Exploring real time data collection and parameter monitoring in an Internet of Things system

Author: Henry David Rubiano Poveda
Supervisor: Prof. Thomas Thomson

27th April 2021

Abstract

The Internet of Things has grown at an exponential rate over the past few years. The technological progress in the field has allowed us to integrate even further the Internet into almost every single area of society. Thanks to IoT we can now connect "things" to the Internet by embedding small devices, which can collect data, communicate between each other or even make decisions without almost any kind of human intervention. This IoT project explores data collection and monitoring by further developing and implementing a system able to collect data from a variety of sensors and send it to a database to store it over time. The project will go through different stages in which different areas will be explored and different solutions will be developed. First, the project will explore the range of hardware components to be used, along with the different software and database system. Then, it will emphasise on the importance of storage optimisation on devices with limited resources, following with the implementation of an intelligence system able to monitor the data and make decisions accordingly, in order to preserve the well-being of the system. The project will finish with the implementation of different logging systems to add an extra layer of reliability in terms of information storage.

Acknowledgements

First, I would like to thank my supervisor Professor Thomas Thomson. His valuable help and support have been very important for me, and it was with his orientation and assistance that I was able to successfully develop this project.

I would also like to thank my family and friends, who have helped me during these last 3 years and keep supporting me along the way. Thanks to them I have been able to get to this point, and I am forever grateful for their support.

Contents

1 Introduction

1.1 Aims and Objectives

The Internet of Things is rapidly changing how we interact with the world around us, providing us with new more efficient ways of approaching real world problems by integrating technology in many more different areas of society. It is clear that in this new era, data is becoming a very valuable asset for many organisations, as it allows them to have more control over the decisions they choose to take. It also helps them keep track of many different vital system parameters over time, which will certainly be helpful, as they will be able to learn about their product using the experience that this data will provide them. This is why the collection of real time data is becoming more and more important as technology advances and becomes a crucial part of our daily life.

The main objective of this third year project is to explore the collection, pre-processing and transfer of real time data in an Internet of Things system, by further designing and implementing a system able to monitor parameters efficiently and effectively in a diverse range of instrumentation and equipment, while making decisions intelligently according to the different kinds of data collected from the environment.

1.2 Structure

The structure of the report will consist on the following sections, that will explore in more detail some of the key aspects of this project:

- First, the report will try to give a comprehensive definition and background on the history of the Internet of Things, apart from describing its main purpose. This will be very important in order to properly understand the main motivations of the project. In this section the different applications of this field and its system architecture will also be explained.

- The second section will consist on the Design and Approach, which will go through the hardware components selection and some of the most important aspects of the project, such as the concept of storing the specific type of data we are working with (time series data), the visualisation platform that will be used, and a brief overview of the security threats and features of the project.

- We will follow the previous section with the Implementation, where we will look in more detail the techniques used to expand on the previous developed version, which will consist of the new features of data collection, memory optimisation, data monitoring and data logging.

- We will finish the report reflecting on the project, and arriving to conclusions that will certainly be useful for future further developments and progress.

2 Background

2.1 The Internet of Things

2.1.1 Definition

According to the Oxford Dictionary, the Internet of Things (IoT) is "the interconnection via the Internet of computing devices embedded in everyday objects, enabling them to send and receive data". We should not confuse the Internet of Things with smart technology though, as this technology only refers to any device that is able to connect to the Internet. While we need to be present in order for a smart device (such as a smartphone) to use the technology and hence connect to the Internet, an IoT device could be accessed and controlled from anywhere at anytime [1].

In order to talk about the Internet of Things, we must talk about the meaning of a "thing". The only requirement for a thing to be IoT-enabled is just to be able to connect to the Internet and communicate with others while being able to be accessed and controlled from anywhere and anytime, as mentioned before. This means that the personal and business possibilities of these "things" are almost endless. They could refer to anything from a connected medical device, a biochip transponder, to a solar panel, an "intelligent" car with sensors that alert the driver of many parameters to take into account (fuel, tire pressure, needed maintenance, and more) or any object, with sensors embedded in it, that has the ability to collect and transfer data over a network [2]. These "things" exist in the physical world, but we can also have virtual "things", which exist in the information world and may or may not be associated with a physical "thing".

2.1.2 History

The term "IoT", used for the first time by Kevin Ashton in his presentation at Procter and Gamble (P&G) in 1999 when trying to link a new idea by P&G's supply chain to the new and upcoming topic of "the Internet" was only the beginning of a whole new field that would keep expanding more and more rapidly every year, appearing everywhere from the Scientific American journal to a conference in Europe [3]. The announcement by LG of a smart fridge that would detect if products are or not full in 2000 was one of the first efforts by companies to have this kind of technology integrated, but it would be almost a decade later, in 2008, when the ISPO (Internet Protocol for Smart Objects) alliance started promoting the use of IP addresses in IoT communications, developing standards and promoting its use all over the world. Since then, the growth of the interest in the field has been nothing short of unprecedented [4].

This field of computer science has been growing at a really high rate the last few years, as the year 2020 was the first where there were more IoT connections (smart homes, cars, industrial equipment...) than non-IoT connections (smartphones, laptops and computers), and Cisco predicts that there will be 500 billion devices connected to the Internet by 2030. Despite Covid -19, almost 50% of organisations are planning to increase their investments in IoT [5], and it is predicted that its market value will reach around USD 81 billion by 2026

[6].

