INSTITUTO POLITÉCNICO DE BRAGANÇA Escola Superior de Tecnologia e Gestão

UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

Development of a sensory data concentrator for network broadcasting

João Valter Girardi Neto

Dissertation presented to the School of Technology and Management of Polytechnic Institute of Bragança to the Fulfillment of the Requirements for the Master of Science Degree in Industrial Engineering (Electrical Engineering branch), in the scope of Double Degree with Federal University of Technology - Paraná.

Work oriented by:

Prof. Dr. Pedro João Rodrigues

Prof. Dr. Nelson Ricardo Rodrigues

Prof. Dr. Fábio Luiz Bertotti

Bragança

2020

Acknowledgement

First I want to thank my supervisor, professor Pedro João Rodrigues, my co-supervisors, Fábio Bertotti and Nelson Rodrigues, all the partners of the ON-SURF project and the IPB and UTFPR as well. I also want to thank my colleagues that worked in this project with me, Nicolle and Marcos.

Finally, I want to specially thank my family and friends who, even being far away, encouraged and supported me all this time.

This work has been developed under the project ON-SURF- Mobilizar Competências Tecnológicas em Engenharia de Superfícies Projeto n.º POCI-01-0247-FEDER-024521.

Abstract

This work is part of the ON-SURF project, which is involved in surface engineering in manufacturing processes such as metal stamping and plastic injection. On this matter, ON-SURF explores surface optimization procedures, aiming to improve the cost-benefit of the molds and the quality control of the produced parts. To this end, a data acquisition system was created to monitor the pressure and temperature of these surfaces, meeting the requirements of modularity and scalability, so that it can be expanded or contracted depending on the number of sensors required. This project consists of three different modules, in which one is responsible for the signal conditioning of the sensors, the other module, where the Olimex ESP32-PoE-ISO microcontroller was used, acquires, digitizes and sends this signal to a concentrator through Ethernet, while the last module, which was implemented in a Raspberry Pi and is the focus of this dissertation, is the data concentrator, responsible for gathering, processing and storing the data in a database. Some robustness tests were performed to verify the system. It was concluded that each conditioning module can support up to eight sensors simultaneously, with a sampling rate for 1kHz data acquisition, transferring to the concentrator using UDP protocol, with virtually zero packet losses, at a speed of 2000 packets per second.

Keywords: On-surf; concentrator; modular; scalable; data acquisition

Resumo

O presente trabalho faz parte do projeto ON-SURF, que está envolvido com engenharia de superficies nos processos de manufatura, tais como a estampagem metálica e a injeção plástica. Neste contexto, o ON-SURF explora procedimentos de otimização, visando melhorar o custo benefício dos moldes e o controle de qualidade das peças produzidas. Com esse objetivo, criou-se um sistema de aquisição de dados, visando monitorizar a pressão e a temperatura dessas superfícies, atendendo aos requisitos de modularidade e escalabilidade, para que possa ser expandido ou contraído dependendo do número de sensores necessários. Esse projeto é composto de três diferentes módulos, em que um é responsável pelo condicionamento do sinal dos sensores, o outro módulo, onde foi usado o microcontrolador Olimex ESP32-PoE-ISO, faz a aquisição, digitalização e envio desse sinal para um concentrador através de comunicação Ethernet, enquanto o último módulo, implementado em um Raspberry Pi e é o foco dessa tese, é o concentrador de dados, responsável por reunir, processar e guardar em banco de dados. Alguns testes de robustez foram realizados e se concluiu que cada módulo de condicionamento pode suportar até oito sensores simultaneamente, com uma taxa de amostragem para aquisição de dados de 1kHz, transferindo para o concentrador usando protocolo UDP, com perdas de pacotes virtualmente nulas, a uma velocidade de 2000 pacotes por segundo.

Palavras-chave: On-surf; concentrador; modular; escalável; aquisição de dados.

Contents

Acknowledgement **iii**

Abstract **v**

Resumo **vi**

1 Introduction **1**

1.1 Background and Motivation 1

1.2 Objectives 2

1.3 Document Structure 3

2 State of Art **5**

2.1 Injection Molding and Metal Stamping 5

2.2 Data Acquisition System 7

2.3 Embedded Systems 9

2.4 Data Concentrator 11

2.5 NoSQL Database 12

3 Materials and Methods **15**

3.1 Architecture 16

3.2 ESP32-POE 19

3.3 Raspberry Pi 20

3.4 Python 21
3.5 Ethernet communication and UDP protocol 22
3.6 MongoDB 23

4 Development **25**
4.1 Multicast and Ethernet Configuration 26
4.2 Clock synchronization 29
4.3 Receiving Data 30
4.4 Data Processing 32
4.5 Database Communication and Writing 33

5 Tests and Analysis **37**
5.1 Tests 37

6 Conclusions **39**

Bibliography **45**

A Code Developed **46**

List of Figures

2.1 DAQ System diagram 7

3.1 Project schematic 17
3.2 Data timestamping and sending 19
3.3 Raspberry PI 3B+ 21

4.1 Code flowchart 26
4.2 Visualization of data in MongoDB Compass. 36

5.1 Statistics from Wireshark test 38

Chapter 1

Introduction

1.1 Background and Motivation

This thesis is in the context of the ON-SURF project, a Portuguese program for research and development around surface engineering, where CEDRI-IPB (Research Centre in Digitalization and Intelligent Robotics) is involved in the sensory surface matter. This is a research unit aiming for scientific development and application in technological systems, located at IPB Bragança. CEDRI is the bound between ON-SURF and this master's degree.

The industry nowadays uses molds in the production of a lot of metal and plastic parts. Whether in cold stamping for metallic pieces or plastic injection molding, the mold is a very important part of the machine and possibly one of the most expensive parts. Because it's a component so expensive, it is important to use them as much time as possible, therefore raising its cost-benefit to the maximum. For that, it is necessary to know the efforts in the mold through time, monitoring changes in pressure and temperature in this mold surface. In this context, the ON-SURF project together with CEDRI-IPB, propose the development of a pressure and temperature data acquisition system to apply in the measurement of these

molds quantities.

The data acquisition system, in this case, is a modular and scalable system, divided into three modules, which are study cases for three thesis work, where one of them is approached in this dissertation. The first module is for conditioning the signals of temperature and pressure from the sensors. The second block, which is made with an ESP32 microcontroller, is responsible for configuring the conditioning module, acquire data and send this to a concentrator, through Ethernet communication. The focus of this thesis is the last module, which is the data concentrator, where a Raspberry PI is used. It is responsible to synchronize the clock in the ESP32, receive the data from this second block, organize it and then store this data, sending it after to a MongoDB database. This stored data will be available for future works to graphically interpret them so that the behavior of the measured quantities in the mold surface can be analyzed.

1.2 Objectives

The goal of this thesis is to create a data concentrator unit to receive temperature and pressure data from sensory modules, manage this data and send it forward to a MongoDB database. This module is responsible also to synchronize the clock of the sensory modules, so that data from the different sensing modules are aligned in time, thus enabling its analysis of the mold behavior. This module comprises a Raspberry Pi board that runs a program developed in Python language.

