimes, there will be problems with the network or the power, and

being able to view some data and control the device physically is essential. The first thing implemented for this was a physical button. This was implemented the same as the trip switches; the only difference was that a “filters” configuration was added, which ensures the button was pressed at least 10ms before the ESP acknowledges that the button was pressed. This is to remove errors due to noise when pressing the button. When pressed, the button will open or close the valve, depending on its position.

```
########################################
## PHYSICAL BUTTON TO OPEN/CLOSE VALVE ##
########################################
- platform: gpio
  pin:
    number: GPIO22
    inverted: True
    mode:
      input: True
      pullup: True
  filters:
    - delayed_on: 10ms
  id: open_close_valve_button
  on_press:
    then:
      if:
        condition:
          or:
            # If the valve is fully closed, OR if valve is not fully closed, but not fully open either
            # ie. it is partially open
            # Then open the valve fully
            - binary_sensor.is_on: valve_closed_limit_switch
            - binary_sensor.is_off: valve_open_limit_switch
        then:
          - script.execute: open_water_valve

        else:
          - script.execute: close_water_valve
```

Figure 4.50 - Physical Button Code

In the future, one way to improve this is to allow the valve to be turned manually. This way, the valve doesn’t need a network connection or electricity to be controlled.

A simple Thin Film Transistor (TFT) Liquid Crystal Display (LCD) was also implemented. TFT displays are LCDs that use thin-film transistors to display images [78]. This allows greater control over individual pixels, resulting in a clearer picture and better colour quality when compared to passive-matrix LCDs, which consist of row and column electrodes that overlap to form pixels. Because of this, special matrix drive schemes have to be used, which do not allow a wide range of voltages to be applied to individual pixels [79]-[81].

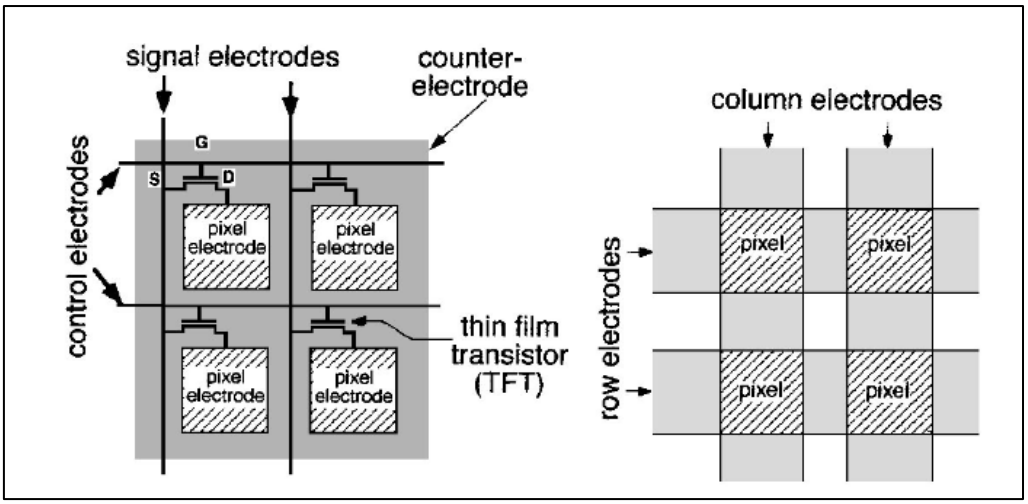


Figure 4.51 - Comparison on TFT LCD (left) with passive-matrix LCD (right) [81]

The display used was a GC9A01. This was available already, so no purchasing was required; however, implementing it into ESPHome required some additional work. The "request" for the component to be added into the main ESPHome codebase had not been accepted yet. Therefore, the "external_component" option was used. This allows functions and components outside of the main ESPHome codebase to be implemented. The "lambda" function of the display then outputs the desired information, which for this project was the flow rate, water temperature and the state of the water valve (open, closed, partially open).

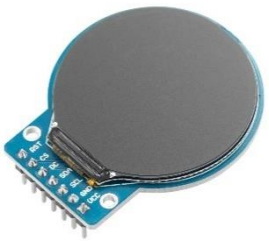


Figure 4.52 - GC9A01 TFT LCD Display [71]

```
######################
#### LCD DISPLAY ####
######################
external_components:
  # External Component needed because not integrated yet
  - source: github://bearpawmaxim/esphome@pr3625fix
    components: ["gc9a01"]
font:
  - file: "gfonts://Roboto"
    id: font_roboto
    size: 20
  - file: "gfonts://Material+Symbols+Outlined"
    id: icon_font_30
    size: 30
    glyphs: [
        "\U0000e1ff", # thermostat
        "\U0000e176", # water waves
        "\U0000e224", # valve
      ]
spi:
  clk_pin: GPIO14
  mosi_pin: GPIO13
display:
  - platform: gc9a01
    cs_pin: GPIO27
    dc_pin: GPIO12
    reset_pin: GPIO26
    rotation: 0
    update_interval: 15s
    # Display is 240x240
    lambda: |-
      // Flow rate
      it.print(24, 40, id(icon_font_30), "\U0000e1ff");
      it.printf(60, 40, id(font_roboto), "%.1f °C", id(supply_water_temperature_sensor).state);

      // Water Temperature
      it.print(24, 80, id(icon_font_30), "\U0000e176");
      it.printf(60, 80, id(font_roboto), "%.1f L/min", id(flowrate_sensor).state);

      // Position/open-close state
      it.print(24, 120, id(icon_font_30), "\U0000e224");
      if(id(valve_open_limit_switch).state){
        it.printf(60, 120, id(font_roboto), "OPEN");
      }else if (id(valve_closed_limit_switch).state) {
        it.printf(60, 120, id(font_roboto), "CLOSED");
      } else {
        it.printf(60, 120, id(font_roboto), "PARTIALLY OPEN");
      }
```

Figure 4.53 - Code to implement the TFT display

4.8. Edge Device PID Controller

The final part of the design of this sensor looked at the possible implementation of a PID (Proportional Integral Derivative) [72] controller to control the water flow through this sensor. To do this, a particular flow setpoint would be set, and the valve would automatically adjust itself to maintain a consistent flow rate. This has many applications in both domestic and commercial environments. Automatic irrigation is one example in domestic households, ensuring plants and trees consistently get the right amount of water.

The control system proposed is a unity feedback system [73] that uses the error between the desired and actual flow rate to control the position of the valve. A positive error indicates that the desired flow rate exceeds the actual flow rate, indicating that the valve needs to open. A negative error means the valve will have to close. To make the ball valve move, the desired position of the encoder is required. The necessary amount of movement can be calculated from the PID controller. The output from the controller would then be subtracted from the actual encoder value. This calculates the position to which the valve should move. Position 0 on the encoder indicates “fully open” (and eight indicates “fully closed”); hence, a positive error will decrease the value of the encoder, which will cause the valve to open, increasing the flow. Similarly, the valve will close when a negative error occurs (see Table 4.1 for an example). The desired encoder value is passed into the ”move_valve_to_position” function described earlier, however a saturation block is used to ensure the value lies between 0 and 8. Inside the software, this can be implemented separately, or when calling the “move_valve_to_position” function, the input can be capped accordingly. The flow rate is then measured, and the loop starts again.

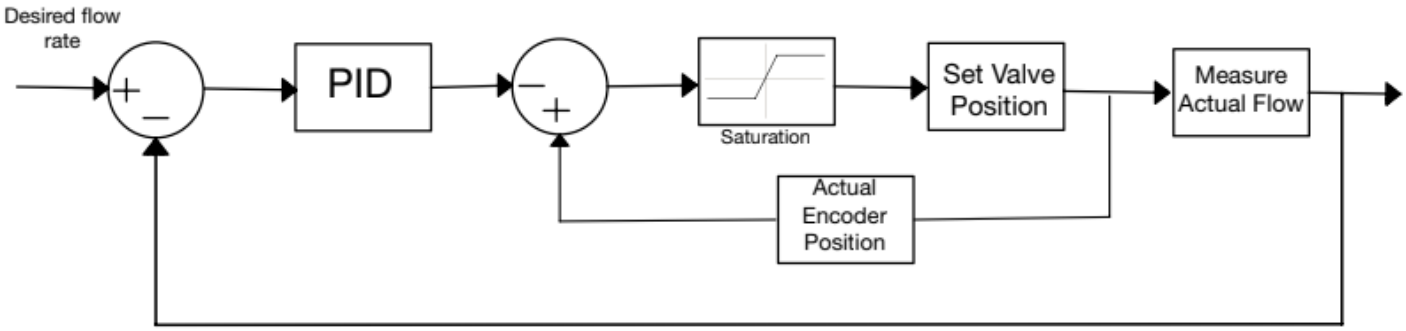


Figure 4.54 - Block diagram for edge device control system

Table 4.1 - Example of control system outputs

Desired Flow Rate (L/min)	Current Flow Rate (L/min)	Error	Example output from controller	Actual Encoder Position	Desired Position	Value passed to "move" function
25	15	10	3	4	1	1
5	15	-10	-3	4	7	7

Table 4.1 shows how the control system would react to two different inputs. It is assumed the flow rate is 15 L/min. One of the inputs will cause a positive error, and the other will give a negative error. The outputs from the controller are just examples in this case. Fully designing this controller would take considerable time as many variables would have to be considered. For example, the water pressure in the pipe would also affect the response, which might not always be constant.

Implementing this control system in ESPHome could be done using a "lambda" function, or the "pid" component could be extended. Currently, this component only works with "climate" sensors, where a specific output controls the temperature of a zone, but it could be modified/extended to work with more general input/output systems.

Future work for this controller includes designing the control parameters for a PID controller (Kp, Ki, Kd) and fully implementing and testing this control system. A more precise position control method could also be explored, for example, by using a potentiometer to measure an analogue value for the position rather than discrete steps.

4.9. Summary

Chapter 4 delved into the development of the device used for flow control and water metering. This included the mechanical design, microcontroller programming and circuit setup. Firstly, the programming method and the integration into Home Assistant were explained. The flow sensor was then implemented, and the methods for calculating the flow and volumetric usage were described. The mechanical design and flow control using the motor method were explained. The power supply and physical display implementation were discussed before looking at how a PID control system could be implemented to use this device to keep the flow rate constant. The entire circuit diagram, SolidWorks drawings, and edge sensor can be seen in the Appendix section.

Chapter 5 - Additional Temperature Sensor Design

5.1. Reason for Implementation and Microcontroller Chosen

An additional sensor was developed to show the expandability and power of Home Assistant and ESPHome. This was a temperature sensor. This chapter will explore the circuit and software setup for this sensor. Chapter 6 will look at how this sensor was integrated into the home automation server alongside the flow control device.

The ESP32 microcontroller was used again as these can easily be programmed using ESPHome and do not require much hardware to make a simple device. The advantage of this over buying a temperature sensor with Wi-Fi capability is that the GPIO pins on the ESP can be used to include additional sensors or functionalities. This also allows for a very customisable solution to be created with very advanced features implemented.

The ESP32 was set up using the same template code as shown in Chapter 4 to allow for Wi-Fi connection, over-the-air updates, and Home Assistant integration.

5.2. Sensors Implementation

5.2.1. Temperature and Humidity Sensor

Firstly, the temperature sensor had to be implemented. The sensor that was available to use was a DHT11. This sensor can read temperature and humidity and has a built-in component in ESPHome.

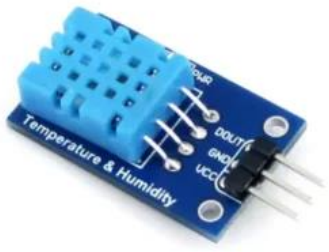


Figure 5.1 - DHT11 Temperature Sensor

The sensor module has three pins: V_{cc}, GND and DOUT. V_{cc} and GND were connected to the 3.3V and ground pins, respectively, and the DOUT pin was connected to GPIO23. A single block of code was required to add the functionality to read the data:

```
sensor:
  ############################
  ## TEMP & HUMIDITY SENSOR ##
  ############################
  - platform: dht
    pin: GPIO23
    model: DHT11
    temperature:
      id: zone_air_temperature_sensor
      name: "FYP Temperature"
    humidity:
      id: zone_air_relative_humidity_sensor
      name: "FYP Relative Humidity"
    update_interval: 60s
```

Figure 5.2 - Temperature and Humidity sensor code

Implementing this in a low-level programming language like C requires deep knowledge of the sensor's output signal. This requires the microcontroller to send a signal to the DHT, which will then output a series of pulse signals at fixed time intervals before sending the data packet, which consists of five 8-bit segments [74]. By using ESPHome, the data can be read and integrated into other smart home systems using just 14 lines of code.

5.2.2. Adding extra sensors

An additional component was added to showcase the expandability and customisation of using an ESP32. This was a simple light-dependant resistor (LDR) used to display the light level in the room. The LDR acts as a variable resistor, and the resistance decreases as more light shines on it [75].

Figure 5.3 - Light Dependant Resistor

This was wired in series with a 10kΩ resistor, and the voltage was measured using the ESPHome Analog-to-Digital Converter component. At this point, the maximum and minimum voltage were measured, and the reading was turned into a percentage using the "filter" functionality so that the measurement was easier to understand.

```
########################
## LIGHT SENSOR - LDR ##
########################
- platform: adc
  id: illuminance_sensor
  pin: GPIO34
  name: "LDR Light Sensor"
  attenuation: auto
  update_interval: 15s
  filters:
    - lambda: |-
        const float max_voltage = 3.12; // Maximum voltage
        const float min_voltage = 0.08; // Minimum voltage
        float percentage = ((x - min_voltage) / (max_voltage - min_voltage)) * 100.0;
        return (percentage < 0) ? 0 : (percentage > 100) ? 100 : percentage; // Keep value between 0 and 100
  unit_of_measurement: "%"
  accuracy_decimals: 2
  icon: "mdi:brightness-5"
```

Figure 5.4 - Light Dependant Resistor in ESPHome

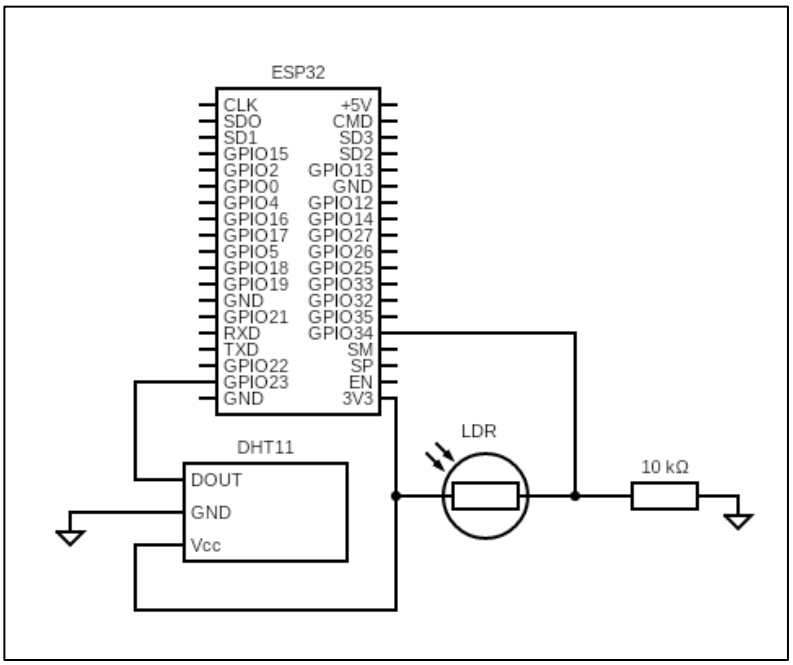


Figure 5.5 - Wiring diagram for DHT11 and LDR

This additional sensor could then be used for data collection, or can be used to trigger automations such as turning lights on at a certain point. Thanks to the advancements in open-source software, implementing additional features like this has become extremely straightforward and opens a wide array of opportunities for home automation. Additionally,

if the final product is designed with a modular case and circuit, then additional features can be implemented at later times and allow for sensors to be easily replaced when they break. This saves money and reduces the waste from purchasing new devices when only a small problem occurs, helping combat the e-waste problem in the world [76].

5.3. LCD Display for Physical Monitoring

As mentioned earlier, although it is nice being able to monitor the data in Home Assistant, or other dashboards, it is always good to have a physical means of seeing the data without needing a network connection or having to take out a smartphone.