2.1.3 Architecture

The IoT system architecture is usually described as a four part process, in which data is collected from endpoints (all kinds of sensors) and after some pre-processing they finally arrive at a data center where it is properly processed, analysed, and stored. We must also take into account that data or instructions can flow in the opposite direction (from the data center to the endpoints) in order to make decisions about actions to take, usually according to the data previously received. Now we will go through each stage, which will also be taken into account when developing this project [7][8]:

- The first part would consist of the sensors and the actuators. Those will be in charge of collecting the data from whichever object they might be embeeded in, with parameters such as chemical composition, temperature, humidity or more. The actuators will be the ones able to decide and take actions based on the information previously collected by the sensors, like opening or closing the curtains of a room, depending on if the sun is pointing right at the window.

- The second part consists of the Sensor Data Acquisition Systems. These systems will be able to perform tasks such as converting the data into the right format or compressing it to reduce its size.

- The third stage would be the pre-processing of the data at the endpoint side. Tools can be used for this stage, like machine learning or some kind of intelligence system able to make some quick decisions without having to wait for the data center to give the instructions. We want to always make sure that the intelligence is close to the endpoints, instead of close to the data center.

- The last part of the architecture will be focused on the analysis, visualisation and storage of the data collected. This will be performed by the data center, which can be in a physical location, or in the cloud. The analysis of the data center will be much deeper than the one in the pre-processing part, as business rules and specific analysis will be taken place there.

2.1.4 Applications

Internet of Things' systems have proven to be able to be very important tools within a very wide range of areas in society, going from consumer to industrial and infrastructure applications:

- Some of the consumer applications of IoT include everything from home automation systems that monitor lights, temperature, home appliances and even home security to self-driving cars or wearable technology such as fitness trackers that are able to monitor your health just by wearing them, etc.

- Industrial applications of IoT have been very beneficial for most of the economic sectors; for example, primary sector activities such as agriculture are being automated and made more efficient, with systems which collect data from the environment (which is similar to what this project is about) and decide on which techniques are more appropriate for the land. The supply chain industry has also been revolutionised, as now it's a lot easier to figure out where the goods are at any point in time, hence being able to make accurate timelines, making the traffic a lot smoother and more efficient [9].

- When we want to talk about large-scale infrastructure applications of IoT we must mention the new Smart Cities, with useful integrations of IoT ranging from more efficient public transformation systems (with real-time information on times and routes and an easier ticket purchasing system) to an automated and smart healthcare system (with an IoT system facilitating data collection and thus improving its administration) [10].

3 Design and Approach

3.1 Project Structure

This project will consist on different stages that must be achieved in order to gain the necessary knowledge to develop a product that can be properly applied to a specialist real world application:

- We will first need to understand the IoT system architecture developed in a previous project, which will serve as the foundation for our monitoring system.

- We will then need to develop the functionality for this system, focusing on key aspects such as how the data is collected, how this data can be stored, and also how the data can be transferred through the network to the database.

- In order to make the system "smarter" we will need to focus on making the appropriate decisions in order to maintain the well-being of the system, which is the principal objective of parameter monitoring.

- We will also need to focus on enhancing data reporting and visualisation, in order to show the monitoring status of the different devices of the IoT network, and in data logging, important to store data locally (and therefore keeping the last pieces of information collected safe temporarily) for unexpected cases that could arise like loss of power or the impossibility of transferring the data through the network.

3.2 System Architecture

3.2.1 Overview of the system architecture

This project will consist of one or multiple microcontrollers that will periodically collect data from sensors. Once multiple readings of the data have been collected, the microcontrollers will be in charge of analysing the data, pre-processing it, and sending it to a time series database (InfluxDB). When analysing the data, the microcontrollers will be able to make quick decisions, thanks to their monitoring system, which will check the well-being of the device. The status of the monitoring device will be easily available thanks to an integrated OLED display. The data stored over time at InfluxDB will be easily available for monitoring thanks to the Grafana visualisation platform. In order to avoid possible loss of data, logging will be implemented in various ways, such as the use of flash memory or an external memory card. Here we can have a look at a diagram about the system:

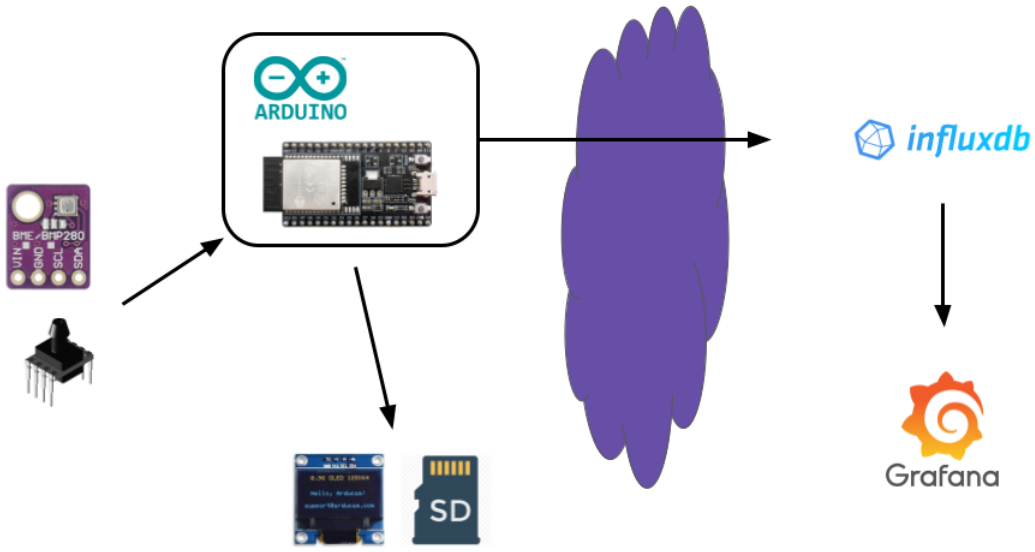

Figure 1: Simple diagram describing the system architecture

3.2.2 Hardware components selection

The decision of the microcontroller board and the other hardware components will be crucial for the best possible and most efficient system. Therefore, we need to be able to define which of them are the most adequate for this project.