1.3 Document Structure

This thesis is divided into six chapters that describe the work developed during the research. The first chapter introduces the subject matter, the objectives, and the issue at hand. The second chapter brings research done on certain topics used in the manufacture of the final product. How the research was done, the methods and means used to conclude are presented in chapter 3. Chapter 4 presents the development of the thesis in detail. In chapter 5 the tests and analysis are shown and the conclusions can be found in the sixth and last chapter.

Chapter 2

State of Art

This chapter is dedicated to the contextualization of the project, showing some bibliographic references and concepts, which were necessary for the construction of this thesis. Such researches was properly referenced, starting with section 2.1, which talks about molds and the problem in the industry. Soon after, section 2.2 deals with data acquisition system. To talk about embedded systems and the data concentrator the sections 2.3 and 2.4 were reserved, respectively. The fifth and final section discusses about the database.

2.1 Injection Molding and Metal Stamping

There are some types of molding process used in production of metal and plastic parts, such as injection molding and metal stamping. Injection molding is a process which plastic is melted and injected with pressure into a mold. Shortly after, the material conforms to the mold, the pressure and temperature drop and the part is finished when it solidifies. For metal parts, metal stamping is a very common process used in the industry, which can be hot or cold pressed. The process of stamping consists of a metal plate placed over the mold, then a press

hits that plate against the mold, conforming the metal to the mold. The difference between both process is that the metal is heated before going to the press in hot stamping. Hot stamping produces a strong and lightweight piece, but it needs ultra high strength steel boron alloys [1], commonly the Usibor 1500 boron steel [2] due to its very high level of strength after hot stamping. The heating process with the need for a specific metal tends to make this process more expensive than cold stamping. In cold stamping a greater variety of metals can be used, but the pieces are not as strong as in hot stamping [3].

The problem for industries in general is that these molds have high purchase value and cannot have any imperfections as the parts must come out of them without any flaw. Therefore, the molds, which have a certain estimated lifespan, end up being suspended ahead of time, to prevent that any imperfections can occur because they don't have precise monitoring through their lifetime. This way they end up not being taken full advantage of. Thus, it is necessary to know more precisely the behaviour of these molds during their production time and detect variations in pressure and temperature on their surface by sensoring the mold surface. Another point of this sensing is quality control for the fabricated parts [4], [5]. If variations on the pressure and temperature pattern occur, by measuring these values through time it is possible to detect calibration problems in the machine or some imperfection in the manufacturing.

Using an embedded system of data acquisition, installed together with the sensored mold is the way to gather this kind of data so one could analyze the mold stress behaviour and predict more precisely the lifetime of it.

2.2 Data Acquisition System

By measuring temperature and pressure variations over time, the mold surface monitoring can be performed. The process of measuring a physical quantity from the environment, like temperature, and translating it to digital data is the process named data acquisition or DAQ. Data acquisition systems are divided into three categories: computer-based, embedded micro-controller based and Field Programmable Gate Array (FPGA) based. [6]. The flow of data in a basic structure of a DAQ system starts with a sensor, that usually has a sensitive part and a transducer, going to a data acquisition hardware and finally to a computer to process and store the data. The sensor is the part that reacts with the energy of the physical quantity being measured, for example temperature, and the transducer will convert this energy to another type of energy that can be used, like electricity. This electrical signal coming from the sensor must be conditioned, because it usually is very low energy, so it must be amplified to workable levels. This is done in the signal conditioning hardware, that also will digitize it, converting this analog signal to digital signal. Thus having this digital data representing the physical quantity, that the computer can interpret, it is possible to process, store in a database and graphically analyze the physical quantity with specific software for the application. A general DAQ system is represented in Figure 2.1.

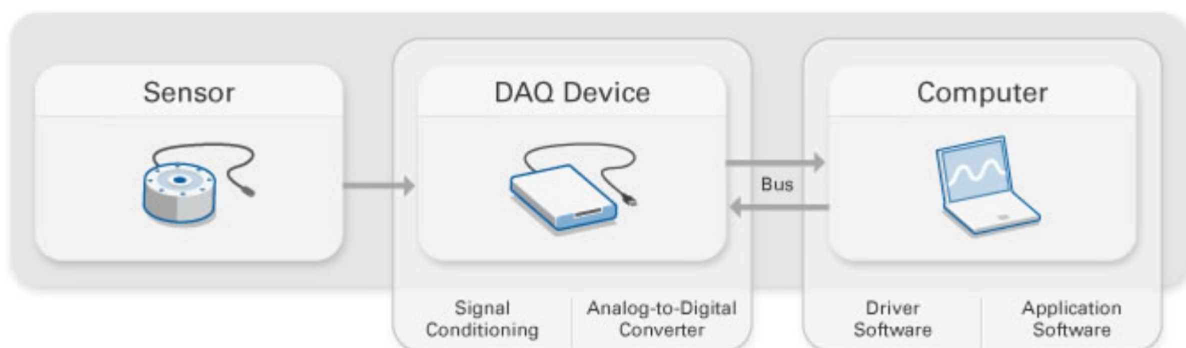

Figure 2.1: DAQ System diagram

For pressure to be detected, there are different types of sensors for this purpose: piezoelectric, capacitive and resistive. Piezoelectric sensors are ceramic materials with electrical voltage generation properties when subjected to mechanical loading. Capacitive sensors, on the other hand, have their capacities varying according to the force applied to it. Finally, resistive sensors vary their electrical resistances as the strain deform them [7]. In the context of the resistive sensors there is a good example of a sensor for measuring strain on surfaces of molds that is the extensometer or strain gauge. It is a transducer used to measure body deformations crafted as a thin wired film that deforms together with the base where it is fixed. It is based on the fact that the elastic deformation of a material changes its electrical resistance. The extensometer is fixed on the surface that is intended to measure the deformations. When the sensor conductive wires vary in length due to strength in surface, its electrical resistance also varies. This electrical resistance is measured and its variation can be related to the deformation of the piece, thus calculating the strain [7].

To measure temperature there is a set of sensors to use, like the thermocouple, thermistors and Resistance Temperature Detectors (RTD). This can be used in similar cases and the choice depends on the temperature range and the precision needed. The thermocouple is a sensor with two different metals coupled, that will produce a voltage dependent of the temperature as a result of the thermoelectric effect. It can be made from various metal combinations, depending on the temperature range of the measured environment, the sensitivity wanted and the chemical inertness. The RTD vary their electrical resistance with temperature and can be made with pure metal or a semiconductor. The metallic RTD are produced with a very pure metal, usually platinum, copper or nickel, with platinum the most common, due to its inertness and high fusing temperature. The thermistors are semiconductors that also vary their electric resistance with temperature but have

a higher temperature coefficient than the RTD [8].

Sensors usually need a signal conditioning stage so that their data is acquired correctly. This step is placed as close as possible the sensors to minimize external interference and acts mainly to amplify the measured signal for the range of the micro-controller's ADC (analog to digital converter) and filter this signal from possible external noise. The conditioning step enables the data acquisition module to collect data more accurately [9].