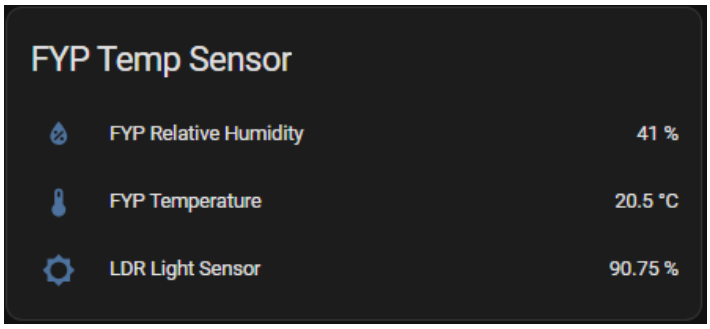


Figure 5.6 - Data in Home Assistant

To overcome this, an LCD display, an LCD ADM1602K-NSW-FBS, was implemented to visualise the data on the device itself. This was different from the TFT display used in Chapter 4. It had a lower resolution and less flexibility in what could be displayed, but it was enough to display the information needed for this device. This can display 16x2 characters.

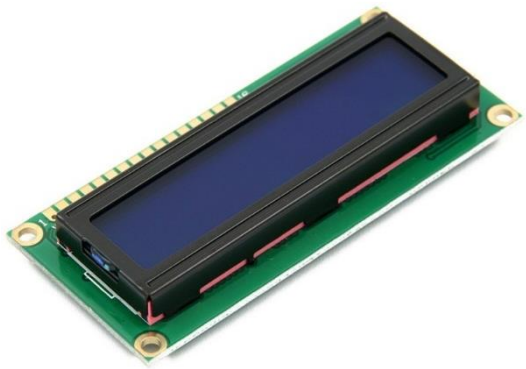


Figure 5.7 - LCD used for this project

This display can function in 4-bit and 8-bit mode. For this project, it was decided to use 4-bit mode as all the necessary symbols and alphanumeric characters were available this way. It

also saves on the number of GPIO pins that it uses on the ESP, allowing for additional expansion in the future. To use 4-bit mode, the pins D4 to D7 on display were connected to GPIO32, 25, 26, and 27, respectively. The display pins D0 to D3 were left disconnected. The other pins were connected as follows:

Table 5.1 - LCD pins connections

Pin	Description and Connection
V_{SS}	This is the display connection to ground. It was connected to 0V.
V_{DD}	This is the power to the display. It was connected to +5V.
V_0	This pin controls the contrast of the display. It was connected to 5V with a 2.2kΩ resistor and a potentiometer so that the contrast could be manually adjusted via the potentiometer.
RS	This is the register select pin. It was connected to GPIO13.
RW	This is the read/write select pin. "Write" mode is used to display information on the display, and "read" mode is used to read the data programmatically. Since the display would not be put into "read" mode, it was connected to 0V, as this keeps it in write mode at all times.
E	This is the "enable" pin. It was connected to GPIO14.
A	This is the positive side of the backlight LED, which controls the brightness of the display. It was connected to +5V via a 1kΩ resistor.
K	This was the negative side of the backlight LED. It was connected to ground.

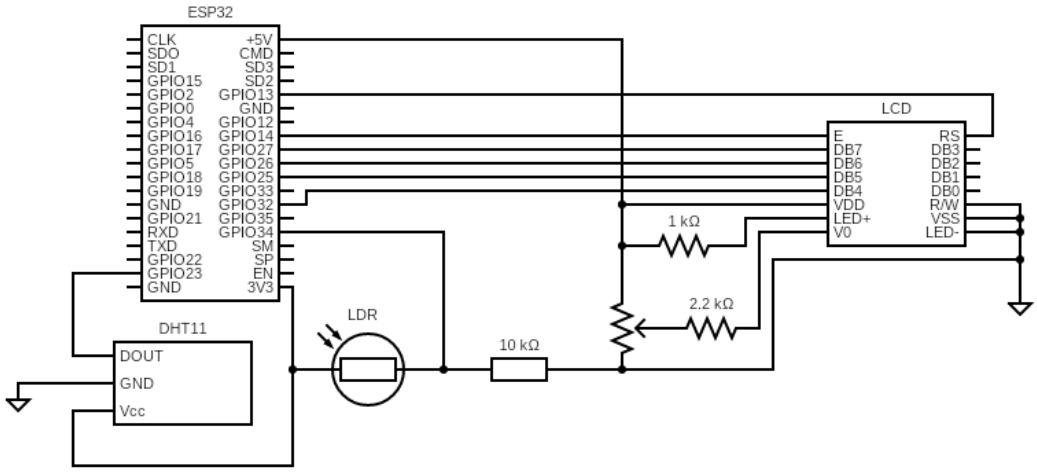


Figure 5.8 - Additional Sensor Circuit Diagram

Chapter 6 - Smart Home Integration and PID Controller

6.1. Controlling Flow Rate using Temperature Sensor

Implementing the ability to control the flow required the setup of a control system. This was done similarly to the block diagram in section 4, except the input was the desired temperature value, and the output was the actual temperature value. The basis of this control system was that increasing the mass flow rate of water through a radiator will increase the heat output [82], similar to automatic thermostatic radiator valves (TRVs). Therefore, in the proposed system, opening the ball valve (reducing the encoder value) will cause the room to heat up.

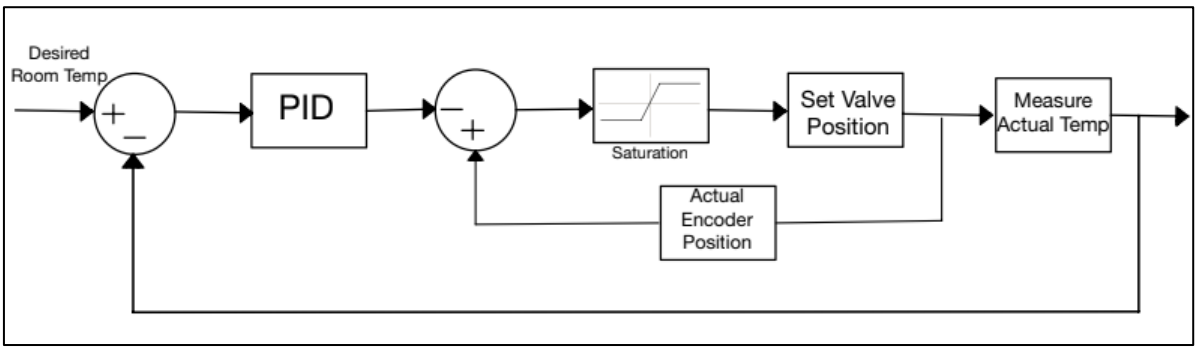


Figure 6.1 - Control System for flow control using room temperature

This system integrated all three components from this report. The desired temperature was set inside Home Assistant and set up in Chapter 3. The actual temperature was measured using the temperature sensor from Chapter 5, and the flow rate to heat up the room was controlled using the device from Chapter 4. This system could be implemented via Home Assistant automation or inside Node-RED. The latter option was chosen as it has a much better user interface, and the implementation of a PID controller using the functional block is possible.

The Node-RED system was developed using the ability to trigger one loop of the system or by letting it run independently and polling for the temperature every 5 minutes. Figure 6.2 shows the entire flow developed. The components were grouped together to highlight which device/system each “node” corresponds to. The data is read from the Home Assistant database but could also be read directly from the device. The dotted black arrows represent this in the architecture diagram (Figure 3.15).

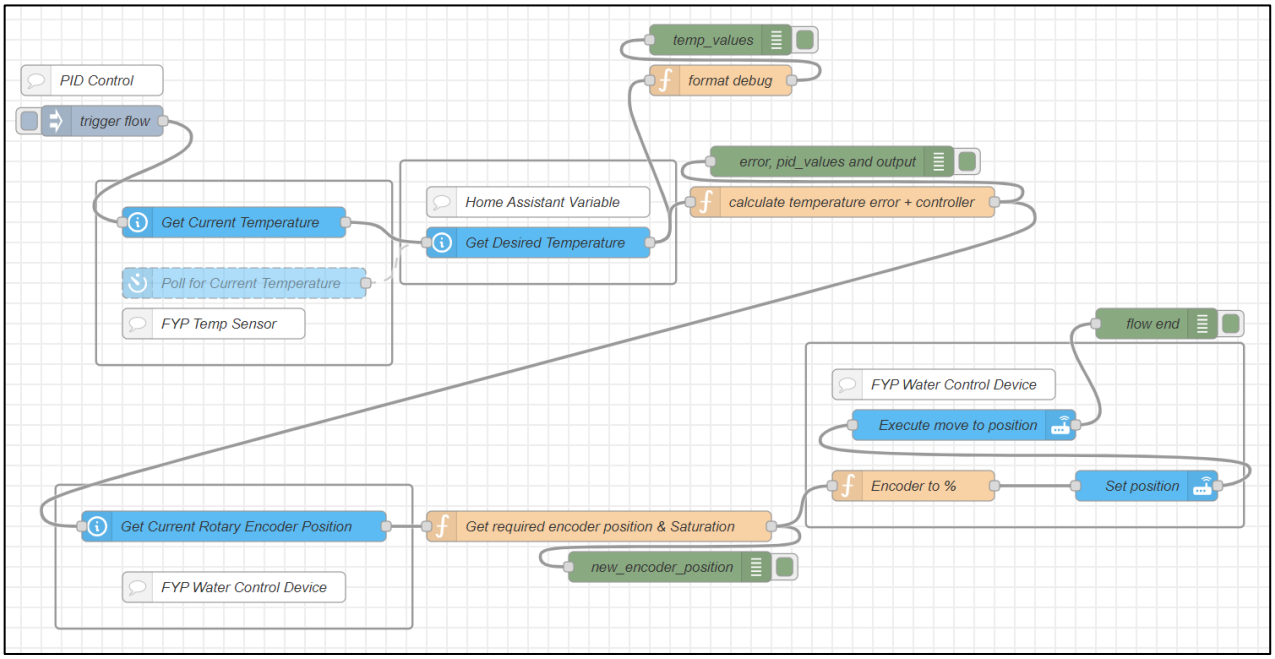


Figure 6.2 - Control System in Node-RED

The green nodes in this figure are “debug” blocks used to log desired values in the debug console. The flow starts by reading the current temperature and desired temperature. The “controller” functionality was implemented using JavaScript code:

```
let temp_error = msg.desired_temp - msg.current_temp;

// PID Constants
let Kp = 2.5;
let Ki = 0.1;
let Kd = 0.01;

// initialise stored data
let integral = context.get('pid_integral') || 0;
let previousError = context.get('pid_previousError') || 0;

let Pout = temp_error * Kp;
let Iout = integral + (temp_error * Ki);
let Dout = (temp_error - previousError) * Kd;

let output = Pout + Iout + Dout;
```

Figure 6.3 - PID controller in JavaScript

Chapter 7, ' Results and Discussions, ' explains how to tweak the three constants, Kp, Ki, and Kd. The ‘integral’ and ‘previousError’ are stored inside the global context of Node-RED so that they are not re-initialised after each iteration.

The controller output was then used to find the new required rotary encoder position, which was then used to call the water control device's “move_valve_to_position” (Figure 4.48) function.

```
let encoder_change = msg.current_encoder_pos - msg.controller_output;
encoder_change = Math.round(encoder_change);

if (encoder_change > 8){
    encoder_change = 8;
}else if (encoder_change < 0){
    encoder_change = 0;
}

msg.new_encoder_position = encoder_change;
```

Figure 6.4 - Code to get the new encoder position and the saturation block implemented using "if" statements

6.2. Simulating Response

Because it was not possible to physically test this control system using an actual radiator, the system was simulated using thermodynamic principles of heat transfer and typical values for a room. Taking a household radiator with a maximum heat output of 1000W and a typical drop of 11°C across the inlet and outlet of the radiator [83], the heat output could be estimated using the heat capacity formula [84]:

$$\dot{Q} = \dot{m} * c_p * \Delta t \tag{9}$$

At 1000W, using the heat capacity of water as 4186J/(kg-K), the flow rate = 0.0217 kg/s, or 2.17×10^{-5} m^3/s. In this project, the ball valve could only be in discrete positions, as it was controlled via the rotary encoder. Hence a relationship between the angle of the ball valve and the heat output from the radiator could be found. The angle of the ball valve could only be between 0 and 90° and in steps of 11.25° (because of the rotary encoder). At fixed velocity, the mass flow rate is proportional to the cross-sectional area [85]:

$$\dot{m} = \rho * A * v \tag{10}$$

So, by finding the cross-sectional area at certain angles, the flow rate and the heat output can be calculated. The ball valve in this project was a ½ inch pipe, which has an internal diameter of 12.7mm. From [86], the formula for the cross-sectional area of a ball valve is:

$$A = \pi * r^2 * (1 - \sin(\theta)) \tag{11}$$

At 1000W heat output, θ = 0 corresponds to the entire area of $1.267x10^{-4}$ m^3. Using Equation 9, the mass flow rate based on the 1000W heat output was 0.00217 kg/s. The mass flow rate is 0 when the area is 0; hence, the linear relationship can be found. The slope is $\frac{0.0217-0}{1.267x10^{-4}-0} = 171.4.$; hence, the mass flow rate for water can be found from the area:

$$\dot{m} = 171.4 * A \tag{12}$$

This flow rate is used in Equation 9 to get the heat output into the room.

Table 6.1 – Encoder Position and Ball valve angle vs heat output into the room

Encoder Position	Angle (°)	Area (m^2)	m_dot (kg/s)	Heat to room $\dot{Q}_{in}$ (W)
0	0	1.267E-04	2.17E-02	1000
1	11.25	1.020E-04	1.75E-02	804.91
2	22.5	7.820E-05	1.34E-02	617.32
3	33.75	5.630E-05	9.65E-03	444.43
4	45	3.710E-05	6.36E-03	292.89
5	56.25	2.135E-05	3.66E-03	168.53
6	67.5	9.643E-06	1.65E-03	76.12
7	78.75	2.434E-06	4.17E-04	19.21
8	90	0	0	0

The room will lose some heat depending on its size and the outside temperature [87]. The "Net Heat" into/out of the room is found using: $\dot{Q}_{net} = \dot{Q}_{in} - \dot{Q}_{out}$. This would mainly be lost through the walls, windows, roof, and ventilation. These values depend on the temperature difference between the inside room and outside the building however, for brevity it was kept constant throughout the simulation. Two values for the heat loss were used; 415W was the first, which is the value used in [87]. The initial room temperature used was 13°C, and the outside temperature was set at 0°C, so using the formulas in [87], the heat loss at this temperature difference was 234.78W. Since the system in Node-RED was set up to poll for the temperature every 5 minutes, the net heat was multiplied by 300 seconds to get the total heat gained by the room during that time. The room used in [87] was $4 * 5 * 2.45 = 49m^3$. The density of air is 1.293kg/m^3 [88] and has a heat capacity of 721 J/(kg-C) [84], so the temperature change *inside* the room during a 5-minute period was calculated by rearranging Equation 9:

$$\Delta t_{room} = \frac{\dot{Q}_{net} * 300}{721 * 1.293 * 49} \tag{13}$$

Table 6.2 - Finding Temperature change at different heat input levels when Qout = 415W

Heat to room $\dot{Q}_{in}$ (W)	Heat Loss $\dot{Q}_{out}$ (W)	Net Heat Gain $\dot{Q}_{net}$ (W)	Total Energy Transfer (5 mins - J)	Δt 5 mins (°C)
1000	415	585	175500	3.84
804.91	415	389.91	116972.9	2.56
617.32	415	202.32	60695.0	1.33
444.43	415	29.43	8828.9	0.19
292.89	415	-122.11	-36632.0	-0.80
168.53	415	-246.47	-73940.9	-1.62
76.12	415	-338.88	-101663.9	-2.23
19.21	415	-395.79	-118735.6	-2.60
0	415	-415	-124500	-2.73

Table 6.3 - Finding Temperature change at different heat input levels when Qout = 234.78

Heat to room $\dot{Q}_{in}$ (W)	Heat Loss $\dot{Q}_{out}$ (W)	Net Heat Gain $\dot{Q}_{net}$ (W)	Total Energy Transfer (5 mins - J)	Δt 5 mins (°C)
1000	234.78	765.22	229566	5.03
804.91	234.78	570.13	171038.9	3.74
617.32	234.78	382.54	114761.0	2.51
444.43	234.78	209.65	62894.9	1.38
292.89	234.78	58.11	17434.0	0.38
168.53	234.78	-66.25	-19874.9	-0.44
76.12	234.78	-158.66	-47597.9	-1.04
19.21	234.78	-215.57	-64669.6	-1.42
0	234.78	-234.78	-70434	-1.54

During the simulations, the initial temperature of the room was varied. However, the valve always started in its closed state (i.e. rotary encoder = 8). The system was simulated via Excel by first setting the initial conditions and then using the error and the K_p, K_i, and K_d coefficients to find the required rotary encoder position. The ball valve would move to this position, and remain there for 5 minutes, during which the room temperature would change according to the values in Tables 6.2 and 6.3. The output temperature can be seen in Chapter 7, with different values for the PID coefficients and initial conditions.