In the previous project, there were sketches for the use of an Arduino compatible microcontroller board (ESP32) and a DHT22 temperature and humidity sensor. We will consider those choices plus others in the comparison.

Choosing the right hardware components for an IoT system is a very important decision to make, as we want to make sure that the different components will be the most useful for our

specific needs and that the system will run as efficiently as possible. We must take many factors into consideration, such as scalability, functionality, cost, power requirements, etc [11].

Microcontroller: The most important decision will be choosing the appropriate microcontroller / single-board computer, as this device will power the system and will do most operations with the data. The first important choice was between a microcontroller and a single-board computer (such as a Raspberry Pi). While a Raspberry Pi is more powerful and has its own Operating System, there was no real purpose for its use, as an Arduino microcontroller and its open hardware nature allows for multiple different types of sensors and embedded systems to be connected to the microcontroller thanks to its GPIO pins. We must also take into account that while the microcontroller cannot handle intensive tasks like the Raspberry Pi can, it also uses a lot less power, which is important for this project, as it is running uninterruptedly. Here we can see the different microcontrollers considered in the decision:

- The first option considered was the Particle Photon. This microcontroller had a STM32F205RGY6 120Mhz ARM Cortex M3, with 1MB of flash memory and 128KB of RAM. It had 18 mixed-signal GPIO pins and its recommended power supply was from 3.6V to 5.5V.

- The second option was the ESP32 microcontrollerby Espressif. It was powered by a Tensilica Xtensa Dual-Core 32-bit LX6 microprocessor, running at 160MHz, with 4MB of flash memory and 520 KB of SRAM. It had 34 GPIO pins with multiple functionalities and run with a recommended power supply from 2.3V to 3.6V.

- The last option considered was an Arduino UNO WiFi Rev2, with a ATMEGA4809 processor, running at 16MHz. It had 48KB of flash memory, and 6KB of SRAM. It only has 14 GPIO pins and its recommended power supply is from 7V to 12V.

After looking at the different possibilities for a microcontroller board to be used, I have decided that it would be best to go for an ESP32 type microcontroller, because of the following reasons: It has the fastest processor at 160 MHz, better for processing all the information that needs to collect from the sensors; more amount of memory, which will allow it to be faster with more intensive tasks; more GPIO pins available, which will be necessary for the number of sensors that we might need to connect to it and it also has the lowest recommended power supply, which will be beneficial as it will be connected at all times. Apart from all of the previous reasons, it is also the cheapest of all of the options.

Sensors: In this system, we will need to collect basic data for the management of the environment of a cleanroom. Those would be: Temperature, humidity, absolute pressure, barometric pressure. Most of the temperature sensors in this comparison are also able to measure other features such as humidity or even barometric pressure. We will take that into account when comparison. The different options considered for the temperature and humidity sensors are:

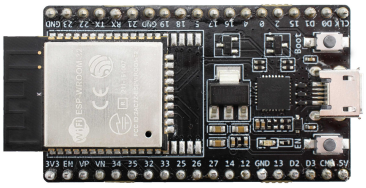

Figure 2: ESP32 microcontroller by Espressif

- The first option is the DHT22 temperature and humidity sensor. This was the sensor chosen in the previous version of this project. It measures temperature from -40C to 80C, with an accuracy of +-0.5C, and it covers from 0% to 100% of humidity with an accuracy of +-2%, with its recommended power supply from 3V to 6V. However it does not measure barometric pressure, which might be useful in terms of using as least hardware as possible.

- The next option is the LM35DZ temperature sensor. It measures temperature from -55C to 150C, with an estimated accuracy of +-0.5C at 25C, with its recommended power supply from 4V to 30V. However it does not measure humidity or barometric pressure.

- The last option is the BME280 temperature, humidity and barometric pressure sensor. It measures temperature from -40C to 85C, with an accuracy of +-0.5C, and it covers from 0% to 100% of humidity with an accuracy of +-3%, with its recommended power supply from 3.3V to 5V. It also measures barometric pressure sensor, from 330hPa to 1100hPa, with an accuracy of +-1hPa.

After looking at these three main possibilities, I decided to go for the BME280, because I found it easier to find, and cheaper. This was my line of thoughts: I ruled out the LM35DZ because it did not have a humidity sensor as well as the temperature sensor, so it meant to look for a different sensor for the humidity. Most of the humidity sensors came in conjunction with a temperature sensor so it was not a good choice. Also, it was analogue, so it would use more power, needing more power supply for it. I decided to go for the BME280 because it was easier to find and cheaper, apart from also being able to measure barometric pressure.

Figure 3: BME280 temperature, humidity and barometric pressure sensor

We will next look at the absolute pressure sensors, also called pressure transducers. At the beginning of the search I found some options that I thought could be valid for our project (such as the MPXH6400AC6U, MPXA4250AC6U and the MPX5100AP). However, these all could only measure from 0kPa to 400kPa approximately. After learning that we were looking for a sensor with an operating pressure range of 0bar to 10bar, I had to start searching

again. I could only find one model able to operate within that pressure range, the Honeywell HSCDLNN010BASA3 (010BASA3 for short). Due to various factors (including Covid-19 restrictions) the sensor did not arrive, and I was instead delivered a differential pressure sensor (010BGAA5). Due to the fact that we were not going to the CMN facilities at all (because of the pandemic) and both are analog sensors that connect to the microcontroller in an identical way, I decided to use that sensor (even if it was not what I originally wanted). This change does not affect the outcome of the project in any way, so there are no problems with this decision.