The data acquisition commonly is associated to the conditioning circuit, acting as an analog to digital converter. A micro-controller with ADC is the usual device chosen for this application and there are several things to be considered before picking the right micro-controller, since the devices differ in terms of power supply and ADC characteristics. Characteristics of the input signal, sampling rate and resolution needed for the ADC are technical features that, along with the price limit impact in the product choice [10].

DAQ systems are used in many different applications, from very simple applications to very complex measurement systems. For example, the pressure measurement of inert gases in industrial processes, for the implementation of a control system for the pressure and flow of these gases, require a system of data acquisition for these gas-related physical quantities [11]. Or as an example of a pretty complex measuring system, this neutrino observatory in the South Pole, called the IceCube, that measure data with sensors in 2400 meters deep holes, digitizing it at and transmitting to the data center [12].

2.3 Embedded Systems

With the evolution of information technology, some technologies come to aid the achieving of the so-called ubiquitous information or pervasive computing [13],

[14], aimed by the modern computing in applications like the IoT (Internet of Things) and embedded systems are a major part of this necessary technologies. Embedded systems are information processing systems that are enclosed into a larger product and that are normally not directly visible to the user. Frequently they are connected to sensors and actuators to collect information and interact with the environment [15]. An example of an embedded system is the electronics of a car, like the fuel and oil level measurement or the engine injection control. A system can be considered embedded when it is dedicated to a task. In this way, the specific task for the system is optimized.

It is desirable that the system deal with a possible variation in its usage need, like when a user of a data acquisition system may want to increase the number of sensors scattered around its measurement plant, depending on the size of the plant or just the desire for more sensors. That's why the system must be scalable, enabling this expansion of its functionalities as needed. The concept of scalability refers to the capability of a system to be expanded as needed, without the performance being compromised. It's about a system of easy maintenance that can have its functionality expanded without much impact on the work [16]. In Bondi words, scalability "connotes the ability of a system to accommodate an increasing number of elements or objects, to process growing volumes of work gracefully, and/or to be susceptible to enlargement" [17]. A system with no scalability may need to be re-engineered to follow the growth of its use. Another definition of a scalable system is "if it can handle the addition of users and resources without suffering a noticeable loss of performance or increase in administrative complexity" [18]. Another concept related to scalability is horizontal and vertical growth [19]. When the replacement of a system's component can be done, enabling an upgrade in the processing and overall capacity of the system, this refers to vertical scalability. The concept related to horizontal scalability refers to when components or nodes

can be added and thereby increase the system capacity without the need to replace any existing component of the already existing system.

A system generally is desirable to be scalable, but scalability brings a problem in systems that has several sensing devices acquiring data for a common purpose. Applications that will use gathered data from different sensors assume that all information marked with the same timestamp was taken at the exact same instant, but when a system has several measuring values, a problem with the clock synchronization may occur. Imprecise timestamps cause wrong results in the post-analysis, such wrong correlations, missed event detection and false predictions [20].

This thesis data acquisition system is meant to scale horizontally because it is able to withstand an increase in the number of sensing modules as more sensors are needed in the measuring plant. As this DAQ system is designed to be horizontally scalable, the measuring system needs something to gather and manage all the data measured. To do this concentration of information from the several sensors, a data concentrator is be implemented in the topology.

2.4 Data Concentrator

Internet of Things is said to have two key aspects, which are the devices and the server that supports them, but often a third aspect is added to the definition, that is a gateway to perform aggregation, processing and bridging between the device and the internet [21].

The data concentrator is the part where the measured data from all the sensory modules are gathered and is needed to acquire data from the various measuring sources. The data concentrator collects realtime data and sends these data to realtime applications or to data storage (eg, database) at a regular time [22]. In telecommunications, a concentrator is a functional unit that permits a common

path to handle more data sources than there are channels currently available within the path [23]. It's similar to a data hub, but with a little more "intelligence", and capability of change.

There are works that show the use of data concentrators for monitoring power lines, as a part of the topology of the system [22] [24] and a facilitator in the gathering of the information from smart energy meters [25].

2.5 NoSQL Database

NoSQL (Not Only SQL) is a term to represent a non-relational database. To put it in context, SQL (Structured Querry Language) is the standard language for managing data in the relational model, that is based on the concept of entity and relation. Each relation is a table, where the data is stored and separated into columns. Each row represents a collection of values, which represent data related to real world entity. The name of the columns and tables is used to interpret the values of each row. Formally a row is a tuple, a column is called an attribute, and a table is a relation [26]. This predetermined structure requires the database user to plan the data structure, because each attribute of each table and their relations must be defined beforehand so that information can be stored in it. If any new information needs to be added, it is necessary to create another table and determine the relations between the tables by means of identification keys [27]. This way the traditional relational database management system (RDBMS) is static, meaning that its data structure is fixed and the rows must fit in the planned model. This fixed structure allows for fast dynamic queries by making early assumptions and optimization [28].

So NoSQL databases have a different proposal to deal with data, based on models for accessing and managing data such as key-value, graphical and document-oriented databases. There are some features that help define NoSQL, such as data flexibility, allowing the use of semi-structured or unstructured databases; scalability. These models all introduce a different way of dealing with problems of high data load, scalability and fast change in data structure, which are characteristics of the project being described in this dissertation [29].

The main reasons for using NoSQL database are performance and flexibility. According to Parker et al., when performing inserts, updates, and selections, MongoDB (a NoSQL database) is faster, but MS SQL Server (SQL database) outperforms MongoDB by executing complex queries rather than simpler access to key-values [30]. Given that the data to be stored by the system developed in this thesis is quite simple, this fact described by [30] will optimize the process of using the databas, [31] describes the use of NoSQL databases for managing the large volumes of data in IoT environments, since IoT databases must be flexible to follow IoT applications and are this category of database easily accommodate different data types and structures without the need for predefined.

NoSQL premises such as the schema-free property fits in the scope of the concentrator system being described in this dissertation since its easier to modify data structure with the likely change that will happen over time and from system to system. It also makes possible to extend its use to other applications outside the molds context.

Chapter 3

Materials and Methods

The context of this thesis is inside ON-Surf project, that aims to develop and apply surface modification processes that promote innovative solutions in different activity sectors, such as Automotive, Aeronautics, Molds & Tools, Health and Electronics. Being a National Program, ON-SURF involves different Portuguese companies and non-business entities around Surface Engineering, aiming to bring together scientific and technological knowledge in Surface Engineering (SE), with the recognized needs of the business fabric to modify the surface of products and components. Within this objectives is inserted the project of this thesis, which target to develop a sensored surface, by building a data acquisition system to acquire sensor measurements from thin layer sensors installed in molds.

Chapter 3 will approach the general modeling of the DAQ system project, focusing in explaining the structure and the components of the concentrator module, as well as some notion of the whole pressure and temperature measuring system. Some subjects studied throughout the project will be presented, to introduce some basic knowledge about the context.