6.3. Advantages and Disadvantages of the System

Typical automatic TRV valves come with a temperature sensor built into them. The accuracy of this sensor can be affected due to the proximity of the sensor to the radiator, whereas having the sensor away from the valve improves the accuracy. Although the system in this scenario was pretty idealised due to the complicated nature of room heating, it helps act as a baseline for potential smart home heating implementations. The modular approach to this system and

the accessibility of open-source software allows users to create bespoke solutions that overcome these challenges.

An advantage of using the central server for processing, rather than the devices communicating directly, is that it reduces the load on the edge devices and allows the system to expand with more devices. This expandability paves the way for a truly unique and personalised user experience in a smart home.

A disadvantage of this system is the inability to control the hot water flow precisely. Due to the discrete nature of the ball valve's available positions, the room's heat output would be slightly too low or too high to maintain a constant temperature once the setpoint is met. To improve this project, adding the ability to control the valve in smaller steps, even as low as 1°, could improve the system's performance, reducing any oscillations in temperature.

6.4. Summary

This chapter explored the integration of the smart home setup and the two devices developed to set a room's temperature by controlling the flow into a radiator. The control system was set up in Node-RED and read the data from Home Assistant. A PID controller was incorporated to calculate the required flow rate.

A room simulation was used because implementing the system in a room or via a testing rig was not feasible for this project. The formulas and approximations used to model the room's heat gain were explained.

The final section looked at the advantages and disadvantages of the specific system, mentioning potential areas of improvement for future iterations of the product(s) and system.

Chapter 7 - Testing, Results, Discussions

7.1. Water Flow Sensor Accuracy

7.1.1. Testing

Testing the accuracy of the flow sensor consisted of connecting it to a water supply and letting the water flow into a container that showed the volume via markings on the side. Once the water in the container reached a desired point, it was stopped, and the measurement on it was compared to the results published to Home Assistant.

Because the device contained an electronic circuit in close proximity to a water source, the highest safety standards had to be maintained at all times. As the motor was not required for the accuracy test, the circuit was powered via a 5V power bank to remove the connection to mains power. The system was also contained in a water-resistant enclosure.

Figure 7.1 - Water Flow circuitry inside water-resistant enclosure to remove risk of damage and electrocution

7.1.2. Results

Multiple tests were carried out at different water volume values. Unfortunately, keeping a constant flow rate for each test was not feasible due to the testing rig available. This is a significant limitation as it may have introduced variability into the system.

The test was repeated for multiple volumes of water, starting from 0.5 litres up to 2.5 litres. The results for the first test were:

Table 7.1 - Water flow sensor test #1

Litres Used	Litres Measured	Error	Percentage Error (%)
0.5	0.3674	0.1326	26.52
0.5	0.3605	0.1395	27.9
0.5	0.3418	0.1582	31.64
0.5	0.356	0.144	28.8
0.5	0.3455	0.1545	30.9
0.5	0.3403	0.1597	31.94
0.5	0.3405	0.1595	31.89
0.5	0.3479	0.1521	30.41
0.5	0.3461	0.1539	30.78
0.5	0.3486	0.1514	30.29

The average error for this test was 30.01%, which is well outside a suitable range. The average errors for the remaining tests were:

Table 7.2 - Average error for flow sensor tests

Litres used	Average % error
0.5	30.10698095
1	30.31
1.5	30.90
2	31.33
2.5	31.22

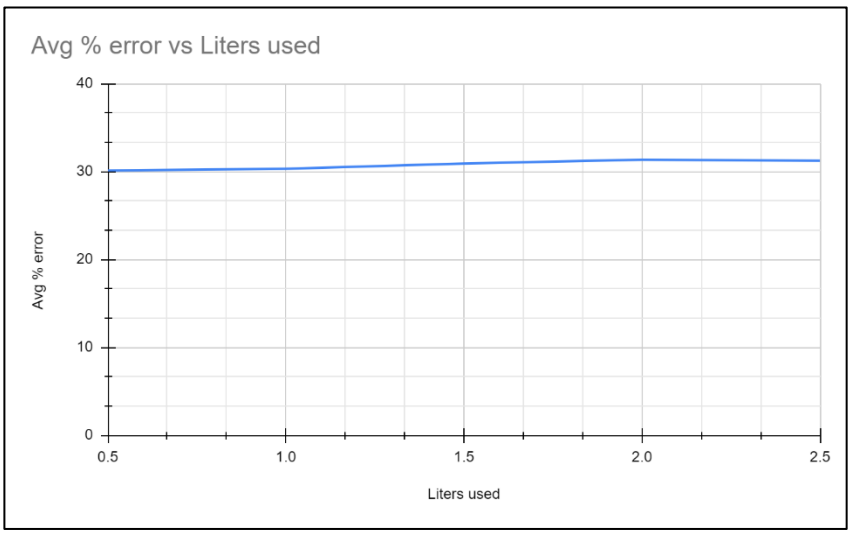


Figure 7.2 - Graph of average error for the flow sensor tests

The rest of the tables can be seen in the appendix section.

7.1.3. Discussion

Based on the test results, the accuracy of the flow sensors does not meet the standard required. It also does not fall within the range given in the datasheet of ±3%. It is noticed, however, that the errors are all within 3% of the average error for all tests. This leads to the conclusion that the sensor is not faulty/inaccurate, but rather the flow constant given in the datasheet is incorrect (see Equation 1). This can be analysed further by thoroughly examining the results using different multiplier constants.

Other factors that may affect the results could include the update interval of the pulse counter in ESPHome; however, this would not cause such a significant change in the results.

To thoroughly test this sensor, a testing rig that can supply a set flow rate would be ideal, as the flow rate and volumetric usage can both be compared. At the moment, many variables may be affecting the results.

If the flow control device were to be made into a full product, an ultrasonic sensor would be preferred; however, this can add to the cost. If this sensor could be tested thoroughly and the correct flow coefficient found, it could be a great substitute for more expensive flow sensors.

7.2. Node-RED Control System Response

7.2.1. Simulation(s) Response

The room temperature described in Chapter 6 was predicted using different values of the PID controller coefficients. The system always started with the valve in the fully closed position, and the controller would calculate the required position based on the difference between the desired and current temperature. The desired temperature was set to 21°C and the initial room temperature to 13°C. The interval at which the control system calculated the required position was in 5-minute intervals. Between readings, the temperature change was approximated as a linear change, hence the straight lines used in the graph.

The two best simulations can be seen below: the first, when the room's heat loss was calculated at 234.78W, and the second when it was 415W. The controller coefficients are in the caption under each figure. The choice for the room's heat loss values is explained in Chapter 6 (See Tables 6.2 and 6.3).

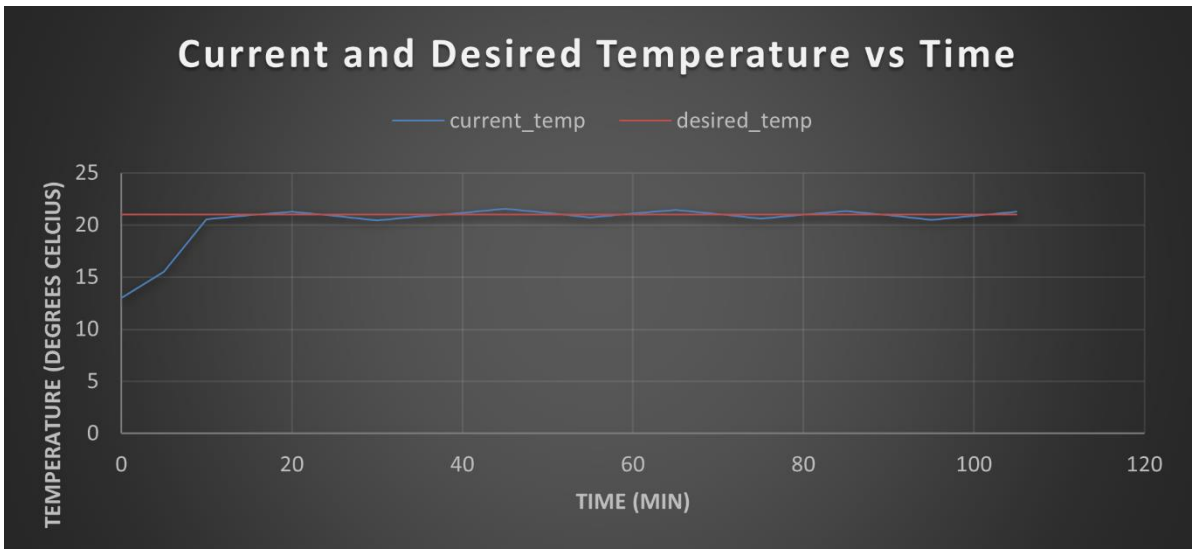


Figure 7.3 - Qout = 234.78W, Kp = 0.8, Ki = 0, Kd = 0.8

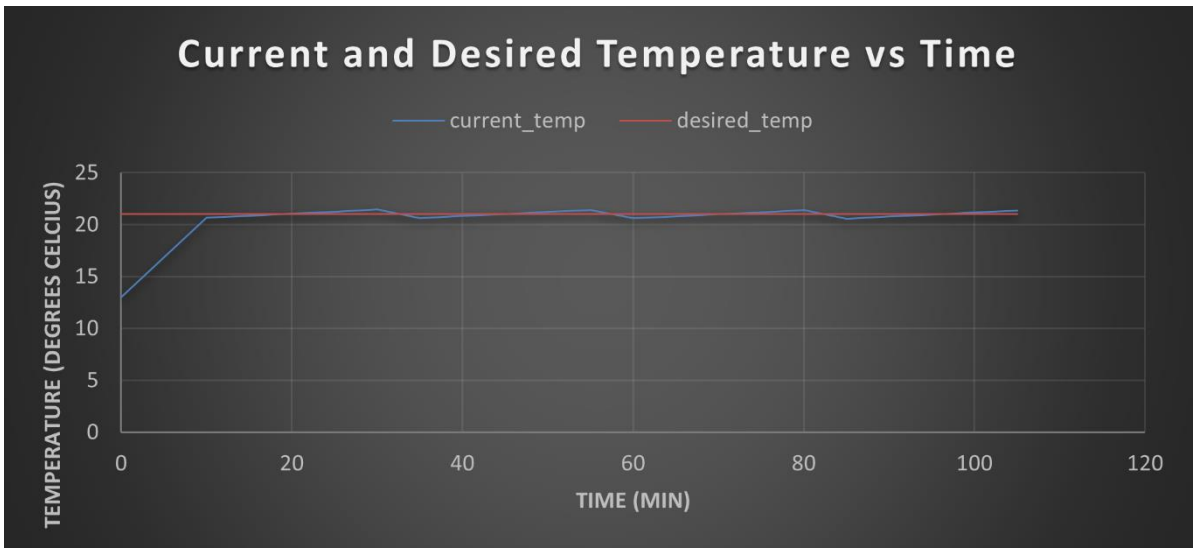


Figure 7.4 - Qout = 415W, Kp = 1, Ki = 0, Kd = 0.8

7.2.2. Discussions

The two graphs show the best results of the simulation. When the room's heat loss was 234.78W, the PID coefficients were K_P = 0.8, K_I = 0, and K_D = 0.8. Similar results were obtained with the simulation when the heat loss was 415W. In this scenario, the coefficients were K_P = 1, K_I = 0, K_D = 0.8. Due to the discrete positioning and application of the control system, the graphs do not have their typical smooth curve, however two response graphs have familiar shapes. Both charts show a first-order response, with no overshoot present. This is preferred, as causing a room to heat up too much before letting it cool down is not ideal for

the occupants, and it slows down the settling time (which would also affect the energy use in reality). The discrete nature of the positioning also makes the slight oscillation visible. However, this is so minimal that it would not be noticeable in real life. The desired setpoint is reached in 10 minutes (two control system cycles). Having the ability to control the valve precisely would remove the oscillations as the controller would find the position of net-zero heat loss, which means the radiator is providing the exact amount of heat the room is losing, hence keeping the temperature constant.

Multiple tweaks of the controller parameters were completed to get these results. This was done using trial and error by changing the values and seeing how it affected the output response. During the tweaking, it was found the K_I caused overshoot to occur in the system. As explained earlier, this is not ideal in a heating scenario, as a room that is too warm is uncomfortable for the occupiers.

These results are not a perfect representation of how the room's temperature would react because the model of the room heating is fairly idealised. As the heat loss depends on the difference between indoor and outdoor temperatures, more accurate results could be found by modelling this heat loss as a function using the two temperature values. However, these simulations show great potential in automating individual room temperatures. Typical thermostats would trigger the heating of the entire building to start/stop, which can be wasteful and unnecessary. Using this system, only desired rooms are heated, and it is also expandable to include multiple rooms by implementing various sensors and control valves. The heating of rooms also depends on many variables, causing heating to happen at different rates in each room. By being able to model the room's temperature based on these parameters (size, windows, wall thickness, etc), each control system can be easily modified to give a truly personalised experience. The PID controller variables could be calculated by Home Assistant based on other sensor values (outdoor temperature, etc.) or inputted by the occupant directly, and the room’s heating observed to get the best results.

The Excel sheet used to derive the results: https://docs.google.com/spreadsheets/d/1WCr-pj_5fI8n8utomYGmq0MgR3k7acHB/edit?usp=drive_link&ouid=107548194556341954004&rtpof=true&sd=true

Chapter 8 - Ethics

Who owns the data?

A growing ethical concern in the world of IoT is data ownership. It can be argued that the device's manufacturer has a right to the data and may require that it be stored in their database, as well as anywhere the user of the device desires (e.g., Home Assistant). This is a growing debate and a large reason many smart home enthusiasts are using platforms like ESPHome and Home Assistant, where the data does not leave their local network. However, less technical users of smart home devices may not be able to set up these services and must use the devices as they are supplied by the manufacturer. In this scenario, the manufacturer must ensure that the data is stored safely, especially if it is personal information such as health data or presence/location-based information. This is to ensure data breaches cannot occur. On top of this, many companies sell the data they collect to third parties. This has caused large debates in the past, and reputable companies should avoid this practice.

Vendor-lock

In the realm of IoT, vendor lock is when manufacturers “lock” customers into their ecosystem of devices by not allowing them to communicate with devices from other manufacturers. This is often difficult to overcome, especially for less technical users. It also hinders the advancements of new solutions using interconnected devices, like the one in this project. When vendor lock happens, the device must be connected to the vendor's server, which allows them to read all the data collected by the device or even stop it from working. Hence, to allow for continuous innovations, device manufacturers must do the ethical choice, and allow users to control their devices without requiring a connection to their servers. However, they might not prefer this because it can reduce their recurring revenue, as they usually charge a subscription to allow the devices to work.

Water supply shutoff

This project mentioned that internet-connected water meters could be remotely shut off by water suppliers if the respective household does not pay the necessary charges. Although it may seem correct that the occupants of a house must pay the necessary fees, water is still a basic human right. The water supply company must take great care to ensure that the house is not fully deprived of water. They may consider limiting the water supply rather than fully shutting it off.

Health and Safety

Working with electronics near water sources is always a big concern regarding health and safety. This device also requires a connection to mains electricity power, which can pose an even bigger risk and may even result in death. The device must always be designed to prevent water from entering the system and ensure all electronics are in water-resistant enclosures. In this project, the adapter connecting to mains power was long and had all electronics in a water-tight enclosure. This should always be the case to ensure that users are always safe.

Sustainability

All devices should be designed with sustainability in mind. In this project, “modular” approaches to design are mentioned. Many companies design and build devices that are not repairable. This adds to the e-waste in the world, causing good electronic components and non-recyclable parts to go into waste. Designing devices to be repairable and expandable helps increase the lifetime of these devices and can even allow devices to be upgraded without having to purchase an entire new device. This means that companies make less money, so an ethical balance must be struck between profitability and sustainability.

Chapter 9 - Conclusions and Further Research

The overall solutions proposed in this project are a great starting point for solving many engineering problems. The flow control valve was used in the example of water metering, using Home Assistant to detect leaks, notify users and shut off the valve while also displaying the water usage in the home. The same device was then used to mimic the automatic control of room heating, and explanations of how this device can be tailored to improve the current implementations were given. Some simulations were conducted to display how this solution would keep the temperature in a room at the desired setpoint. These proved to be successful, as the device was able to maintain the desired temperature with little variation. This solution consisted of an additional temperature sensor to provide feedback to the device on the current room temperature. ESPHome was used to program both ESP32-based devices. These devices can still be extended thanks to ESPHome being open-source. The devices, with additional data or functionality, can be tailored to each occupant's wishes. Home Assistant was used to connect the devices, which is another open-source platform. The “local-first” approach of Home Assistant helps in the uptime of the overall system, as internet outages do not affect the ability of devices to communicate with one another.