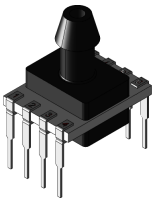

Figure 4: Regular analog pressure tranducer

3.2.3 Storing Time-Series Data: InfluxDB

We call Time Series Data to "the collection of data through repeated measurements over time". This means that we will get a sequence of data points that will be organised by the time they were measured at. The purpose of the collection of this type of data is mainly to monitor certain parameters closely and easily detect any change that may occur [12].

In this project we will be storing time series data in the form of temperature, humidity, barometric pressure and differential pressure readings. Because of this we have to choose the most appropriate database for this purpose. The database selected for this project was InfluxDB, a time series database designed for fast and high availability storage and retrieval of time series data. Time series data is stored by InfluxDB in a way that each field contains a series of blocks of time which are stored contiguously on disk, therefore being very optimised for fast operations on the data. Apart from that, InfluxDB uses a Line Protocol when sending time series data, which easily provides the query with the measurement, tags, fields and timestamp for very fast lookups [12]. It also offers clear advantages in speed when storing and indexing time-stamped data. This is because while other traditional databases would slow down when organising complex indices, InfluxDB is able to maintain high indexing speed because it uses a much more simple indexing system [13].

The structure of the data points stored in InfluxDB is very simple. At the highest level we have the name of the measurement. This will then be followed by two main key value pairs called the tags and the fields. These tags describe the metadata, while the fields will contain the actual values of the measurement that are to be analysed and stored. All of this will be accompanied by the time stamp of the data point.

One of the most important features of InfluxDB specific for this project is its schemaless design, as it allows for a better management of discontinuous data. This means that if some

of the data does not appear in a few readings, this does not result into a problem with parsing and transferring to InfluxDB.

InfluxDB is also very easy to work with and implement because it has no external dependencies, making it a lot easier to run, and also contains very high performing HTTP APIs, which will make the data transfer using these APIs a lot easier. It also has a sample client available, so it becomes very fast and easy to start for beginners or new developers interested in this database.

3.2.4 Data visualisation: Grafana

Grafana is one of the most important data visualisation platforms, as it provides an extensive range of functionalities and it is very easy to use and manage. We will explore its different advantages and features in order to gain a basic understanding of the purpose of this data visualisation platform.

Even if InfluxDB has its own dashboard in which you can keep track of the data, using Grafana also allowed us to introduce an extra layer of monitoring (this time from the server side), thanks to its Alerting feature. Grafana's alerting feature allows to send messages in many different ways (Discord, Teams or Slack notifications, e-mails, HTTP POST requests to specific URLs, etc.) according to the data collected. This can easily be done by everyone thanks to its graphical user interface, as you can define ranges of values allowed or conditions to be met interactively, without any need of programming knowldege. As this feature would be focused more on the backend/server side of this IoT system, it will be futher explored by my partner, in his counterpart of the project.

Grafana also has a wide variety of dashboards, such as heatmaps, histograms, geomaps, etc. This is also very useful, as it provides with different ways of displaying data, such that we can use different ones according to the data received, making it a lot more comfortable to use and easier to visualise.

Finally, Grafana is also able to provide support for many data sources, so we are not only able to import data from InfluxDB, but also from many others such as Graphite or Prometheus. This is a really interesting feature in terms of future scalability of the project, as the possibility of integrating many different data sources will be really useful for different types of data.

3.2.5 Security and Authentication: Tokens

This system is meant to be running in the University of Manchester Eduroam network, within the Manchester CMN server. Eduroam is a world-wide roaming service for the research and education community. It allows students and staff to connect to the Internet in an easy and secure way. This WPA2 Enterprise network uses the IEEE 802.1X standard for port-based Network Access Control, providing authentication of users in order for only authorised users to access the data shared over the network. Thanks to this, most of the security breaches are prevented, as only known users with access to the network would be able to access the

data.

However, this does not mean that all security threats are prevented, as users authorised by the network could still attack the system. In order to ensure the interaction between different users and the database in a secure way, InfluxDB uses a token authentiation system. A specific token belongs to a user/organization that interacts with the database. This token is also able to reflect their InfluxDB permissions.

There are three types of tokens: Admin tokens, All-Access Tokens and Read/Write Tokens. Admin tokens cannot be created manually, and have full access to all the different organisations in the database, with both read and right permissions to every single resource in each organisation. All-Access Tokens have full read and write access to every resource in a single specific organisation, while Read/Write Tokens have access to some specific resources in an organisation. In order to avoid possible mistakes when handling multiple organisations, it is more preferable to create an All-Access Token for every single organisation, rather than using the same Admin token for all of them [14].

Thanks to Grafana's Alerting feature, it is possible to remotely use the network to send HTTP requests that could remotely change things such as the sensor configuration parameters and more. In order to carry this new functionality out (which is being explored by my project partner in his counterpart of the project) the Arduino would need to host a HTTP server. However, this would mean that new possible security threats would appear, in which authorised Eduroam users could change the sensor configuration through an HTTP request. In order to prevent this, authentiation to the HTTP server should be necessary and would need to be implemented.