3.1 Architecture

The project that this thesis is contained need to develop a data acquisition system to acquire sensor measurements from sensors installed in molds. There are signal conditioning modules, signal acquisition modules and a concentrator module in this system. The system is designed to be scalable in a way that if the user needs, it is possible to add more sensory modules if more sensors are required. This way, the measuring system needs something to gather the data measured and that is the role of the data concentrator, the focus of this thesis.

The concentrator scalability of this thesis is likely to be limited to the point that the concentrator cannot process the data that reaches it quickly enough when many data acquisition modules are added. It is also probably limited by Raspberry Pi's Ethernet bus bandwidth, which will determine the amount of information passing through the Ethernet socket.

This thesis plan is to communicate a set of ESP32 with a Raspberry Pi, via Ethernet communication through UDP protocol. When data is sent by the ESP32, Raspberry takes the information given in a String vector and organize in a correct format to the database. Figure 3.1 shows the schematic of the complete data acquisition system, divided into the modules that will be described below.

The focus of this dissertation is to describe the development of the data concentrator module, which includes Raspberry Pi, the Ethernet switch for networking between ESP32 and Raspberry Pi and the database. The switch chosen has POE (Power Over Ethernet) support. POE is a method to power devices through the Ethernet cable, without the need to add another power supply to it. Having this feature makes possible to provide power to the Esp32 microcontroller through the existing Ethernet cable that establishes the connection to the Raspberry. However the Esp32 has to be POE compatible because there must be an electrical circuit to convert the voltage and current values in its input to the acceptable levels of the

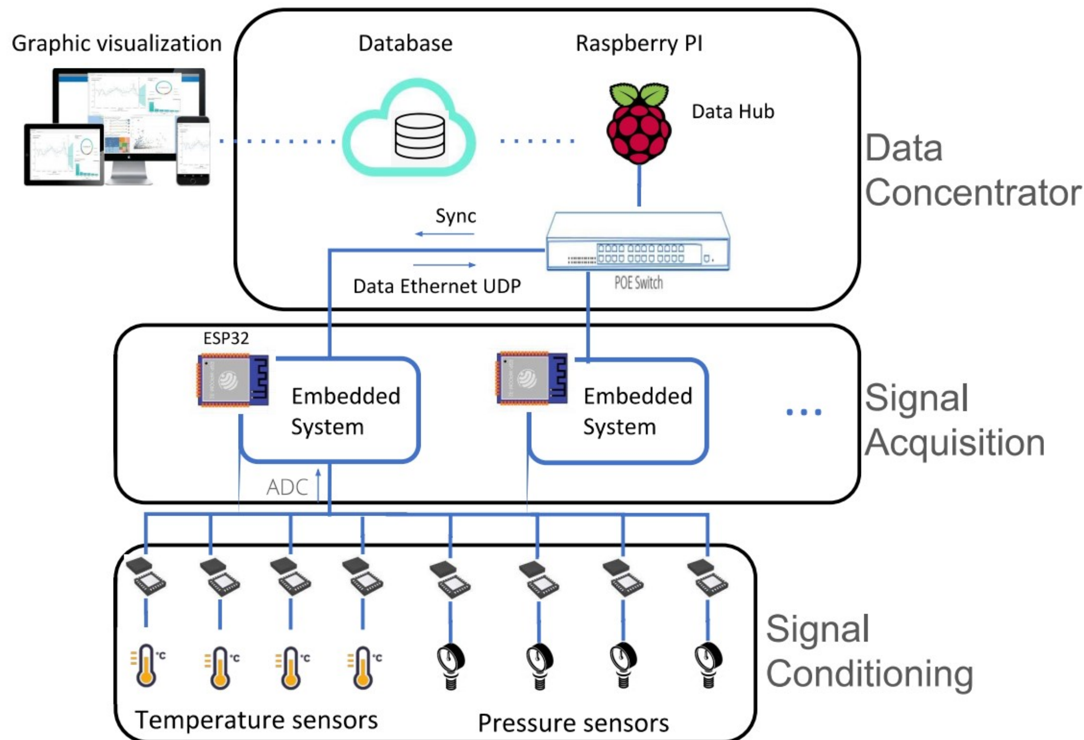

Figure 3.1: Project schematic

microcontroller. This feature is for ease of installation and improves the system scalability because it's not necessary to install power supplies for these modules or use more cables than the Ethernet itself. Another part of the concentrator is the MongoDB database. This database is hosted in the cloud so that future parts of the work will be able to access the data for viewing and analysis of the data collected in the measurements. Atlas is the MongoDB's cloud-hosted database service being used. The main part of this module is Raspberry Pi, which gathers the data sent by the Esp modules and controls the flow to the database, processes this data and is also responsible for a significant part that is the ESP32 clock synchronization for timestamp administration. The programming is all done in Python language to execute these tasks. Another important detail is that the clock synchronization signal from the Raspberry Pi is transmitted to all Esp32 at

the same time, through multicast addressing. This method of addressing packets allows the communication to all the modules without the need of addressing each one individually.

The measurement starts with the sensors, which will measure pressure and temperature values analogously. This sensors are being developed by another institution, member of the ON-Surf program, but they were not finished on time for the tests. That is why similar sensors were used instead, like the extensometer for pressure measurement and the PT100 for temperature. These physical values measured are collected by a signal conditioning circuit, which will amplify the magnitude of this signal and also give it the correct offset, putting it at acceptable levels for the analog inputs of Esp32 (limited by 3.3V). The Esp32 will support eight simultaneous sensors, which are four pairs of temperature and pressure sensors and these elements together make up the sensory module. In the Esp32, the conditioned data is acquired by the ADC, which converts the analog signal to digital. At this stage, the measures from the eight signal conditioners are merged into a single data vector and this vector is timestamped, so that it can be sent forward in the system to the concentrator.

The timestamp is associated to the eight simultaneous measurements of the different sensors connected to Esp32 and this is the time instant considered for their measures. The information of each sensor is identified inside the data vector, with the measured voltage or pressure value and the respective identification Id of the ESP and the sensor that measured it.

After this data concatenation process and timestamping, the vector with the measurement information is sent to the data concentrator through the Ethernet network, using UDP protocol. The information packet is addressed with Unicast method, therefore it will be addressed only to Raspberry Pi.