This project shows that open source is the future of home automation. No other vendor comes close to the integration abilities of Home Assistant. This is great for home occupants, as they can get started on their smart home completely free! The advancements of ESPHome also show how less technical users can design and program microcontroller-based devices in a much faster time, and it also can work as a basis for prototyping. The ability to quickly implement the software while still testing the hardware choices can save time and money, boosting innovation. For example, this project's YAML file for the flow control device contained over 500 lines. When translated into C++, the folder is over 17.6MB and includes over 800 files. Implementing this much software in less than a year would be impossible without the help of a platform as powerful as ESPHome.

The use of a PID controller to implement automatic heat control also shows great potential. Although the implementation in this project was fairly limited, the simulations completed still showed positive results. This system can further be improved from this point forward by creating a more accurate model of the room's heat loss and implementing a more precise flow control valve. The ability to integrate different devices to create such a system is also positive news towards a more connected and “open” smart home.

Some future work is needed on implementing the PID controller on the edge device. Although this was only touched on slightly in this report, it can be an extremely powerful option for automation.

As mentioned many times throughout this report, a PID controller for automatic room heating could be drastically improved by being able to monitor the position of the valve more precisely. This could be done using a potentiometer.

The ability to manually control the valve is also essential. In this project, a button was implemented on the device to open/close the valve. However, a better approach would be for the gears to be decoupled and the ball valve rotated by the user, allowing the valve to be opened/closed without turning the motor (which may potentially damage it).

Some of the software implementations could also be improved in this project. The "execute_move_to_position" should be called automatically when the number is set. Unfortunately, due to time constraints, it wasn't possible to work on improving this, but it can definitely help with the user experience of this product.

Improving the implementation of better leak detection would also be highly beneficial. At the moment, leaks are only detected at night, when there are usually "times of null flow". However, a better approach would be to have a more real-time detector that works based on previous usage data. This is an enormous task, which could be its own project; however, it shows the potential expandability of this project.

Finally, designing an enclosed case and custom-printed circuit boards would help with the aesthetics of this project. However, with all the possible improvements that can be made, it is best to leave this until the end, when a final product is designed.

References

[1] M. R. Alam, M. B. I. Reaz, and M. A. M. Ali, "A Review of Smart Homes—Past, Present, and Future," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, no. 6, pp. 1190–1203, Nov. 2012, doi: 10.1109/TSMCC.2012.2189204.

[2] M. Schiefer, "Smart Home Definition and Security Threats," in *2015 Ninth International Conference on IT Security Incident Management & IT Forensics*, May 2015, pp. 114–118. doi: 10.1109/IMF.2015.17.

[3] P. P. Ray, "A survey on Internet of Things architectures," *Journal of King Saud University - Computer and Information Sciences*, vol. 30, no. 3, pp. 291–319, Jul. 2018, doi: 10.1016/j.jksuci.2016.10.003.

[4] S. Y. Y. Tun, S. Madanian, and F. Mirza, "Internet of things (IoT) applications for elderly care: a reflective review," *Aging Clin Exp Res*, vol. 33, no. 4, pp. 855–867, Apr. 2021, doi: 10.1007/s40520-020-01545-9.

[5] S. Al-Sarawi, M. Anbar, R. Abdullah, and A. B. Al Hawari, "Internet of Things Market Analysis Forecasts, 2020–2030," in 2020 Fourth World Conference on Smart Trends in Systems, Security and Sustainability (WorldS4), Jul. 2020, pp. 449–453. doi: 10.1109/WorldS450073.2020.9210375.

[6] "Three main elements that make up a successful IoT ecosystem," IIoT World. Accessed: Mar. 07, 2024. [Online]. Available: https://www.iiot-world.com/industrial-iot/connected-industry/three-main-elements-make-successful-iot-ecosystem/

[7] "Internet of Things (IoT) Terms Explained: Internet Protocol (IP) | Startup Tips | SigmaOS." Accessed: Mar. 07, 2024. [Online]. Available: https://sigmaos.com/tips/startups/internet-of-things-iot-terms-explained-internet-protocol-ip

[8] M. Asadullah and A. Raza, "An overview of home automation systems," in *2016 2nd International Conference on Robotics and Artificial Intelligence (ICRAI)*, Nov. 2016, pp. 27–31. doi: 10.1109/ICRAI.2016.7791223.

[9] H. Assistant, "Home Assistant Yellow," Home Assistant. Accessed: Mar. 07, 2024. [Online]. Available: https://www.home-assistant.io/yellow/

[10] G. N. Aodha, "Ireland is losing 43% of its drinking water in leaks - and has just gotten the data to start fixing the problem," TheJournal.ie. Accessed: Mar. 09, 2024. [Online]. Available: https://www.thejournal.ie/irelands-leaking-pipes-4847084-Oct2019/

[11] "Irish Water claims 46m litres leaking from homes daily," The Irish Times. Accessed: Mar. 09, 2024. [Online]. Available: https://www.irishtimes.com/news/environment/irish-water-claims-46m-litres-leaking-from-homes-daily-1.2159153

[12] N. Donaghy, "Dealing with water and flood damage in your home," Claim Management Group | Loss Assessors Ireland. Accessed: Mar. 11, 2024. [Online]. Available: https://claimmanagementgroup.com/dealing-with-water-and-flood-damage-in-your-home/

[13] "What Does Home Insurance Cover? | Chill." Accessed: Mar. 11, 2024. [Online]. Available: https://www.chill.ie/blog/what-does-home-insurance-cover/

[14] J. Singh, "Toxic Moulds and Indoor Air Quality," *Indoor and Built Environment*, vol. 14, no. 3–4, pp. 229–234, Jun. 2005, doi: 10.1177/1420326X05054015.

[15] "Cost of Black Mould Removal in Ireland" Accessed: Mar. 11, 2024. [Online]. Available: https://perfectclean.ie/blog/car-mould-removal-cost-ireland

[16] T. Boyle *et al.*, "Intelligent Metering for Urban Water: A Review," *Water*, vol. 5, no. 3, Art. no. 3, Sep. 2013, doi: 10.3390/w5031052.

[17] "Price of water - how UK water rates are calculated," Uswitch. Accessed: Mar. 12, 2024. [Online]. Available: https://www.uswitch.com/water/price-of-water/

[18] "Public reminded to conserve water throughout the year," Uisce Éireann. Accessed: Mar. 12, 2024. [Online]. Available: https://www.water.ie/news/public-reminded-to-conser/

[19] "Human Rights to Water and Sanitation," UN-Water. Accessed: Mar. 12, 2024. [Online]. Available: https://www.unwater.org/water-facts/human-rights-water-and-sanitation

[20] "The state of open source software," The State of the Octoverse. Accessed: Mar. 06, 2024. [Online]. Available: https://octoverse.github.com/2022/state-of-open-source

[21] Steve, "Understanding Smart Home APIs." Accessed: Mar. 12, 2024. [Online]. Available: https://stevessmarthomeguide.com/smart-home-apis/

[22] Y.-S. Hong and C.-H. Lee, "A design and implementation of low-power ultrasonic water meter," *Smart Water*, vol. 4, no. 1, p. 6, Nov. 2019, doi: 10.1186/s40713-019-0018-9.

[23] J. Zhang, "Designing a Cost Effective and Reliable Pipeline Leak Detection System," in *Proc. Pipeline Reliability Conference*, Houston, USA, Nov. 1996, pp. 1-11.

[24] A. Pietrosanto, M. Carratù, and C. Liguori, "Sensitivity of water meters to small leakage," *Measurement*, vol. 168, p. 108479, Jan. 2021, doi: 10.1016/j.measurement.2020.108479.

[25] R. Ghube, K. Kap, and M. G. Ghogare, "Development Of Motor Operated Ball Valve," in *Journal of Instrumentation & Control Engineering Department*, vol. 1, no. 19, pp. 23, Apr. 2012.

[26] "StreamLabs Control," StreamLabsWater. Accessed: Mar. 15, 2024. [Online]. Available: https://streamlabswater.com/pages/streamlabs-control

[27] "Water Hero - Advanced Leak Detection Systems," Water Hero. Accessed: Mar. 15, 2024. [Online]. Available: https://waterheroinc.com/

[28] "Flo Smart Water Monitor & Shutoff - 0.75" -- 900-001 -- Moen." Accessed: Mar. 15, 2024. [Online]. Available: https://www.moen.com/products/Flo-by-Moen/Flo-Smart-Water-Monitor-Shutoff---0-75-/900-001

[29] "IP Network (Internet Protocol Network)," Kentipedia. Accessed: Mar. 15, 2024. [Online]. Available: https://www.kentik.com/kentipedia/ip-network/

[30] T. Lee, "How to Set Up a Home Network: A Professional Guide." Accessed: Mar. 15, 2024. [Online]. Available: https://www.comms-express.com/infozone/article/how-to-set-up-a-home-network-a-professional-guide/

[31] "What is a Virtual Machine? | VMware Glossary," VMware. Accessed: Mar. 15, 2024. [Online]. Available: https://www.vmware.com/topics/glossary/content/virtual-machine.html

[32] "What Is Virtualization? | IBM." Accessed: Mar. 15, 2024. [Online]. Available: https://www.ibm.com/topics/virtualization

[33] "Oracle VM VirtualBox." Accessed: Mar. 15, 2024. [Online]. Available: https://www.virtualbox.org/

[34] “Enterprise Open Source and Linux,” Ubuntu. Accessed: Mar. 15, 2024. [Online]. Available: https://ubuntu.com/

[35] P. Kaur, J. K. Josan, and N. Neeru, “Performance Analysis of Docker Containerization and Virtualization,” in *Proceedings of Third International Conference on Communication, Computing and Electronics Systems*, V. Bindhu, J. M. R. S. Tavares, and K.-L. Du, Eds., Singapore: Springer, 2022, pp. 863–877. doi: 10.1007/978-981-16-8862-1_56.

[36] “What Is Docker? | IBM.” Accessed: Mar. 16, 2024. [Online]. Available: https://www.ibm.com/topics/docker

[37] “Docker Web UI: Portainer :: Docksal Documentation.” Accessed: Mar. 16, 2024. [Online]. Available: https://docs.docksal.io/use-cases/portainer/

[38] “What is headless server? | Definition from TechTarget,” WhatIs. Accessed: Mar. 16, 2024. [Online]. Available: https://www.techtarget.com/whatis/definition/headless-server

[39] H. Assistant, “Home Assistant,” Home Assistant. Accessed: Mar. 16, 2024. [Online]. Available: https://www.home-assistant.io/

[40] “Installation - Home Assistant.” Accessed: Jan. 17, 2024. [Online]. Available: https://www.home-assistant.io/installation/

[41] B. Nuttall, “What is a Raspberry Pi? | Opensource.com.” Accessed: Mar. 16, 2024. [Online]. Available: https://opensource.com/resources/raspberry-pi

[42] R. Hoeppli, “Forget microSD cards! Run your Raspberry Pi with a SSD!,” The Pi Project. Accessed: Jan. 18, 2024. [Online]. Available: https://medium.com/the-pi-project/forget-microsd-cards-run-your-raspberry-pi-with-a-ssd-4bbbeef7bd0d

[43] “ESPHome,” ESPHome. Accessed: Mar. 17, 2024. [Online]. Available: https://esphome.io/index.html

[44] “Flashing ESPHome on White Label Smart Plugs,” Flameeyes’s Weblog. Accessed: Mar. 17, 2024. [Online]. Available: https://flameeyes.blog/2021/03/02/flashing-esphome-on-white-label-smart-plugs/

[45] “What is YAML?” Accessed: Mar. 17, 2024. [Online]. Available: https://www.redhat.com/en/topics/automation/what-is-yaml

[46] “Automations and Templates,” ESPHome. Accessed: Mar. 18, 2024. [Online]. Available: https://esphome.io/guides/automations.html

[47] “External Components,” ESPHome. Accessed: Mar. 18, 2024. [Online]. Available: https://esphome.io/components/external_components.html

[48] “Node-RED.” Accessed: Mar. 19, 2024. [Online]. Available: https://nodered.org/

[49] “About : Node-RED.” Accessed: Mar. 19, 2024. [Online]. Available: https://nodered.org/about/

[50] “Welcome to Python.org,” Python.org. Accessed: Mar. 19, 2024. [Online]. Available: https://www.python.org/

[51] I. Alharsha, F. Memon, R. Farmani, and W. Hussien, “An investigation of domestic water consumption in Sirte, Libya,” *Urban Water Journal*, vol. 19, pp. 1–23, Aug. 2022, doi: 10.1080/1573062X.2022.2105239.

[52] S. Santos, “ESP32 vs ESP8266 - Pros and Cons,” Maker Advisor. Accessed: Mar. 20, 2024. [Online]. Available: https://makeradvisor.com/esp32-vs-esp8266/

[53] C. G. C. Carducci, A. Monti, M. H. Schraven, M. Schumacher, and D. Mueller, “Enabling ESP32-based IoT Applications in Building Automation Systems,” in *2019 II Workshop on Metrology for Industry 4.0 and IoT (MetroInd4.0&IoT)*, Jun. 2019, pp. 306–311. doi: 10.1109/METROI4.2019.8792852.

[54] “ESP32 Development Board - DEVKIT V1.” Accessed: Jan. 20, 2024. [Online]. Available: https://grobotronics.com/esp32-development-board-devkit-v1.html?sl=en

[55] E. Ramsden, "Chapter 1 - Hall Effect Physics," in Hall-Effect Sensors: Theory and Application. Elsevier, 2011

[56] “Water flow rotor/motor sensor wiring guide on Arduino | 14core.com.” Accessed: Jan. 17, 2024. [Online]. Available: https://www.14core.com/water-flow-rotormotor-sensor-wiring-guide-with-arduino/

[57] Y.-S. Hong and C.-H. Lee, “A design and implementation of low-power ultrasonic water meter,” Smart Water, vol. 4, no. 1, p. 6, Nov. 2019, doi: 10.1186/s40713-019-0018-9.

[58] “G12 Hall Effect Liquid Water Flow Sensor Switch Meter Counter DC 5V 125Lmin YFB7 on OnBuy.” Accessed: Jan. 18, 2024. [Online]. Available: https://www.onbuy.com/gb/p/g12-hall-effect-liquid-water-flow-sensor-switch-meter-counter-dc-5v-125lmin-yfb7~p77565740

[59] “ESP32 Pinout: Everything You Need to Know.” Accessed: Mar. 21, 2024. [Online]. Available: https://www.flux.ai/p/blog/esp32-pinout-everything-you-need-to-know

[60] “How to Level Shift 5V to 3.3V | Random Nerd Tutorials.” Accessed: Mar. 21, 2024. [Online]. Available: https://randomnerdtutorials.com/how-to-level-shift-5v-to-3-3v/

[61] “What Is a Transistor? (Definition, How It Works, Example) | Built In.” Accessed: Mar. 21, 2024. [Online]. Available: https://builtin.com/hardware/transistor

[62] jonk, “Answer to ‘Logic level converter using Transistors,’” Electrical Engineering Stack Exchange. Accessed: Mar. 21, 2024. [Online]. Available: https://electronics.stackexchange.com/a/297092

[63] ON Semiconductor, 'TIP29C Series NPN Silicon Power Transistors Datasheet,' Mouser Electronics, [Online]. Available: https://www.mouser.com/datasheet/2/149/TIP29C-196919.pdf. [Accessed: 21-03-2024]

[64] W. Storr, “Thermistors and NTC Thermistors,” Basic Electronics Tutorials. Accessed: Mar. 21, 2024. [Online]. Available: https://www.electronics-tutorials.ws/io/thermistors.html

[65] “RS PRO Motorised Valve, Ball type , 1000 psi | RS.” Accessed: Mar. 23, 2024. [Online]. Available: https://ie.rs-online.com/web/p/motorised-valves/7605728

[66] T. Agarwal, “Permanent Magnet Stepper Motor : Construction, Working & Applications,” ElProCus - Electronic Projects for Engineering Students. Accessed: Mar. 23, 2024. [Online]. Available: https://www.elprocus.com/what-is-a-permanent-magnet-stepper-motor-its-working/

[67] “H Bridge.” Accessed: Mar. 23, 2024. [Online]. Available: https://encyclopedia.pub/entry/30808

[68] Texas Instruments, "L293 Quadruple Half-H Drivers," Texas Instruments. [Online]. Available: https://www.ti.com/lit/ds/symlink/l293.pdf. [Accessed: Mar. 23, 2024].

[69] “Using a rotary encoder on an Arduino | by Melanie Chow | Medium.” Accessed: Mar. 23, 2024. [Online]. Available: https://medium.com/@melaniechow/using-a-rotary-encoder-on-an-arduino-18f543ae7f78

[70] “Buck Converters (Step-Down Converter).” Accessed: Mar. 23, 2024. [Online]. Available: https://www.monolithicpower.com/en/power-electronics/dc-dc-converters/buck-converters

[71] “GC9A01 1.28-Inch Round LCD TFT Display for Arduino,” AZ-Delivery. Accessed: Mar. 23, 2024. [Online]. Available: https://www.az-delivery.de/en/products/1-28-zoll-rundes-tft-display

[72] “PID Controller: Types, What It Is & How It Works | Omega.” Accessed: Mar. 24, 2024. [Online]. Available: https://www.omega.co.uk/prodinfo/pid-controllers.html

[73] F. M. Callier and C. A. Desoer, “Unity Feedback Systems,” in *Linear System Theory*, F. M. Callier and C. A. Desoer, Eds., New York, NY: Springer, 1991, pp. 356–402. doi: 10.1007/978-1-4612-0957-7_15.