4 Implementation

Having chosen the appropriate hardware that will be used in order to carry out this project, I will now start talking about the different stages of the implementation, along with the specific areas of interest in which different tasks were performed in order to develop this IoT system.

4.1 Data collection

The first area of focus of the project is the collection of data. In this stage we will focus on how to collect the data from the sensors, in order to analyse it and pre-process it.

First, we will talk about the ESP32 microcontroller GPIO pin system. The ESP32 microcontroller has 48 pins, with multiple different functionalities such as Analog to Digital (ADC) channels, Digital to Analog (DAC) channels, UART, PWM, I2C and SPI communications, capacitive sensing pins, etc. Some pins have functions assigned by default, but most of them can be changed in the code. This is thanks to the ESP32 multiplexing feature (multiple peripherals connected to a single GPIO pin) [15]. In order to talk about how the different hardware components were integrated into the system it is important to understand the

types of communication protocols they use to interact with the microcontroller, the I2C and SPI protocols.

- I2C is a serial communication bus (sends data 1 bit at a time), used for peripherals without a focus on speed, but rather on low manufacturing costs and simplicity. With I2C you can have one or more master devices controlling one or multiple slaves, therefore it is very useful when performing multiple tasks with a single microcontroller, which is our specific case. It consists of two main lines which contain the data to be transmitted between the master and the slaves (SDA) and a clock signal (SCL), which is shared between the devices. I2C is synchronous, therefore the output is synchronised to the clock signal, which is controlled by the master. However, data is sent in packets, with a limited number of bits, therefore interrupting the transmission between each packet.

- SPI is a synchronous serial communication interface, which supports a single master controlling one or multiple slaves. SPI is also known as "the 4 wire bus" due to the fact that consists of 4 four wires: MOSI, MISO, SCLK and CS. MOSI stands for Master Output Slave Input, line used for the data sent from the master to the slave. MISO (Master Input Slave Output) is the line which does the opposite, sending data from the slave to the master. SCLK is the clock signal, used in the same way as I2C, as it is also a synchronous communication bus. Lastly, the CS (Chip Select) line will identify which slave the master is interacting with. Differently to the I2C bus, the stream of data does not need to be interrupted during transmission, as any number of bits can be sent continuously.

4.1.1 Integrating the BME 280 sensor

The specific version of the BME280 sensor is able to support both I2C and SPI communication. In this project it will be connected to the microcontroller using I2C communication. In the ESP32 microcontroller, the default pins for I2C (and the ones we are going to use for this project) are GPIO 21 (SDA) and 22 (SCL). We will also connect the Vin (Voltage input) line to the 3.3V pin and the Ground line (GND) to the Ground GPIO.

In order to easily integrate the sensor with the microcontroller, it is necessary to download and import two essential external libraries specific for the sensor: The Adafruit BME280 library and the Adafruit Unified Sensor library. These are made by the manufacturer in order to easily program the sensor. After this, we just need to include the appropriate code:

4.1.2 Integrating the Honeywell 010BDAA5 sensor

The Honeywell differential pressure sensor will be a lot simpler to integrate, as it is an analog sensor. This analog sensor only has 3 wires: Ground (GND), which is the line that will connect to the ESP32 GND pin, Voltage Input (3.3V), which is the higher voltage with respect to GND, and Voltage Output (Vout) which will contain the measurement and will connect the any Analog to Digital input GPIO. When reading an analog sensor, you are measuring voltage levels between 0V and 3.3V. The voltage measured will correspond to a

```
// Libraries to include
#include <Adafruit_Sensor.h>
#include <Adafruit_BME280.h>

// Declare and initialise the sensor
Adafruit_BME280 bme;

status = bme.begin(0x76);
if (!status) {
    Serial.println("Check wiring!");
    while (1);
}

// Read the values and put them into float variables
float temp = bme.readTemperature();
float hum = bme.readHumidity();
float bar_p = bme.readPressure() / 100.0F;
```

Figure 5: Sample code for integrating the BME280 sensor

value between 0 and 4095, as these analog pins have a 12-bit resolution. However, we must take into account the fact that ADC channels are not linear, so the ESP32 microcontroller will not be able to recognise properly the the highest or lowest voltages, giving the same result for voltages close to each other in these ranges.

For this project I decided to use GPIO 34 as the pin where the Voltage Output line will connect to. Now the next step is to read the value measured by the sensor and convert it according to the specifications of the piece of hardware. In this case, the range of the sensor is from 0 bar to 10 bar, so the value measured will be converted to a final value within the sensor range.

4.1.3 Integrating the Velleman VMA438 OLED display

The Velleman VMA438 OLED display will be connected in a similar way to the BME280 sensor, as it also uses I2C communication. As mentioned before, I2C provides support for many slaves to be controlled by a single master device, so we are able to use the same pins for SDA (21) and SCL (22).

In order to integrate this display, it is necessary to use its external library, U8g2lib. It provides specific methods that will set up the font, text position, etc. as well as very useful methods for drawing a wide variety of shapes.

```
// Libraries to include
#include <U8g2lib.h>

// Declare and initialise the display
u8g2.begin();

// Preparing the OLED display for writing
void u8g2_prepare(void)
{
  u8g2.setFont(u8g2_font_6x10_tf);
  u8g2.setFontRefHeightExtendedText();
  u8g2.setDrawColor(1);
  u8g2.setFontPosTop();
  u8g2.setFontDirection(0);
}

// Main code, draws something on the display
u8g2_prepare();
u8g2.clearBuffer();
u8g2.drawStr( 0, 32, "Hello World!");
u8g2.sendBuffer();
```

Figure 6: Sample code for integrating the VMA438 OLED display

4.1.4 Integrating the microSD card adapter

The HW-125 microSD card adapter uses SPI communication in order to connect to the ESP32 microcontroller, therefore this time we will use different GPIO pins to the ones we used for I2C communication. The MOSI line will connect to GPIO 23, the MISO lie will connect to GPIO 19, the SCLK (Serial Clock) line will connect to GPIO 18 and the CS (Chip Select) line will connect to GPIO 5.