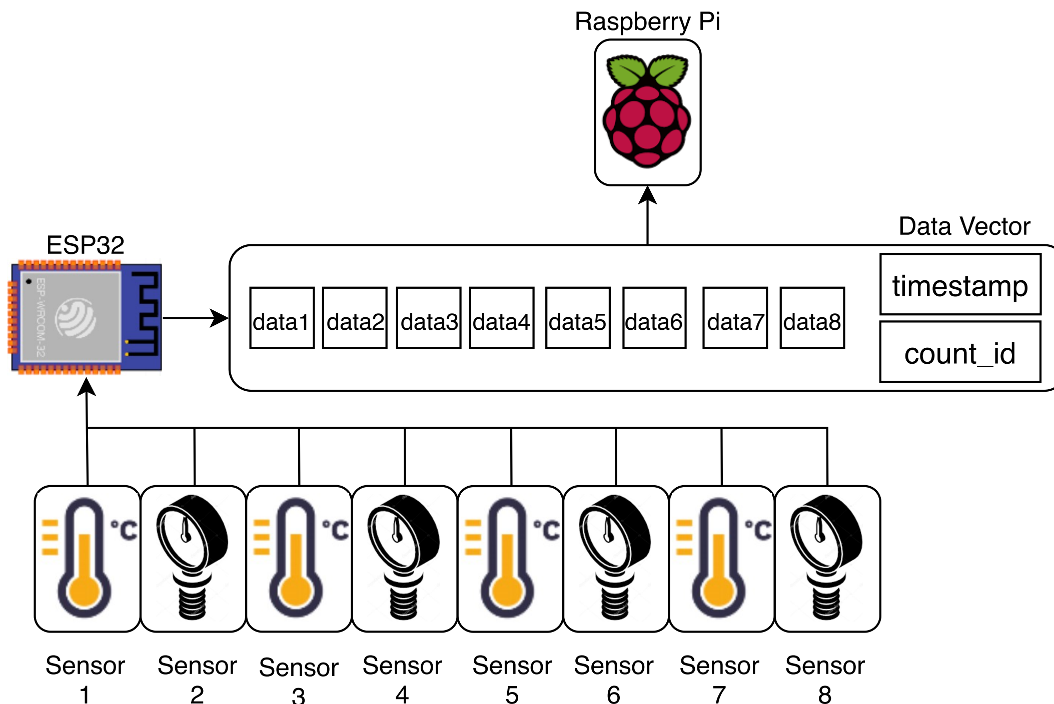

Figure 3.2: Data timestamping and sending

3.2 ESP32-POE

This project used Olimex ESP32-PoE-ISO boards for the acquisition module. Although the focus of development was not this part of the project, it depend on these devices. The board is open-source hardware and software, manufactured by Olimex. This microcontroller model has Power Over Ethernet support and a 3000VDC galvanic insulation from Ethernet power. It has only 12 ADCs channels with 12 bit resolution that is enough for the project.

3.3 Raspberry Pi

Raspberry Pi is the computer used for this project. It is a compact, high-performance and low-cost single board computer, designed by Raspberry PI Foundation, that runs on an open-source operating system based on Linux. It has the peripherals to use as a normal computer, like USB ports to plug mouse and keyboard, HDMI for a monitor or can be accessed remotely with another computer. Also has GPIO used to connect electronic components and explore the Internet of Things (IoT). At the moment this project started, the latest Raspberry PI model was Raspberry PI 3B+, which is the one used. It has better functionality over the previous versions, like faster Ethernet and Power-over-Ethernet support. It also has a 1.4GHz 64-bit quad-core processor, dual-band wireless LAN for using a WiFi network to connect with the database [32]. The memory is an external SD card of 16GB.

The officially supported operating system for PI is Raspbian, an operating system from Debian optimized for PI [33]. In a computer, the operating system is a program managing system resources, supporting computer's basic functions and applications running in the machine and has an interface between computer and user. The programs will run in an environment provided by the OS, that also will provide all services for the program, like file-system manipulation, I/O operations and communication between systems [34]. Debian is a free operating system, based on Linux that is stable and scalable. Also this operating system is not real-time, i.e. it is not possible to guarantee response within a strict timing constraints.

A real time operating system is a system that has a strictly defined time to respond to an event. If the system's scheduler actions can be predicted and are executed within a time, the user has guaranteed to meet real-time requirements. Since the Raspbian doesn't meet this requirement, it is not possible to predict

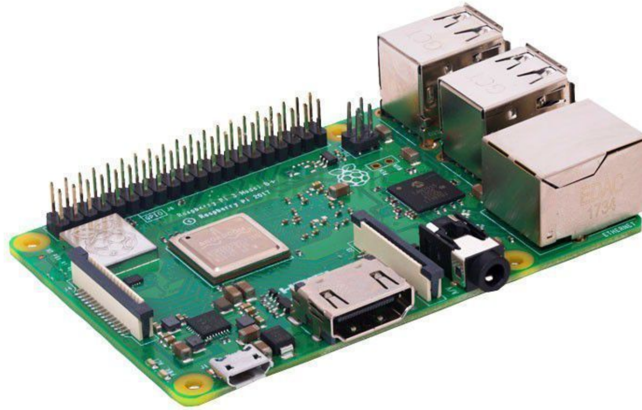

Figure 3.3: Raspberry PI 3B+

if the data arriving at the Raspberry PI Ethernet interface will be processed in time. For this reason, it is the ESP32 module needs to set the timestamp of the measurements, before sending data.

3.4 Python

Raspberry PI has support for several programming languages such as Python, Java and C. This gives broad access to libraries of the chosen language. In this thesis the language of choice was Python. Python is known for its writability, error reduction, and readability. It is open-source, making it freely usable and distributable [35].

Python has easy ways to work with tables and lists, very useful for data managing and is not so challenging to work with. It becomes easier for people not so used with programming because of its expressive language, more understandable and readable It is free and Open Source with plenty of libraries and great support by the users community on the internet, allowing for faster implementation of the functionalities and debugging.

3.5 Ethernet communication and UDP protocol

To communicate through the Ethernet network, the messages will be sent using the User Datagram Protocol (UDP). The internet has two communication protocols: UDP and TCP. TCP is more reliable, because it has a packet loss checking mechanism, that the receiver confirms if the packet arrived, providing acknowledgment to the sender device and if not, it resends the lost packet. UDP, on the other hand, does not have this data arrival acknowledgment, having a speed advantage over TCP, since the sender device does not need to wait for confirmation [36]. However, if a packet is lost eventually, there is no way to recover it. But this loss rarely occurs in a local network, like in this thesis project. This problem of losses can be surpassed by sending the data with redundancy. Receiving more than once the data makes possible to recover an eventual packet loss.

The choice of UDP protocol is justifiable in this project, where the data transfer speed needs to be high due to the sampling frequency set to the ESP (1kHz) and it is not desirable for the processor to be busy waiting for a packet receipt acknowledgment and then resending a packet if an eventual loss occurs. Besides, it is not too relevant a packet eventually being lost, because of the high amount of data measures and the repeatability of data in a process like stamping, that is identical every cycle.

Another point is that UDP will allow to send data in multicast mode. Multicast is a mode of sending data where the sender addresses the data packet to all the devices that are within a group, called a multicast group. In the Raspberry case, it is an essential point to enable simultaneous clock synchronization on all ESPs, since the synchronization signal sent by multicast will be addressed to all devices at the same time. This is way, all ESPs will be connected to the multicast group and will absorb the Raspberry Pi synchronization.

3.6 MongoDB

MongoDB is a free NoSQL database that is easy to learn due to its data flexibility. It is supported for several operating systems such as Linux and Windows, making it the chosen database for this thesis. This ease to run in multiple platforms also helps with the system scalability.