[74] N. Agnihotri, “Arduino compatible coding 15: Reading sensor data from DHT-11 without using a library,” Engineers Garage. Accessed: Mar. 25, 2024. [Online]. Available: https://www.engineersgarage.com/articles-arduino-dht11-humidity-temperature-sensor-interfacing/

[75] “How an LDR (Light Dependent Resistor) Works,” Kitronik Ltd. Accessed: Mar. 25, 2024. [Online]. Available: https://kitronik.co.uk/blogs/resources/how-an-ldr-light-dependent-resistor-works

[76] “Electronic waste (e-waste).” Accessed: Mar. 25, 2024. [Online]. Available: https://www.who.int/news-room/fact-sheets/detail/electronic-waste-(e-waste)

[77] “Interface a 16x2 LCD Display with evive,” STEMpedia Education. Accessed: Mar. 25, 2024. [Online]. Available: https://ai.thestempedia.com/docs/evive/evive-tutorials/how-to-interface-16-x-2-lcd-with-evive/

[78] C. T. Liu, “Revolution of the TFT LCD Technology,” *Journal of Display Technology*, vol. 3, no. 4, pp. 342–350, Dec. 2007, doi: 10.1109/JDT.2007.908348.

[79] “TFT vs LCD Display: which is better? | Hongguang Display.” Accessed: Mar. 26, 2024. [Online]. Available: https://www.hongguangdisplay.com/blog/tft-vs-lcd-display-which-is-better/

[80] D. J. R. Cristaldi, S. Pennisi, and F. Pulvirenti, “Passive LCDs and Their Addressing Techniques,” in *Liquid Crystal Display Drivers: Techniques and Circuits*, D. J. R. Cristaldi, S. Pennisi, and F. Pulvirenti, Eds., Dordrecht: Springer Netherlands, 2009, pp. 75–108. doi: 10.1007/978-90-481-2255-4_3.

[81] D. J. R. Cristaldi, S. Pennisi, and F. Pulvirenti, “Passive LCDs and Their Addressing Techniques,” in *Liquid Crystal Display Drivers: Techniques and Circuits*, D. J. R. Cristaldi, S. Pennisi, and F. Pulvirenti, Eds., Dordrecht: Springer Netherlands, 2009, pp. 75–108. doi: 10.1007/978-90-481-2255-4_3.

[82] P. Sabharwall, V. Utgikar, and F. Gunnerson, “Effect of Mass Flow Rate on the Convective Heat Transfer Coefficient: Analysis for Constant Velocity and Constant Area Case,” *Nuclear Technology*, vol. 166, no. 2, pp. 197–200, May 2009, doi: 10.13182/NT09-A7406.

[83] “Balancing Radiators Explained | PlumbHQ Hub,” PlumbHQ. Accessed: Mar. 26, 2024. [Online]. Available: https://plumbhq.uk/a/central/articles/plumbhq.uk

[84] D. C. P. Department and OpenStax, “5.4 Temperature Change and Heat Capacity,” Aug. 2016, Accessed: Mar. 26, 2024. [Online]. Available: https://pressbooks.bccampus.ca/introductorygeneralphysics2phys1207/chapter/14-2-temperature-change-and-heat-capacity/

[85] “Mass Flow Rate: Equation, Units & Conversions | StudySmarter,” StudySmarter UK. Accessed: Mar. 27, 2024. [Online]. Available: https://www.studysmarter.co.uk/explanations/engineering/engineering-fluid-mechanics/mass-flow-rate/

[86] M. Sandalci, E. Mancuhan, E. Alpman, and K. Küçükada, “Effect of the flow conditions and valve size on butterfly valve performance,” *0 Isı bilimi ve tekniği dergisi = Journal of Thermal Sciences and Technology*, vol. 30, pp. 103–112, Jan. 2010.

[87] M. Teahon, "RBSi Heat Loss Training," May 2020. [Online]. Available: https://berassessors.com/wp-content/uploads/2020/05/RBSi-Heat-Loss-Training-MIKE-TEAHON.pdf. Accessed: 26 March 2024.

[88] N. Earth Science Data Systems, “Air Mass/Density | Earthdata.” Accessed: Mar. 26, 2024. [Online]. Available: https://www.earthdata.nasa.gov/topics/atmosphere/atmospheric-pressure/air-mass-density

Chapter 10 - Appendices

Drive link to appendices: https://drive.google.com/drive/folders/1Ei1lyCnDaHAZ-hyZpY-xhXMcUAfPjDlN?usp=sharing

Videos not in this document: https://drive.google.com/drive/folders/1ed4UnlYj851x4N5iKKzbFaPraWgK0KeS?usp=drive_link

Appendix A – Devices, Home Assistant, Portainer, Node-RED

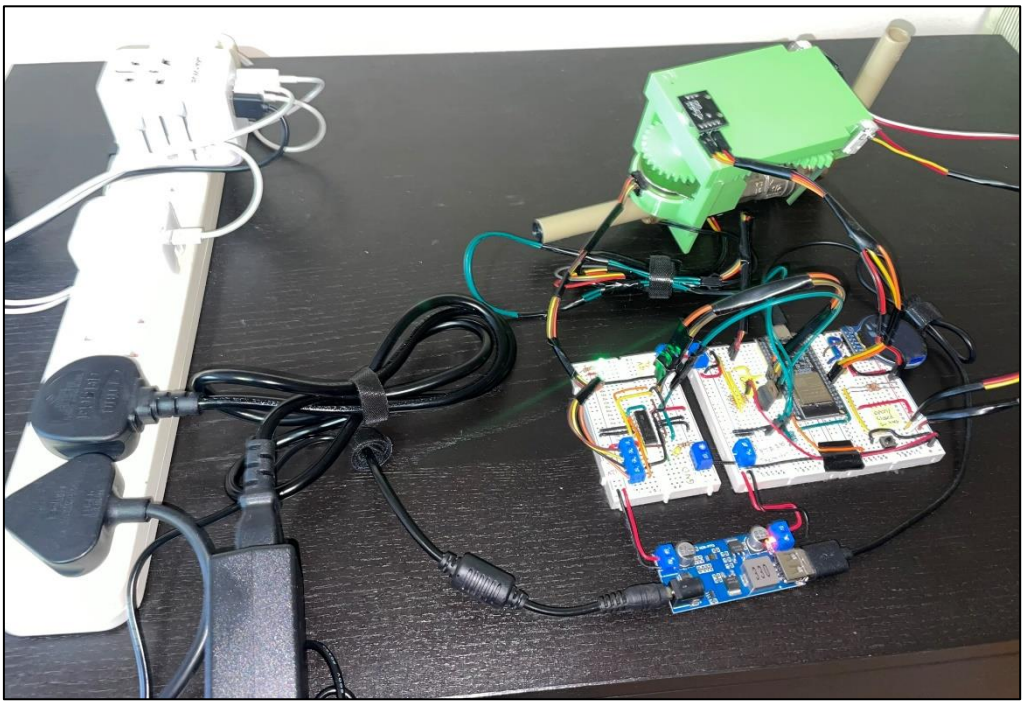


Figure 10.1 - Water control device - power supply, circuitry and mechanical aspects