4.2 Memory usage

One of the biggest challenges of working with one of these Arduino compatible microcontrollers is the appropriate use of its memory, as it is a limited resource that can be very useful, specially for our project. In order to try and store as much information as possible in the limited space that I have, I needed to reduce the size of the information as much as I could.

The raw data that is collected from the sensors is not immediately sent to InfluxDB. Instead I decided to collect various readings from the sensors over a defined period of time. Once I have collected enough readings, I obtain their statistics (I calculate the minimum value, mean value and maximum value) and then proceed to send those through the network to InfluxDB.

```
// Libraries to include
#include "FS.h"
#include "SD.h"

// Initialise the SD card
if(!SD.begin()){
    Serial.println("Card Mount Failed");
    return;
}

// Append to the file log.txt
File logFile = SD.open("/log.txt", FILE_APPEND);
logFile.print("Hello World!");
logFile.close();
```

Figure 7: Sample code for integrating the HW-125 microSD card adapter

In order to easily explain how the memory optimisation will be I will first have to explain how the data is constructed when sending it to InfluxDB: When we send data to InfluxDB, we effectively want to create a Datapoint, which is a data structure able to be representing by a string by being formatted in a specific way. This Datapoint will be formed by a series of data structures called KeyValue pairs. These key value pairs will be formatted as strings in the format [key + "=" value]. Examples of these data structures are "sensor_address=198.162.0.3" or "temperature=28.75". These key value pairs can be divided into 2 groups, tags and values. Tags are those key value pairs that hold the information about each specific microcontroller, such as the sensor address or the sensor group. The tags will help identify each microcontroller and monitor each one of them specifically. Values will hold information about the data that we want to monitor, such as the mean value, maximum value or even the different sensor configurations like the time of every push interval, etc. The format of the Datapoint will then be as follows: [measurement_name, tags values] where each key value pair in both tags and values will be separated to the next by a comma (,) (Example: "humidity,sensor_address=198.162.0.3,sensor_group=3 min_value=45.56,mean_value=47.84,max_value=48.54").

Once the data has been pre-processed and the Datapoints have been created, they must be stored in a buffer. Every certain amount of time, this buffer will be flushed and sent to InfluxDB. However, not all HTTP requests will succeed and go as planned, and therefore we must account for the times when the network will not be functioning properly. When this happens, we do not want to lose the information, as it can be very valuable. Because of this, we want to keep the information stored locally in the buffer, so when the HTTP requests all work again, all the stored metrics would be sent to InfluxDB (If this happens, all those metrics will appear to have been collected at the same time, as the timestamp is created once they reach InfluxDB. This will be futher explored in the later sections of the project, where we will discuss different data logging options).

In order not to lose the data, we need to keep as much information in the buffer as possible, so every small optimisation of that Datapoint would be a big advantage. As this project is about real time data collection, it consists of collecting the same metric periodically over a period of time. This means that over the course of many readings, the buffer that contains those Datapoints ends up storing the same information about the sensor and measurement type many times. That ends up being a lot of wasted storage that could be used in storing more metrics. In order to do that, I chose to implement bidirectional mapping:

- I initialise the map with some strings that I know will appear many times over the collection of readings, such as strings about the tags (sensor configuration) or about the statistics of the data (minimum/maximum value, etc.). I assign to these strings the first few numbers starting from 0. The measurement names (such as "temperature") will be added by the developer when uploading the new microcontroller code, and they will be assigned to the next numbers in order until 255.

- Instead of storing these strings (like "humidity", "max_value" or "sensor_address") every time a new Datapoint gets added, I will only store the associated number.

- By implementing bidirectional mapping, we can make sure that there are unique keys and unique values, and therefore we will never have problems associating a specific byte to a specific string.

- While there is an implementation of bidirectional maps in the Boost external library, it had many functionalities that we were never going to use. I found a lot easier and memory effective to implement two symmetric maps; while we had to store the mapping twice over the course of many readings, this method proved to be very memory efficient.

This implementation will not only be useful for buffer storage optimisation, but it will also provide with a way of identifying the measurements, as we will now have a measurement ID that we can use in situations like data logging (This will be explored in a later section).

4.3 Monitoring system

4.3.1 Monitoring status and Alert levels

In order to add a level of intelligence to the Arduino ESP32 microcontrollers, I chose to introduce a basic monitoring system, which would be in charge of making different decisions according to the data that it is being collected. With this system in place, the microcontroller would be able to change vital parameters of the system such as the time it takes to send a group of measurements to InfluxDB. As mentioned before, the intelligence system should be as close to where the data is collected as possible. This is because we want to be able to quickly make decisions, ideally without having to go through data transfers through the network, where many different problems can arise. Instead, if the microcontroller that collects the data from the sensors is able to take some of the load from the server and already make decisions that can prevent problems with the system,the overall flow of the system will be a lot smoother and less prone to errors.