The decision to choose how to represent the information exchanged, and how to structure it (e.g., ontology), raises an analysis of the various alternatives [37], while in a relational database there is a typical schema design with tables and the relationship between them, MongoDB has no concept of relationship. Mongo is a document-oriented database, stores data in a document, rather than structured tables. It uses a JSON like model, called BSON (Binary-JSON), making it schemaless, enabling to change a document individually, independent of other documents, meaning that each document can differ from one another in number of fields, content and size. The data structure can change through time [38].

MongoDB's support for dynamic queries means that you can run a query without planning for it in advance [28] Its non-structured architecture make possible the easy implementation of a database for the data acquisition system that can have its data structure changing throughout the project development.

It has native sharding support, making easier to scale horizontally when the need comes with large data sets. Sharding means the database partition data for distributing them to multiple machines. A cluster can still perform operations if sharded and one or more machines are unavailable [39].

Chapter 4

Development

Chapter 4 describes the steps to implement the data concentrator module of the data acquisition system, highlighting the most relevant points. Here the evolution of the work is described, which are divided into sections, showing the basic configurations to the data exchange between the sensory module and the database, as well as some problems. Section 4.1 indicates the configuration of Raspberry‘s Ethernet interface. The next session describes the clock synchronization. Sections 4.3 and 4.4 explains the data receiving and the processing of data to prepare for storage. Finally, section 4.5 shows the configuration to connect with the database and the storing of data in it.

The sections follow the flowchart sequence from figure 4.1 that resumes the features to occur inside Raspberry Pi.

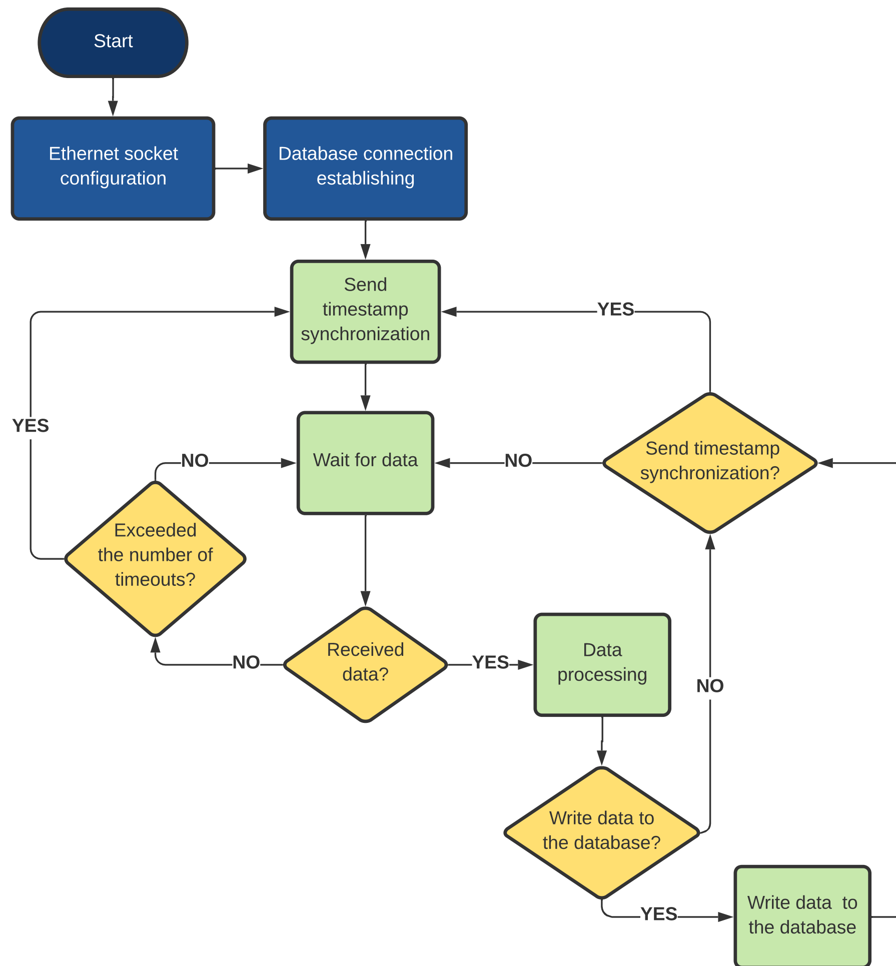

Figure 4.1: Code flowchart

4.1 Multicast and Ethernet Configuration

The first action is the configuration of Raspberry's Ethernet socket. In the multicast configuration function, the Ethernet interface and the multicast group configurations are set. The IP address for the interface will be provided by the network server that provides the connection to the database. The program automatically detect the Raspberry‘s IP address and attribute to the socket in the code. The multicast group address is also defined in this step and should be the

same on all devices in the multicast network (Raspberry PI and ESPs).

```
ifname = 'eth0'
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM,
   socket.IPPROTO_UDP)
INTERFACE = socket.inet_ntoa(fcntl.ioctl(sock.fileno(),
   0x8915, struct.pack('256s', bytes(ifname[:15],
   'utf-8')))[20:24])

GROUP_IP = "224.3.29.71"
GROUP_PORT = 10000
multicast_group = (GROUP_IP, GROUP_PORT)
ip_pack = struct.pack("4s4s",
   socket.inet_aton(GROUP_IP),
   socket.inet_aton(INTERFACE))
sock.setsockopt(socket.IPPROTO_IP,
   socket.IP_ADD_MEMBERSHIP, ip_pack)
sock.settimeout(1)
```

Listing 4.1: Multicast and Ethernet configuration

The `socket()` function creates a socket object, that is an internal point for data transferring on a network and it‘s attributed to the `sock` variable. It requires three arguments that define the address family, socket type and protocol number for the socket. `AF_INET` is for the IPv4 address family. `SOCK_DGRAM` is the socket type for UDP protocol. `IPPROTO_UDP` is the protocol number for UDP. At this point the socket is created for data transfer on the network, which the IP address

of the Ethernet interface will be attributed.

`ifname` tells the name of the interface, in this case 'eth0' for the Ethernet interface. `inet_ntoa(fd, request, arg=0)` function takes an IP address in 32-bit packed binary format and converts to IPv4 string format. `ioctl` is for low-level access to Linux network devices. `fileno()` returns the underlying file description, that is an abstract indicator used to access a file or other input/output resource, such as a network socket. The constant 0x8915 in the second argument indicates the socket ioctl to get the interface address. When the third argument, in this case given by `pack()` function, is a bytes object argument a buffer will be created whose address is passed to the `ioctl`. The content of this buffer is the return value and is the information that the constant 0x8915 indicates `ioctl`, which is the interface address. `INTERFACE` variable is where the internal IP of the Raspberry PI Ethernet interface is inserted.

`GROUP_IP` and `GROUP_PORT` variables are the IP address and port of the multicast group, which should be the same on Raspberry and all the ESP32. Then `multicast_group` variable receives the multicast group address, grouping its IP and port.