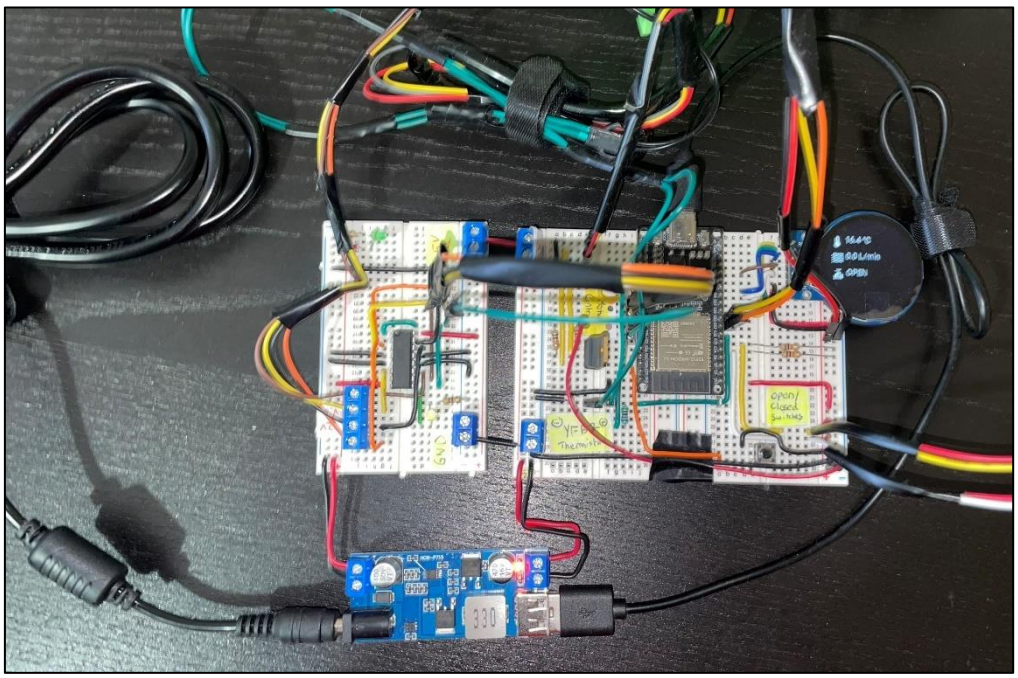


Figure 10.2 - Water control circuitry

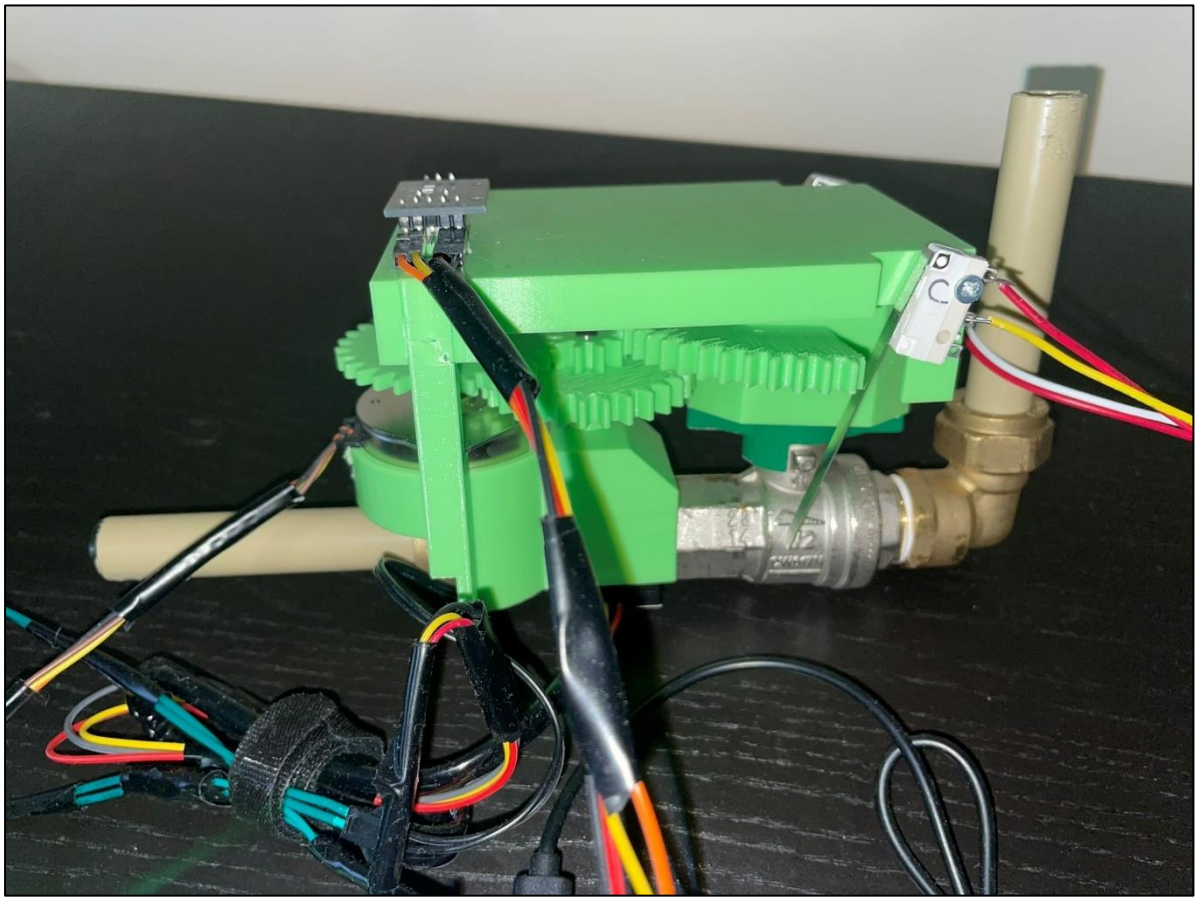


Figure 10.3 - Water control mechanical

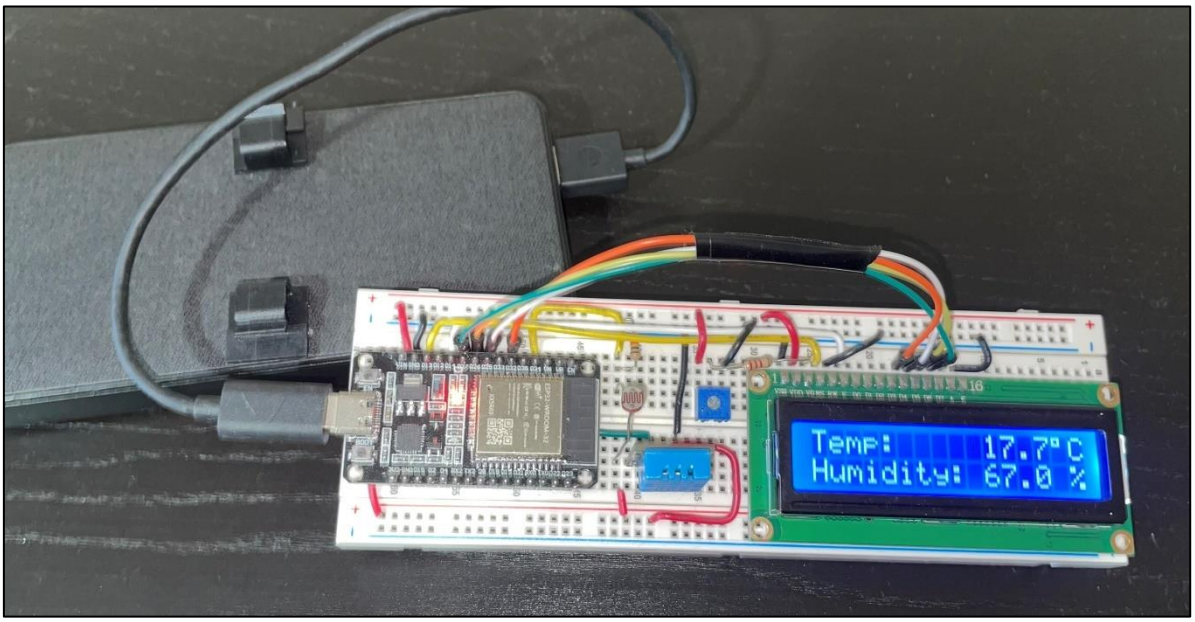


Figure 10.4 - Temperature sensor connected to battery

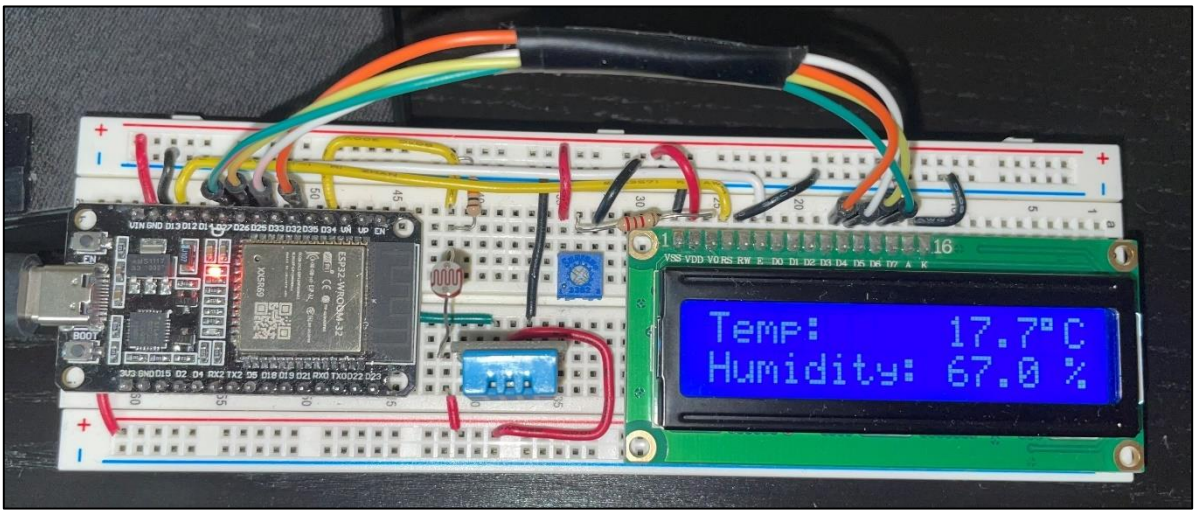


Figure 10.5 - Temperature sensor circuitry

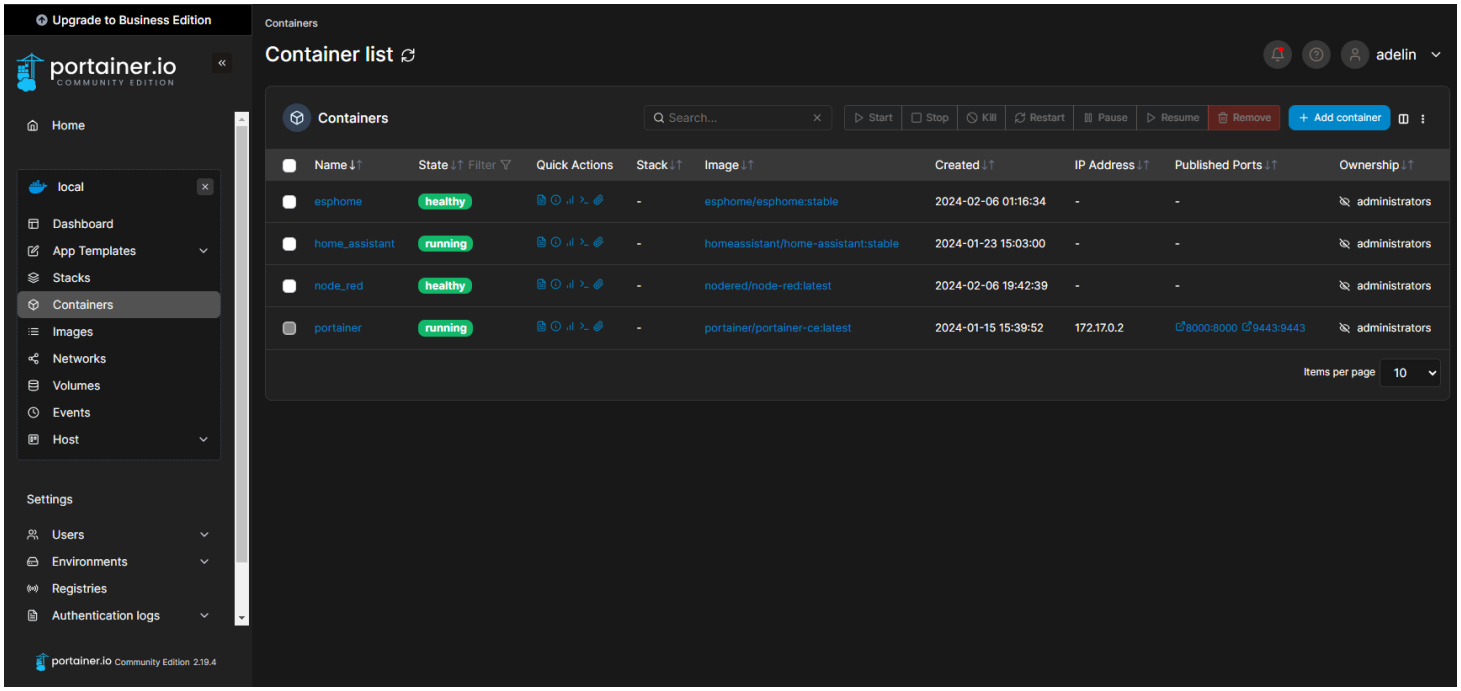


Figure 10.6 - Portainer Dashboard

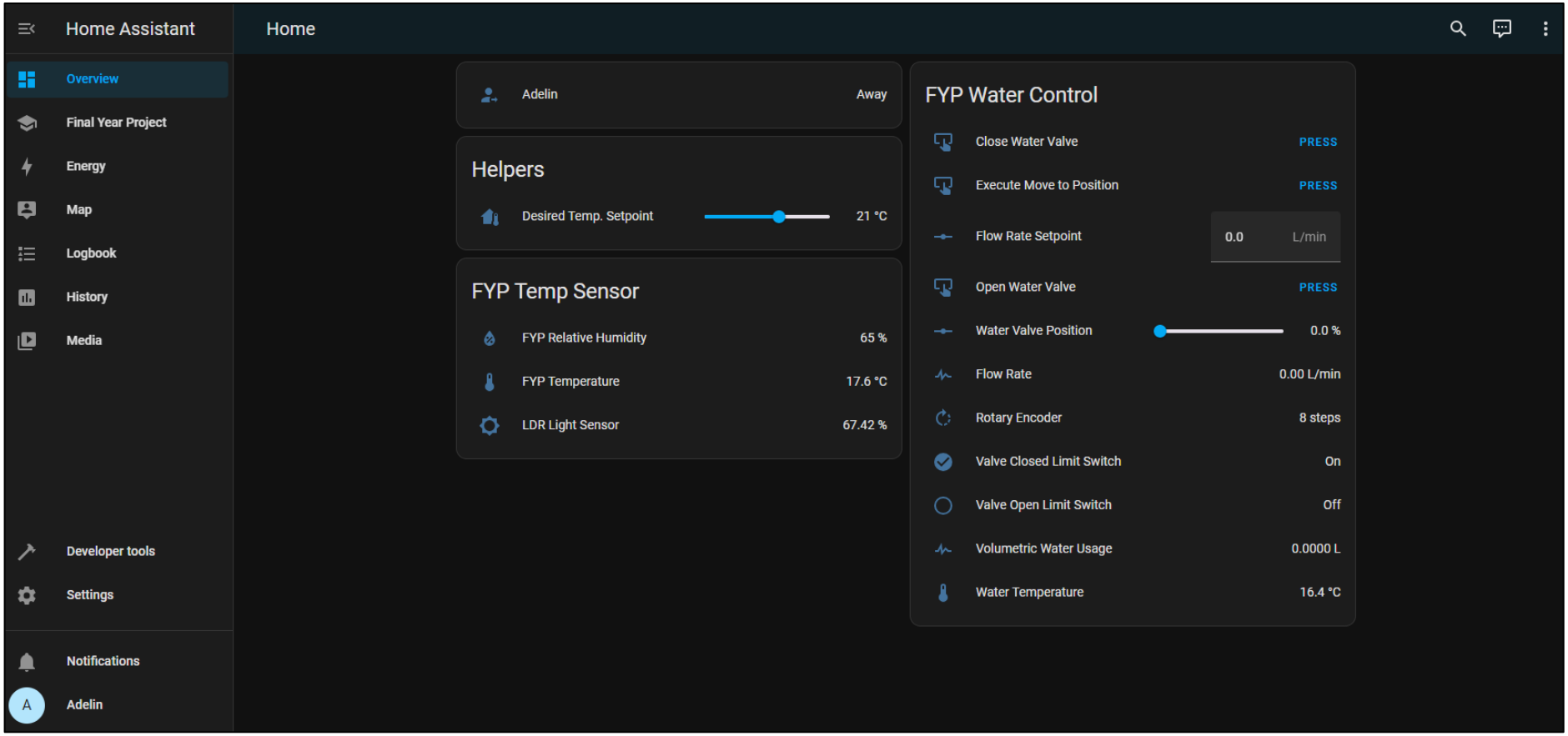


Figure 10.7 - Home Assistant Dashboard

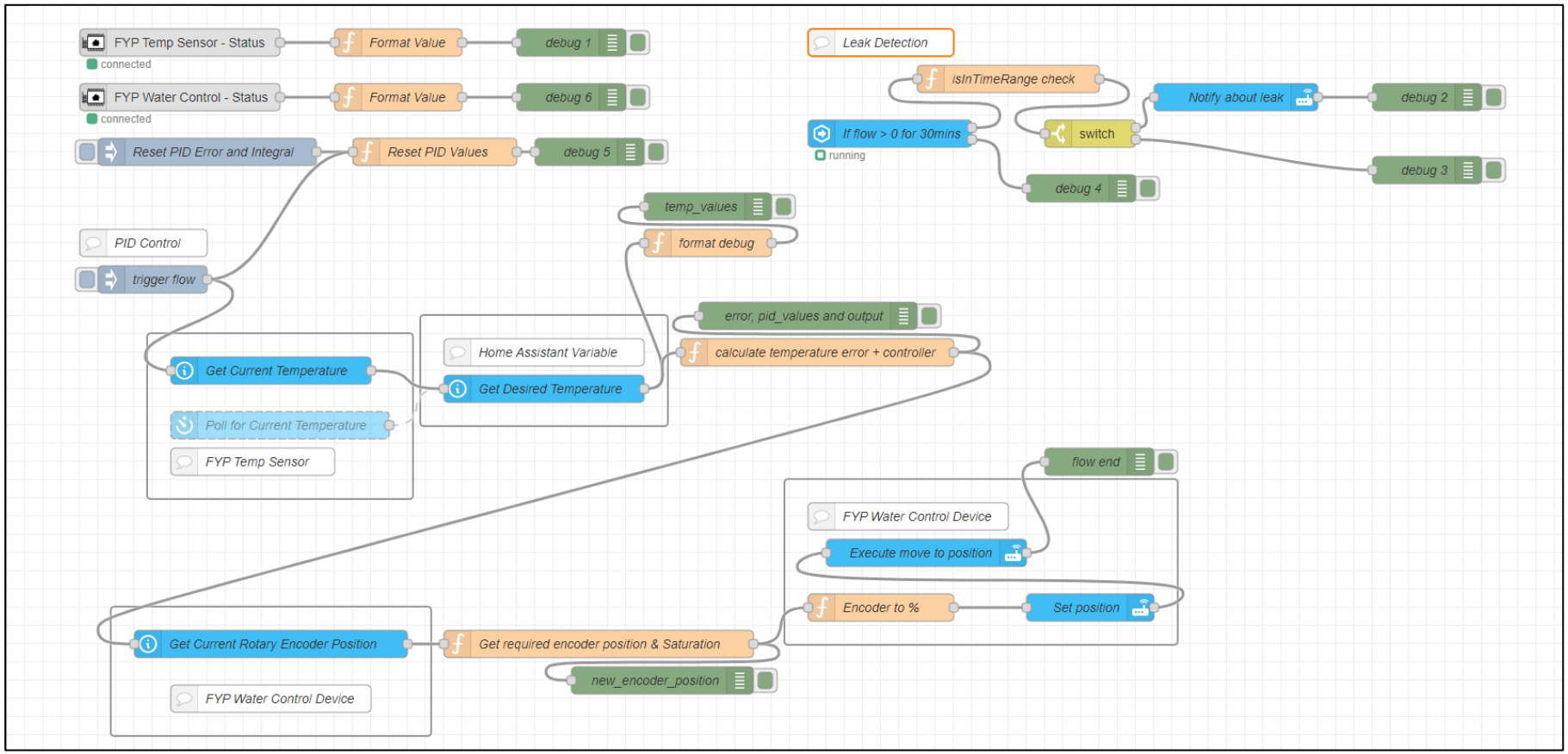


Figure 10.8 - Entire Node-RED flow

Appendix B – Software

Full ESPHome Code for FYP Water Control Device

Full ESPHome Code for FYP Temperature Sensor Device

Full Code Snippets from Node-RED

Entire Folder Containing YAML and JavaScript files

Appendix C – SolidWorks Parts Drawings

Final zip assembly: https://drive.google.com/drive/folders/1t3XDmWTNjU9x0RiV3Ty-MPzRi0VG3OsG?usp=sharing