The monitoring system consists of three main possible status, called "OK_", "WARNING" and "ALERT". These will represent what we will call the "health of the data". This basically means that they will reflect if the data that is being collected from the sensors is within the bounds that we should be expecting them to fall into or not. We will be able to define those upper and lower thresholds before compilation, so if some data is falling outside of the range within the thresholds, we will note it as a possible error or failure in the health of the data. Depending on which status the ESP32 microcontroller is in, the frequency of pushing the data to InfluxDB will vary. The monitoring status of the system will be updated every time data is sent to InfluxDB, as it is then when the data is pre-processed, grouped, and analysed.

In order to appropriately change between status, I had to come up with a design that would specify when exactly the best time is to change monitoring status, with everything that this change could entail. That is why I decided to use a variable called "warningToAlert". This variable will contain a number that will act as the measurement for how far up the alert scale we are right now. When the device is "OK_", this variable will be set to 0. This will be its minimum value.

This is how the data collection frequency will vary, according to the design of my monitoring system:

- OK_: This status will indicate that everything is going as expected, with all values falling within the range defined by the developer. At the moment, the InfluxDB push interval is 120 seconds. This was chosen to be the default value for data transfers, as it allows for multiple readings to be taken before the average, minimum and maximum values are sent through. I also chose 16 as the default number of readings per metric sent, giving a frequency for data collection of 7.5 seconds. I chose 16 because it was a power of 2, so it was easily doubled or halved according to the status of the device. It also means that a large number of values are being collected before sending the data, and hence following the guideline of "collecting as much data as possible and maintaining the system with the least amount of data transfers through the network as possible".

- WARNING: This status will indicate that some data might not be falling within the range of expected values that we defined, but this has happened recently and should not be taken as serious as the final status. When the device is in a WARNING status, the push interval to InfluxDB is lowered to 60 seconds, while the amount of readings per metric is lowered down to 8. By doing this, we keep the same frequency for the data collection from the sensors but we monitor the system more closely, as we send data to InfluxDB more often. If the device is "OK_" but a single measure does not fall within the thresholds, it immediately goes into the "WARNING" status, but the numeric level of alert (reflected by the variable warningToAlert) becomes 2, so the numeric level must go down to 0 by sending 2 sets of "healthy" measurements to InfluxDB.

- ALERT: This status reflects that there have multiple unhealthy measurements collected over the last priod of time, and they should be taken seriously and tried to be fixed.

This monitoring status is reached when the numeric level of alert reaches 5 (this would mean that 4 consecutive unhealthy sets of measurements have been sent to InfluxDB). The push interval to the database is further lowered to 30 seconds, and the readings per metric get lowered to just 4 (keeping the same data collection frequency of 7.5 seconds).

This system has proven to be an efficient way of monitoring the health of the data, with the objective of sending measurements more often to InfluxDB if the data that was being collected was not being expected. This point is the furthest the Arduino microcontroller will go to, as the server side (which was explored in more depth in the other counterpart of this project) consisting of InfluxDB and Grafana (or even a sensor control service application) will then have the possibility of create different types of alerts (such as email notifications) to quickly inform the appropriate members of staff.

4.3.2 Extra visual monitoring: Velleman VMA438

After adding this system, we still needed an easy way to check the health of the data collected by the device. Even though we could do that by going to either the InfluxDB or the Grafana dashboards, I wanted to make it easier to the members of staff already present by adding another piece of hardware to each microcontroller, a 128x64 OLED display. I decided to purchase the Velleman VMA438 OLED display, that connects to the Arduino via the I2C protocol. The display was easy enough to integrate, thanks to the external library U8g2lib that could be downloaded from the manufacturer's website. This library was really easy to use, and provided many different drawing functionalities that made it a great purchase, even if we did not need to use all of its capabilities. Any 128x64 OLED display with I2C protocol would have worked as well, as there are several libraries by Adafruit that easily integrate them with the microcontroller, but I found that the VMA438 would not work seamlessly with those libraries. If the VMA438 is not purchased, the code that integrates the display with the microcontroller would need to be changed. However, I am happy with my decision as it served its purpose, which is to display the monitoring status of the device at all times, letting the staff know if there is something that should be looked further or not.

4.4 Data Logging

4.4.1 Flash memory: Status bytes

In order to add an extra level of monitoring to the Arduino in unexpected conditions (such as loss of power) I needed to make sure that the latest information collected by the Arduino would be stored in flash memory (so when the power goes back on, the information should still be there). However, as mentioned before, memory is a limited resource so I wanted to store as much information as possible in the least amount of memory as possible. After some research and discussions, I came across the concept of "status bytes", and decided to further investigate on them. An example of where status bytes can be found are MIDI (Musical Instrument Digital Interface) messages, where the status byte (which is the first byte in a message) represents the type of action that should be performed [16] [17].

After discovering how much information a single byte could be carrying I decided to implement this system and store the last readings in flash memory, so if there was ever the need to check the microcontroller in order to see what had been happening with the readings until now, we just had to look at the status bytes last stored. These bytes can easily be checked by using the simple program I implemented called the "Maintenance Interrogation" program.

For every measurement analysed and sent to InfluxDB a status byte would be stored in memory, describing if the value had fallen within the range defined by the thresholds, if it was higher than the upper threshold or lower than the lower threshold. After every measurement was analysed, a final status byte with the new monitoring status would also be stored. However, 1 byte was not enough to describe all the information, so instead I decided to use 2 bytes for the status:

- The first byte would correspond to the binary value associated to each measurement name. This value is reused from the buffer optimisation section, making the measurements easy to store and more memory efficient.

- The second byte would contain either the information about the measurement health or the monitoring status. Bits 0, 1 and 2 represent the different monitoring status (OK_, WARNING and ALERT), while bits 3, 4 and 5 represent if the minimum value had been lower than the range, if the maximum value had been higher than the range or if the data had fallen within the range respectively.