`struct.pack(format, v1, v2, ...)` function converts Python values to a C struct, returning a bytes like object, packed in the format given in the first argument. This conversion is needed, because the `setsockopt()` function need a value in network byte order format to the buffer argument. The "s" is for string type and the "4s" means a 4-byte string. So in this case `"4s4s"`, it means a 4-bytes string followed by another 4-bytes string. The second and third arguments are the values being converted. The `socket.inet_aton()` function converts an IPv4 address from dotted-quad string format (like `'192.168.137.1'`) to 32-bit packed binary format. So `socket.inet_aton('INTERFACE')` returns a 4-bytes object converted from '192.168.137.1'.

`setsockopt(level, optname, value: buffer)` sets a socket option. The constant IPPROTO_IP in the first argument, tells that the option is set in the IPv4 level. In the second argument, IP_ADD_MEMBERSHIP is selecting the option to join the socket to the multicast group specified. The last argument is a value representing a buffer for the selected option.

At last a timeout counting is configured for the socket, to be used as a countdown flag to resend the timestamp synchronization.

4.2 Clock synchronization

It was established at this moment how the ESP32 clock would be synchronized by Raspberry Pi, to keep the measurement timestamp aligned between the various modules. The clock synchronization signal also works as a starter for the ESP32, to begin sending the data from the sensors.

ESP32 is in charge of marking with a timestamp the data, so that the sending delay plus the fact that the Raspberry Pi operational system is not real time does not hinder the further analysis of the data. It is needed to ensure that the correct timestamp is known so the timestamping is made in the data source, the ESP32.

The current time of Raspberry Pi is obtained with the function `time()` from the Python's `time` library and is synchronized with a NTP service. This function returns the number of seconds since the Epoch, which is when time begins, that in Unix systems for example is `January 1, 1970, 00:00:00`. This number is then sent multicasting to all the connected ESP32 modules, with `sendto()` function. The time integer also needs to be converted encoded in the UTF-8 format before sending, shown in code 4.2

```
message = str(time.time())
sent = sock.sendto(message.encode(), multicast_group)
```

Listing 4.2: Timestamp sending

The ESP32 timestamp is calculated according to the equation 4.3, where `t_pi` is the clock synchronization number sent by the Raspberry Pi, `timer_esp` is the signal measurement instant referred to internal clock of the ESP32 that counts from the boot. The instant of the last timestamp sent by the Raspberry, referred to the ESP32 clock is attributed to `t0`. The timestamp is the real instant that the current measure is made and it is given by the period between the measurement instant `timer_esp` and the timestamp arrival time referred to the ESP's clock `t0`), plus the clock synchronization `t_pi`.

```
timestamp = t_pi + timer_esp - t0
```

Listing 4.3: Timestamp equation

This Raspberry Pi time is sent again every 10000 cycles to maintain the synchronization in all the sensors, neutralizing the clock drift that occurs over time.

4.3 Receiving Data

After the communication being set and the clock synchronization is sent, the program will start receiving data from the sensors. For this part it was needed to define the format of the data sent by the acquisition modules, to be organized by the concentrator and recorded in the database.

```
while True:
    try:
        c = c + 1
```

```
        data, address = sock.recvfrom(1024)
    except socket.timeout:
        print('timed out, no more responses...')
        if c % 5 == 0:
            break
    else:
        data_processing()
        if c % 10000 == 0:
            break
```

Listing 4.4: Data Receiving

The receiving part of the code is a loop with a `try` inside, that will wait for data to come. The function `recvfrom()` from `socket`, returns a bytes object pair `(data, address)` that are the data received and the address of the socket that sent it, that will identify which ESP32 is sending the information. This is attributed to two variables that will be processed later. If no data is received during one second (defined in the multicast configuration function, showed in section 4.1), an exception will raise and the program will go to the `except` part, which will print an time out message on the console and sum the counter, until it reaches five timeouts, then the program will return to the outer part of the loop, that is the timestamp synchronization sending part, to restart the data sending in the ESP32. If no exception raises, that is, if data is received within the 1 second tolerance, the program executes the `else` part. This part will print the received message on the console and call the function `data_processing`.

4.4 Data Processing

The data vector sent by ESP32 is composed by the sensor measure and its Id, which are separated by "##" as follows: `"measure##id"`. Each sensor information is separated by "$$" like `"sensor1$$sensor2$$"`, and so on for the remaining sensors and at the end is located the timestamp, also separated by "$$" and a `count_id` ta. Together with the data, the `recvfrom()` function also returns the IP address of the source device in the same vector, that will identify which ESP32 sent it. The last octet of the ESP's IP address will be defined by rotary switches, that will allow to manually associate an ESP32 with an unique IP.

An example of the integral data vector is shown bellow:

```
['123$$322$$133$$83$$132$$123$$321$$223$$12341235.12312
  $$count_id', IPaddress]
```

Listing 4.5: Data vector example

To verify if the recently arrived packet has new information, the data is compared to a backup dictionary, that has the backup data from the various modules. This dictionary will have listed the last data received from each ESP32, that are index by the corresponding IP address of the ESP. If the data received now is different than the past one recorded on the dictionary, then the program advances towards the processing of data and send to the database and also assign this new data to the dictionary, substituting the old information indexing by the ESP32 IP address. If not, the program just acknowledge it and ignores the data.

Once the data is received, it will be processed like the following: the data from each sensor along with the address of its module and timestamp are separated from the vector and assigned to individual variables. A data counter is also added to identify the document and count it. Then this data is organized into a JSON format document for MongoDB, along with the timestamp and the module address.

Now, this document is stored in a list called `dataList` that accumulates data for recording in the database later. These steps are shown below in the code 4.6.

```
post_1 = {
            'Timestamp': timestamp,
            'Module Address': IPaddress,
            'Data counter': count_id,
            'Sensor 1': data1,
            'Sensor 2': data2,
            'Sensor 3': data3,
            'Sensor 4': data4,
            'Sensor 5': data5,
            'Sensor 6': data6,
            'Sensor 7': data7,
            'Sensor 8': data8
        }
dataList.append(post_1)
```

Listing 4.6: Document organization for the database

4.5 Database Communication and Writing

Finally, a database was implemented. At this point, it was decided to use the MongoDB database, which was hosted on the cloud through the MongoDB Atlas service. After installing and setting the mongoDB package with Python, the Raspberry Pi connection to the database was implemented. However, the low speed of data recording in the database caused a problem with the data receiving, because the code implemented dedicate the program to perform the recording task when requested and get stuck, preventing Raspberry Pi from other tasks. This

way it becomes unviable for the purpose, because as soon as the receive buffer fills up, the data that reaches the Raspberry will be lost. To solve this problem, an increase of the reception buffer size was needed and also store a number of packets before posting in the database, recording them all at once to save time instead of write one document at a time. The buffer size was increased to 16 MB and this process is shown in code 4.7

```
sock.setsockopt(socket.SOL_SOCKET, socket.SO_RCVBUF,
    16777216)
```