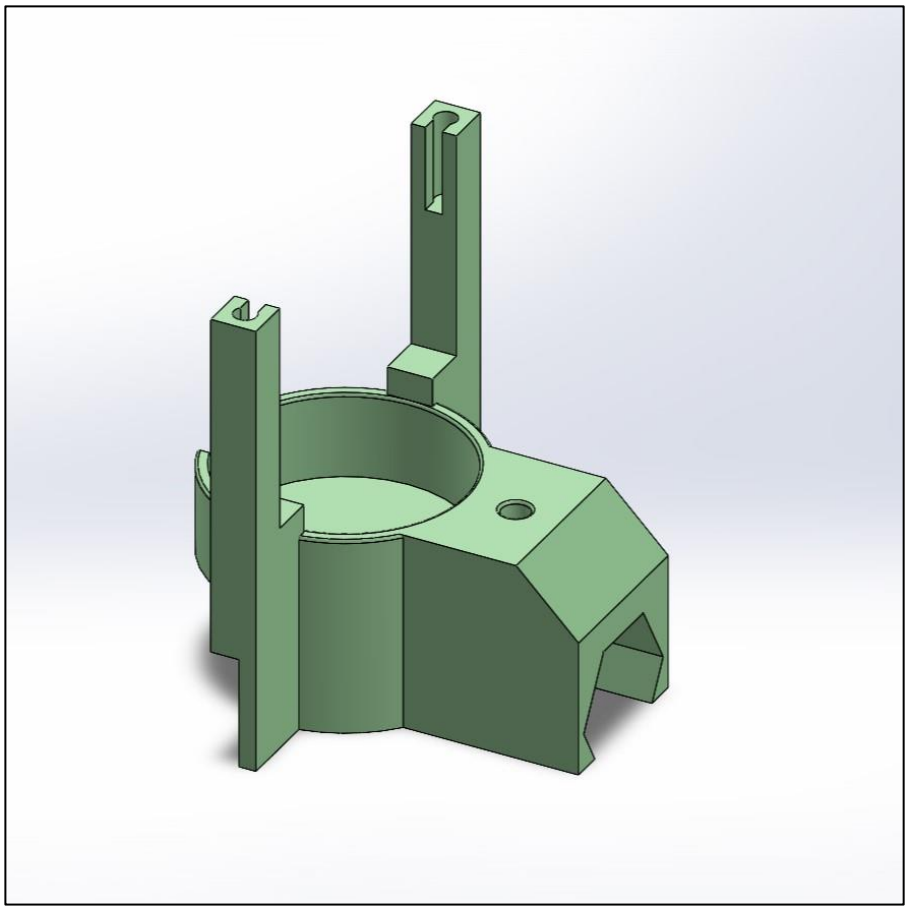


Figure 10.9 - Casing motor holder

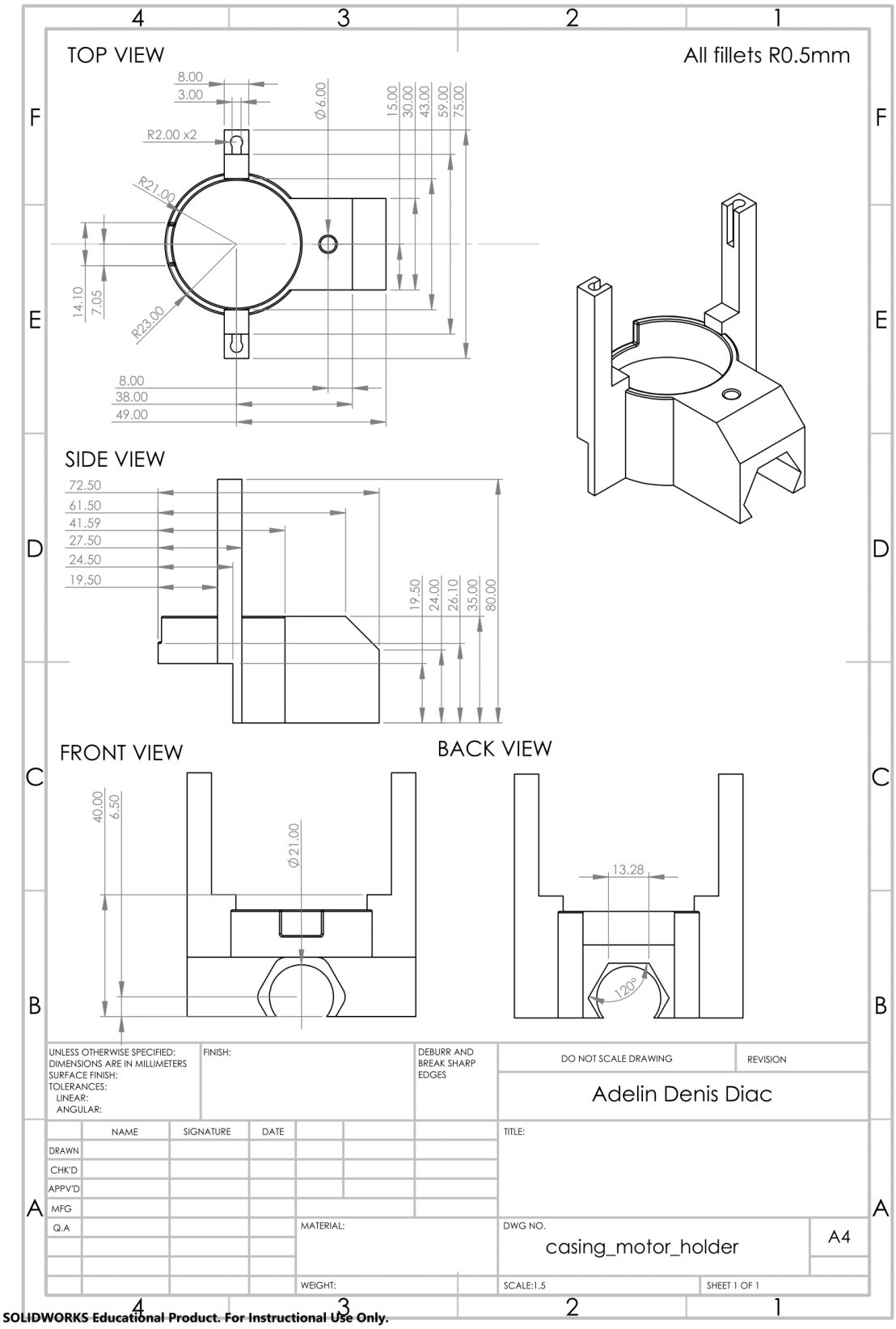


Figure 10.10 - Casing motor holder drawing

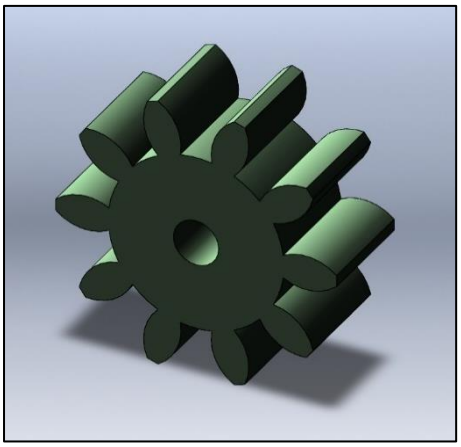


Figure 10.11 - Motor gear

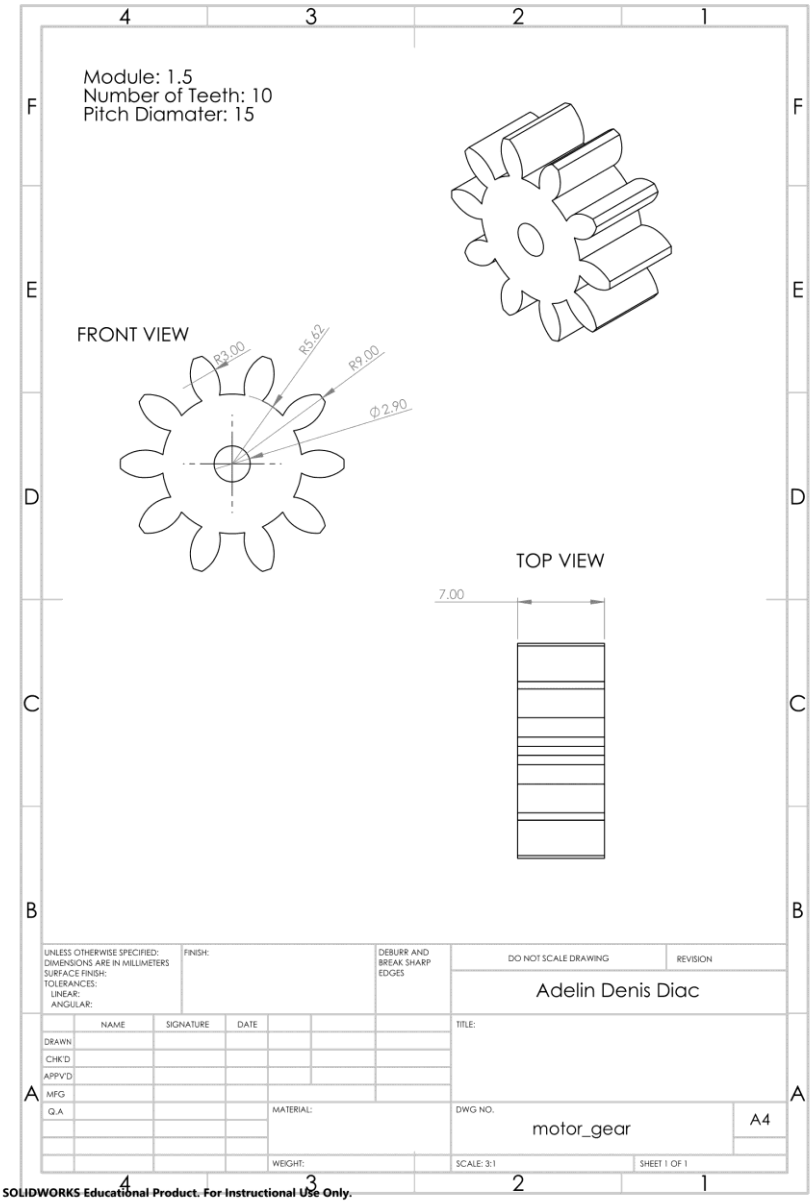


Figure 10.12 - Motor gear drawing

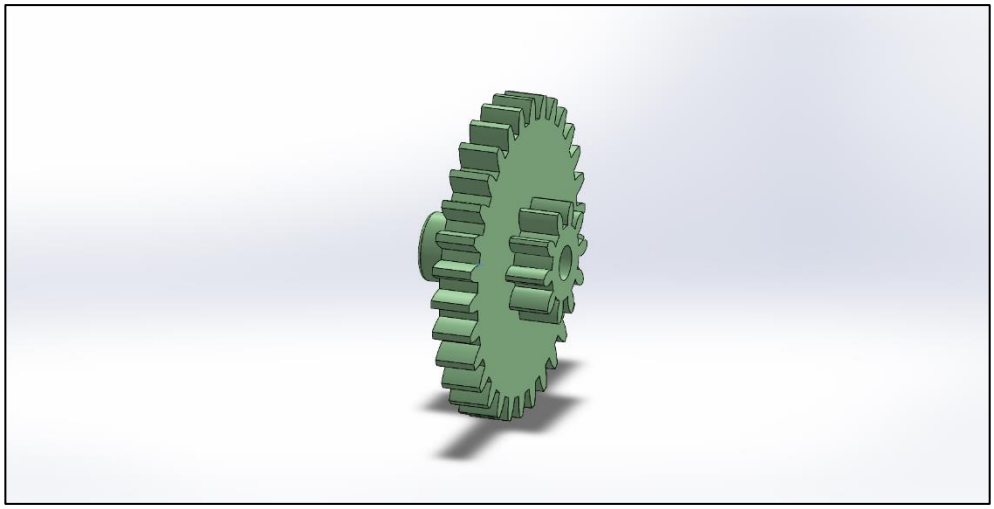


Figure 10.13 - Intermediary gear

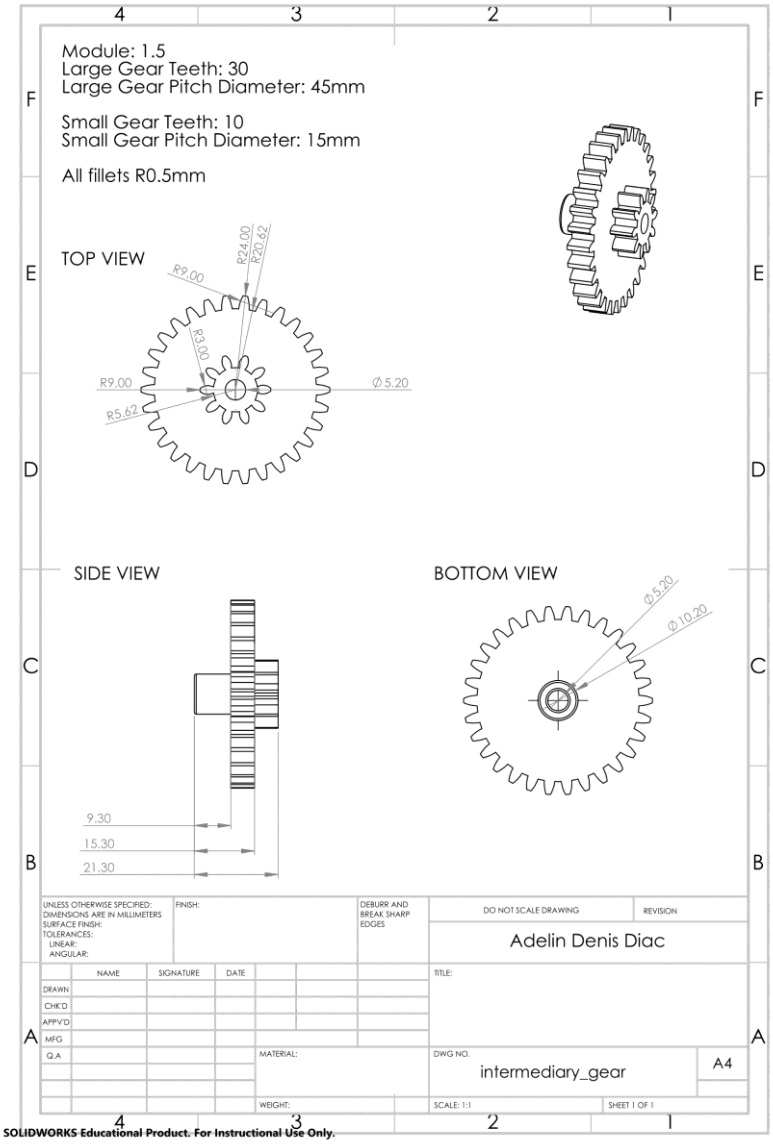


Figure 10.14 - Intermediary gear drawing

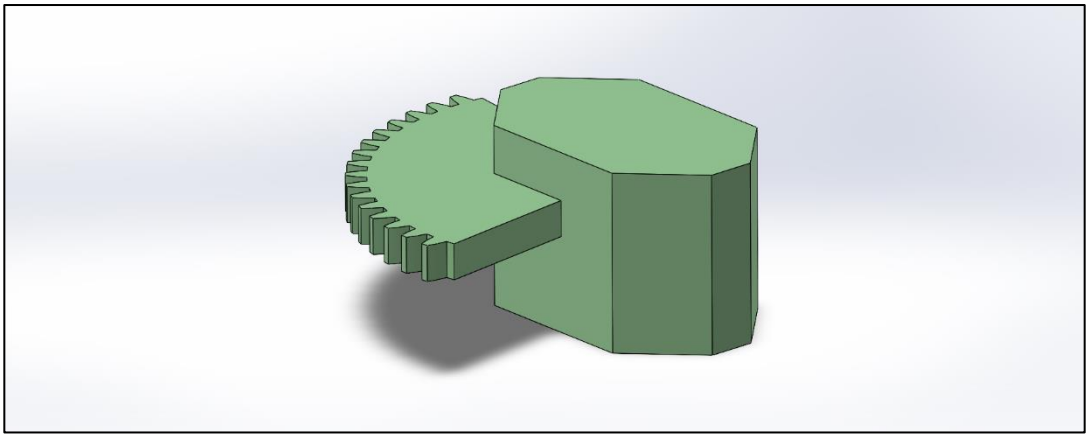


Figure 10.16 - Ball valve cap

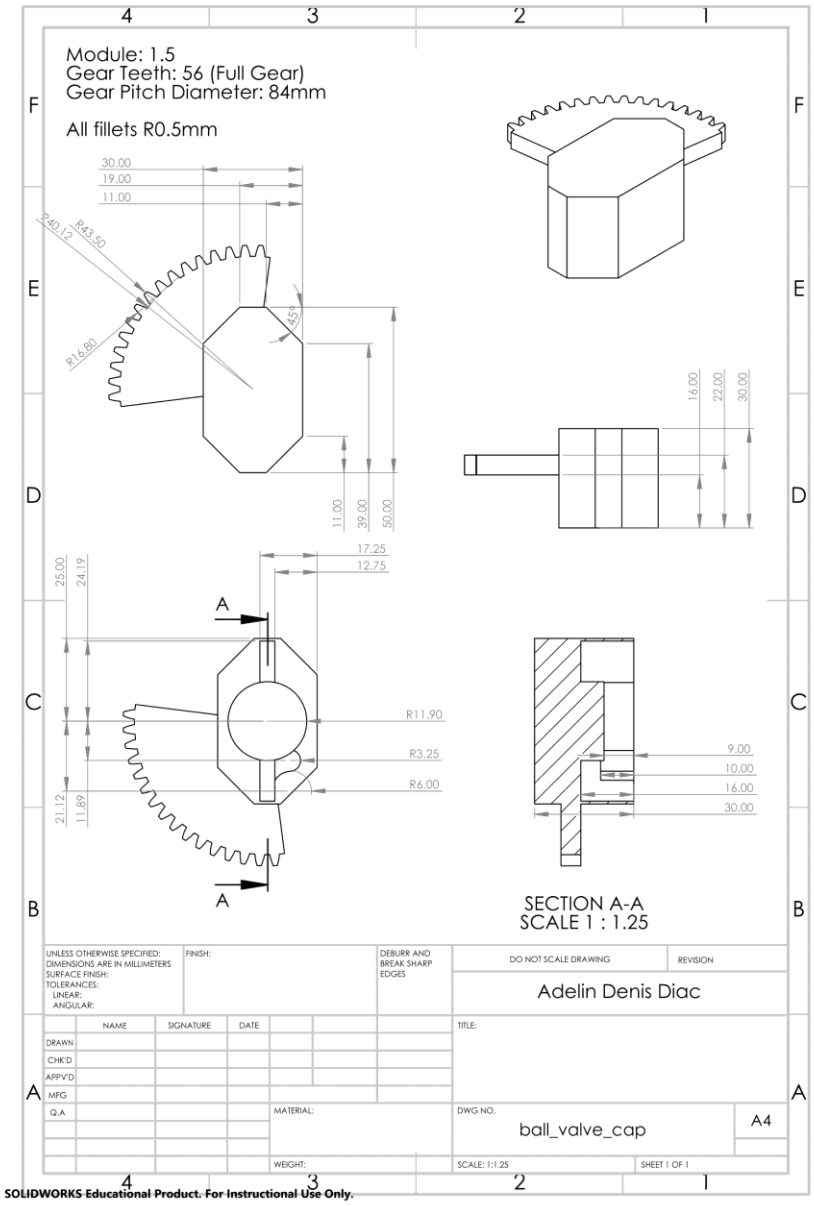


Figure 10.15 - Ball valve cap drawing

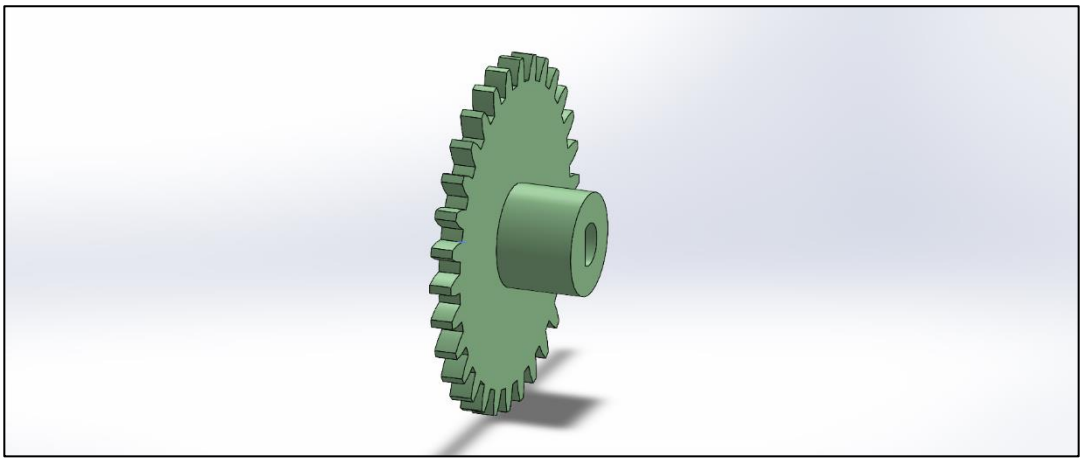


Figure 10.17 - Rotary Encoder Gear

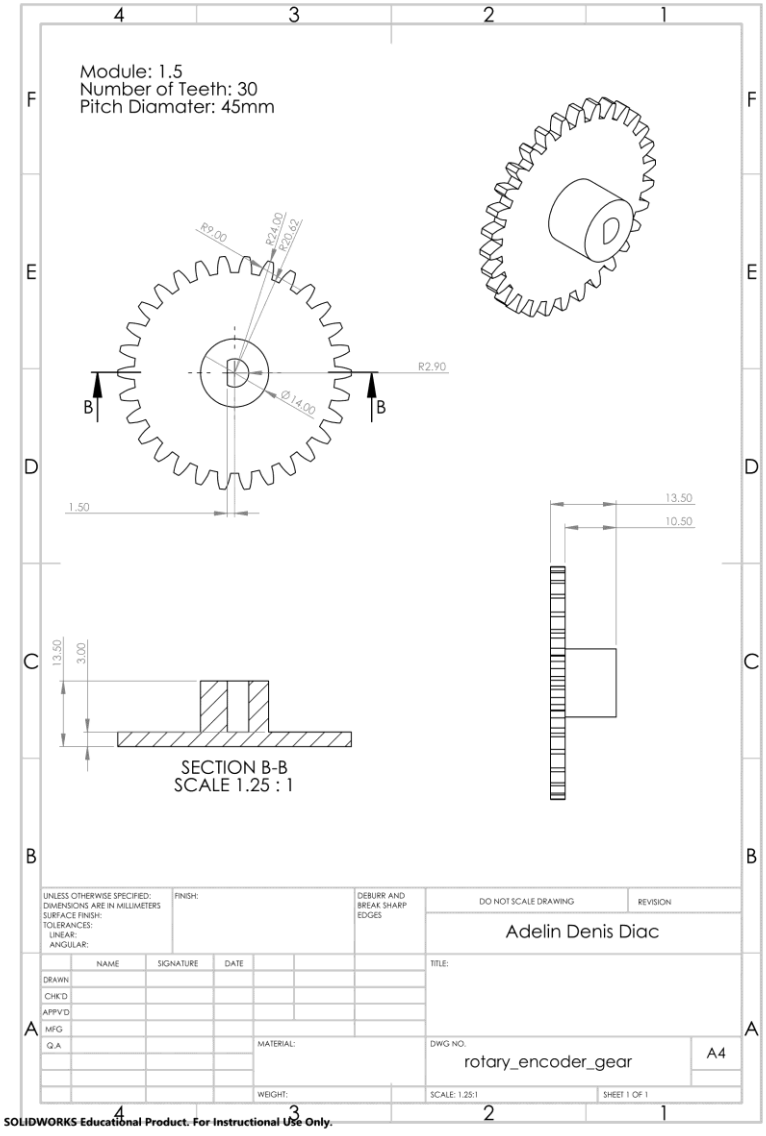


Figure 10.18 - Rotary encoder gear drawing

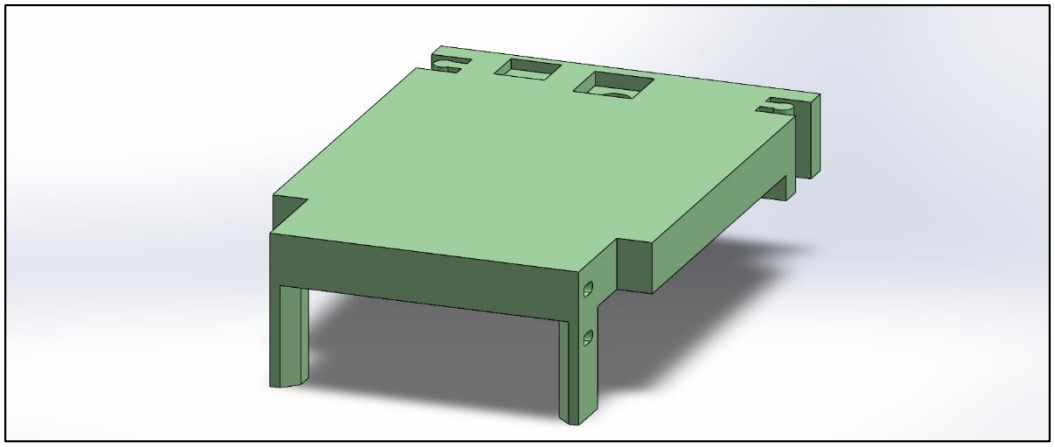


Figure 10.19 - Casing top part

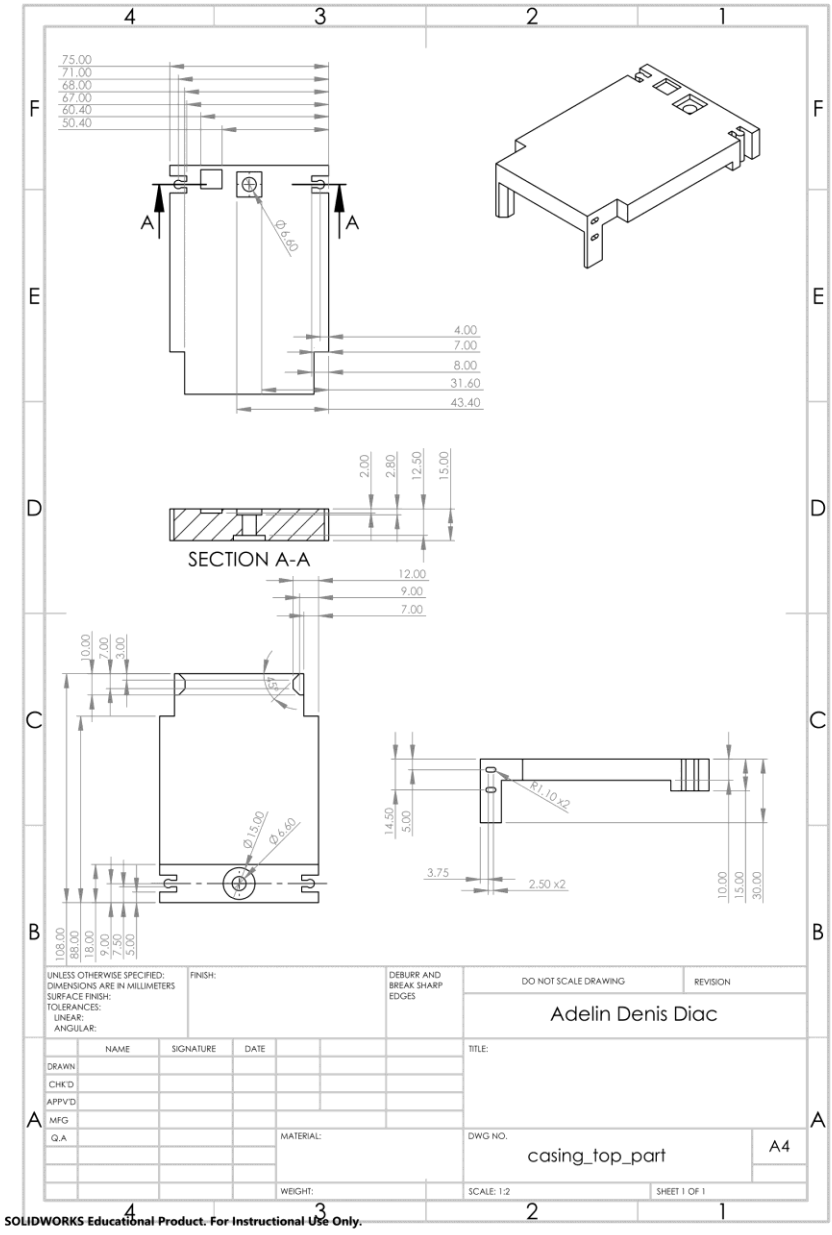


Figure 10.20 - Casing top part drawing

Appendix D – Circuit Diagrams

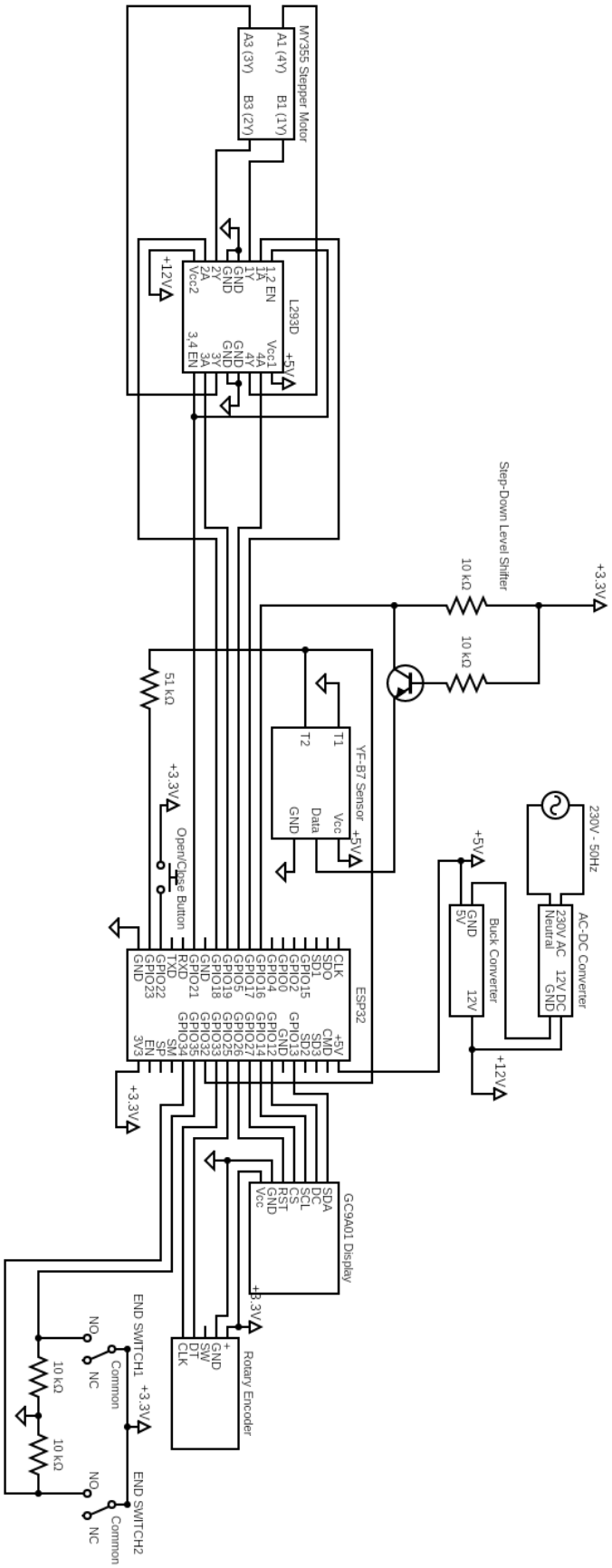


Figure 10.21 - Flow sensing and control (entire circuit diagram)

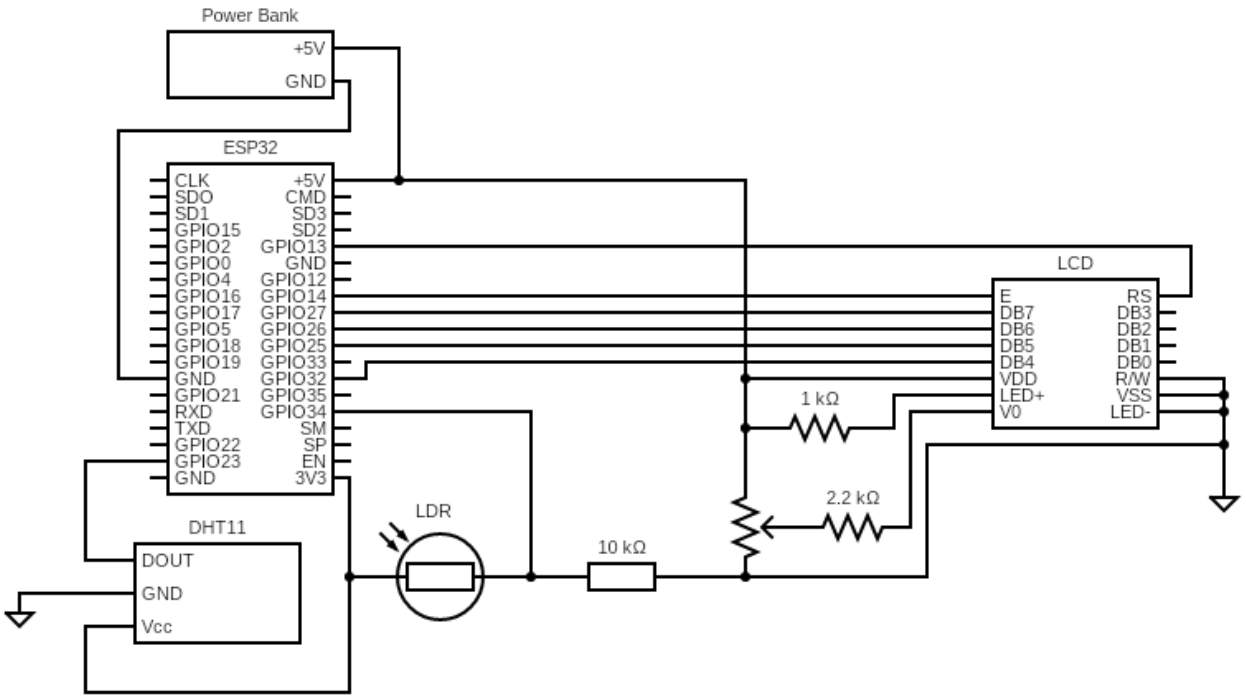


Figure 10.22 - Temperature sensor (entire circuit diagram)

Appendix E – Testing Results

Table 10.1 - 0.5 litres accuracy test

Litres Used	Litres Measured	Error	Percentage Error (%)
0.5	0.3674	0.1326	26.52
0.5	0.3605	0.1395	27.9
0.5	0.3418	0.1582	31.64
0.5	0.356	0.144	28.8
0.5	0.3455	0.1545	30.9
0.5	0.3403	0.1597	31.94
0.5	0.3431	0.1569	31.38
0.5	0.3479	0.1521	30.42
0.5	0.3561	0.1439	28.77
0.5	0.3427	0.1573	31.45

Table 10.2 - 1-litre accuracy test