This system provided a very efficient way of storing the latest data locally, in order to provide even more monitoring in case of unexpected circumstances like loss of power or failure in the transfers through the network. However, the very limited size of the flash memory in our ESP32 microcontroller proved to be a real challenge, as the amount of sets of data that could be stored could sometimes prove not to be enough. After considering the systems' really useful functionality and exploring its limitations, I decided to increase the amount of logging, even if it meant to purchase more hardware components that would help us with the process.

4.4.2 External storage: HW-125 microSD card adapter

In order to futher monitor the data that was being collected by the microcontroller I decided to add another functionality: microSD card logging. With this method, many more sets of measurements could be stored, and the data could be described in a lot more detail. I ended up purchasing the HW-125 microSD card adapter. This hardware component connects to the Arduino microcontroller using the SPI protocol, and can easily be integrated with the system using the SD external library. In order to store the data, I also bought a 16GB microSD card, which covered for more than enough data to be logged. The logging process consists on appending written text with the data to a text file in the microSD card. Each log would consist of:

- A sentence indicating the beginning of the logging process, along with its timestamp. This allows the members of staff to see if logging has stopped at any point (because

of unexpected circumstances), and they will also be able to see the amount of time it has not been functioning properly for.

- After that, every time the data is preprocessed and sent to InfluxDB the raw groups of values of every measurement will be appended to the text file, along with their timestamp and their status bytes. Finally the new monitoring status will also be appended to the text file.

With this new improved logging system, it is possible for the Arduino to hold even more information in other unexpected cases such as failures in the transfers through the network. Thanks to the timestamps we will be able to monitor and keep track of the graph over that period of time. We previously were not able to do that, because if the network was not working, when it would start working properly again, every set of measurements stored in the buffer up to that point would be sent at the same time, and therefore would appear in InfluxDB as collected as the same time as well, because the timestamp would be created once the Datapoints reach InfluxDB.

Overall, This new hardware component successfully increased the level of logging, and thanks to the 16GB of possible storage, there will not be as much need to worry about memory optimisation for logging. I also decided to display in the Velleman VMA438 OLED display the amount of memory used from the microSD memory card. This will easily tell the members of staff when to change SD cards if they ran out of storage.

5 Conclusions

References

[1] A. M. French and J. P. Shim, 'The digital revolution: Internet of things, 5g, and beyond,' *Communications of the Association for Information Systems*, vol. 38, no. 1, p. 40, 2016.

[2] A. India. (Apr. 2020). What is iot? defining the internet of things (iot), [Online]. Available: https://www.aeris.com/in/what-is-iot.

[3] K. Ashton, 'That 'internet of things' thing: In the real world, things matter more than ideas.,' *RFID Journal*, 2009.

[4] P. Suresh, J. V. Daniel, V. Parthasarathy and R. H. Aswathy, 'A state of the art review on the internet of things (iot) history, technology and fields of deployment,' in *2014 International Conference on Science Engineering and Management Research (ICSEMR)*, 2014, pp. 1–8. DOI: 10.1109/ICSEMR.2014.7043637.

[5] L. Goasduff, *Gartner Survey Reveals 47% of Organizations Will Increase Investments in IoT Despite the Impact of COVID-19*. Stamford, Connecticut, USA: Gartner Inc., 2020.

[6] M. Intelligence. (Dec. 2020). Iot analytics market - growth, trends, covid-19 impact, and forecasts (2021 - 2026), [Online]. Available: https://www.mordorintelligence.com/industry-reports/iot-analytics-market.

[7] A. Jahnke. (Jul. 2020). The 4 stages of iot architecture, [Online]. Available: https://www.digi.com/blog/post/the-4-stages-of-iot-architecture.

[8] M. Sharma. (Sep. 2019). Stages of "internet of things" architecture, [Online]. Available: https://www.marlabs.com/blog-stages-of-iot-architecture.

[9] B. Global. (Nov. 2018). How the internet of things is transforming supply chain management, [Online]. Available: https://www.blumeglobal.com/learning/internet-of-things/.

[10] C. M. Wijerathna Basnayaka, *Internet of things for smart cities*, Apr. 2020. DOI: 10.13140/RG.2.2.27861.78566.

[11] D. Team. (Nov. 2019). Comparing arduino vs. raspberry pi for iot projects, [Online]. Available: https://www.digiteum.com/comparing-arduino-raspberry-pi-iot/.

[12] P. Dix, 'Why time series matters, real-time analytics and sensor data,' *The best way to store, collect and analyze time series data*, Jun. 2020. [Online]. Available: https://www.influxdata.com/the-best-way-to-store-collect-analyze-time-series-data/.

[13] T. Matters. (Sep. 2020). Influxdb – explanation, advantages, and first steps, [Online]. Available: https://www.ionos.com/digitalguide/hosting/technical-matters/what-is-influxdb/.

[14] InfluxData. (2020). Manage authentication tokens, [Online]. Available: https://docs.influxdata.com/influxdb/v2.0/security/tokens.

[15] RandomNerdTutorials. (2015). Esp32 pinout reference: Which gpio pins should you use? [Online]. Available: https://randomnerdtutorials.com/esp32-pinout-reference-gpios.

[16] R. Blogs. (Aug. 2011). Status byte (of a midi message), [Online]. Available: https://www.recordingblogs.com/wiki/status-byte-of-a-midi-message.

[17] M. Association. (May 2018). Expanded midi 1.0 messages list (status bytes), [Online]. Available: https://www.midi.org/specifications-old/item/table-2-expanded-messages-list-status-bytes.