Litres Used	Litres Measured	Error	Percentage Error (%)
1	0.6994	0.3006	30.06
1	0.6961	0.3039	30.39
1	0.6842	0.3158	31.58
1	0.6913	0.3087	30.87
1	0.7031	0.2969	29.69
1	0.6989	0.3011	30.11
1	0.6931	0.3069	30.69
1	0.7019	0.2981	29.81
1	0.7028	0.2972	29.72
1	0.6815	0.3185	31.85

Table 10.3 - 1.5 litres accuracy test

Litres Used	Litres Measured	Error	Percentage Error (%)
1.5	1.0447	0.4553	30.35
1.5	1.0371	0.4629	30.86
1.5	1.0267	0.4733	31.55
1.5	1.0280	0.4720	31.47
1.5	1.0268	0.4732	31.55
1.5	1.0355	0.4645	30.96
1.5	1.0373	0.4627	30.85
1.5	1.0461	0.4539	30.26
1.5	1.0266	0.4734	31.56
1.5	1.0347	0.4653	31.02

Table 10.4 - 2 litres accuracy test

Litres Used	Litres Measured	Error	Percentage Error (%)
2	1.3854	0.6146	30.73
2	1.3799	0.6201	31.00
2	1.3826	0.6174	30.87
2	1.3721	0.6279	31.39
2	1.3873	0.6127	30.63
2	1.3820	0.6180	30.90
2	1.3802	0.6198	30.99
2	1.3647	0.6353	31.77
2	1.3678	0.6322	31.61
2	1.3835	0.6165	30.83

Table 10.5 - 2.5 litres accuracy test

Litres Used	Litres Measured	Error	Percentage Error (%)
2.5	1.7098	0.7902	31.61
2.5	1.7259	0.7741	30.96
2.5	1.7111	0.7889	31.55
2.5	1.7149	0.7851	31.40
2.5	1.7197	0.7803	31.21
2.5	1.7260	0.7740	30.96
2.5	1.7284	0.7716	30.86
2.5	1.7164	0.7836	31.34
2.5	1.7244	0.7756	31.02
2.5	1.7032	0.7968	31.87

Appendix F – Datasheets

16x2 LCD Display

19N403L63 Limit Switch

28BYJ-48 5V Stepper Motor

Astrosyn 12V Stepper Motor

DHT11 Temperature and Humidity Sensor

ESP32-WROOM32

GC9A01A LCD Display

KY-040 Rotary Encoder

L293D Stepper Motor Driver

Shut-off Ball Valve

TIP29C Transistor

YF-B7 Flow Sensor

AC-DC Transformer Power Supply

12-24V to 5V Buck Converter