Listing 4.7: Raspberry Pi Receive Buffer Increase Setting

For initial testing, a standard Windows PC hosting the MongoDB database was used, which was communicating by Ethernet cable on a local network between the two. The PC plays the role of the database server, while Raspberry PI is the client communicating and storing data in the database. This required installing MongoDB on PC and the pymongo library on Raspberry for Python. After that, the database needs to be initiated on the PC and the access from remote systems to the server enabled. To do this one must bind an IP to Mongo database, which will be the host address accessed by the remote device and can be configured in the server configuration file in the Mongo installation directory. The MongoDB Compass tool was very helpful in this step as well, making it possible to access the created databases graphically, without the need for any extra programming.

```
client = MongoClient('127.0.0.1', 27017)
db = client.esp_data
global posts
posts = db.sensors
```

Listing 4.8: Mongo Test Configuration

Establishing a connection to MongoDB is pretty straight forward in Python. The first thing is to import `pymongo` library. Then call `MongoClient()`, that will establish the connection to the referred IP and port in the object argument. Code 4.8 shows the IP used in the first test stage and this is the only thing that changes to connect to the actual database used in the complete system. After that, line 30 creates the name of the database. In this case, `client.esp_data` creates the database named "esp_data" and assigns to the `db` variable. In the end, `db.posts` creates a collection named "posts", assigning to the variable `posts`.

After the successful test with the local database, the cloud-based MongoDB Atlas database was used as definitive database. To use it, all that was needed was to change the host address in the `MongoClient()` argument, as shown in code 4.9.

```
client = pymongo.MongoClient(
    "mongodb://login:password@on-surf-shard-00-00-obkot.
       mongodb.net:27017,"
    "on-surf-shard-00-01-obkot.mongodb.net:27017,"
    "on-surf-shard-00-02-obkot.mongodb.net:27017/test?
       ssl=true&replicaSet=On-Surf-shard-0&authSource=
       admin"
    "&retryWrites=true&w=majority")
db = client.esp_data
global posts
posts = db.sensors
```

Listing 4.9: Mongo Definitive Configuration

With a connection to the database established, it is possible to advance to the writing of data in the database. This step is simple and is shown below in the piece of code 4.10. The function `insertmany()` is taking the data in list `dataList`

referenced in 4.6 and writing to the `esp_data` database in the collection `sensors` (shown in the code 4.9). After the recording, the `dataList` is cleared to accommodate more data.

```
posts.insert_many(dataList)
dataList=[]
```

Listing 4.10: Writing data to MongoDB

Using a visual tool like MongoDB Compass or the MongoDB Atlas webpage, make the visualization of posted data very easy. In figure 4.2 shows data visualization in these tools. It is important to note that the `_id` in the first line of each document is generated automatically by Mongo and it is unique.

```
_id: ObjectId("5e1a9beb74fece059d8f636e")
Timestamp: "1578802122.044694"
Module Address: "192.168.137.50"
Data counter: "1"
Sensor 1: " 3.30"
Sensor 2: " 3.30"
Sensor 3: " 0.42"
Sensor 4: " 0.21"
Sensor 5: " 3.30"
Sensor 6: " 1.69"
Sensor 7: " 0.38"
Sensor 8: " 2.84"
```

Figure 4.2: Visualization of data in MongoDB Compass.

Chapter 5

Tests and Analysis

This chapter presents some experimental tests performed to verify the project meets the expected objectives and solves the proposed problem. Some problems faced during development are also explained.

5.1 Tests

All the test were performed in the same platform, composed of the ESP32, a POE switch, the Raspberry Pi and a PC to monitor the device and as a server connecting to the internet to access the cloud database.

Over the test phase, was acknowledged that packet losses are extremely rare after the increase in the receive buffer size in the Raspberry Pi. When letting the system working for over a week, no packet losses were detected.

To perform data flow tests, the Wireshark software was used to monitor the Ethernet network where the transmission happens. Wireshark is a network protocol analyzer that captures packet traffic on the network [40]. From the transmission speed test, an average packet transmission speed around 2000 pps (packets per second) was detected. Figure 5.1 shows the statistics taken from the Wireshark

analysis.

Statistics

Measurement	Captured
Packets	4201723
Time span, s	2100.059
Average pps	2000.8
Average packet size, B	124
Bytes	519733731
Average bytes/s	247 k
Average bits/s	1979 k

Figure 5.1: Statistics from Wireshark test

It was also tested the speed to write data in the cloud database. This was done by acquiring the clock time right before and after the recording process. Testing the recording of 2000 documents at a time, a mean time of 0.8935 seconds was measured, with an average maximum time of 2.936 seconds registered. As the test depends on internet connection sometimes it can take a while to write the data to the database, taking into account that the speed is not guaranteed to be constant and the connection to the database can be lost occasionally.

Taking into account the measured transfer rate of 2000 pps and the receive buffer size of 16MB, it is possible to estimate the maximum number of supported ESPs. Assuming that the time taken to write data is 5 seconds, the Raspberry Pi buffer could theoretically support more than 13 concurrent ESPs without having any packet loss issues due to full buffer. Given that this time is greatly extrapolated, the actual capacity is probably greater.

Chapter 6

Conclusions

A data concentrator was developed in the course of this dissertation, capable to gather sensor data, digitized and sent by the ESP32 microcontroller. As the system is designed to be modular and scalable, the number of modules may increase, i.e the amount of microcontrollers sending data is variable. In addition, the concentrator manages the information and writes it to a database, while also responsible for synchronizing the microcontroller clock to guarantee the correct post-analysis.

The project that this thesis is included wishes to develop sensed surfaces for applications in metal stamping and plastic injection molding and planned in the context of the ON-SURF program that aims at the development of technologies in the area of surface engineering. The project aims to build a DAQ system, where two other dissertations are involved. One of these works deals with the sensor signal conditioning module to be received by an ESP32. The other deals with the development of a data acquisition module using the ESP32 that reads, digitizes and sends the data to the concentrator. By being a part of this larger project, the present dissertation must be integrated with the other two that created the other modules composing the DAQ system.

The consistency of the system was verified, meeting the specifications of the

objectives and resulting in a robust system. However, it was not possible to do actual scalability tests due to the lack of extra ESP32 to test the operation together. It is also important to note that the measured data were simulated, as the definitive sensors to perform the measurements were not ready. Through the tests performed, it was concluded that packet losses are almost nonexistent and will not impact the behaviour of the system. Another thing to note is that the choice to write multiple data at once in the database was good to decrease payload time connecting with the database, improving the speed of the writing process.

There are several points relevant to future works, contributing to the continuity of the ON-SURF project, like testing the real scalability limit of the system when connecting more ESP32. Afterward, the analysis and graphical visualization of the data are still to be developed and the complete integration of the data acquisition system shall be done.

Although the system was inspired by the molds' context, it could be adapted to any application where a data acquisition system is required to acquire a sensorial signal. Changing the gain and offset values in signal conditioning, calibrating to the specifications of the sensors to be used, would be the only point to modify to suit the new system. With the objective of promoting better quality control of the final product, through the development of a system with a modular and scalable architecture, capable of measuring temperature and pressure values on mold surfaces, the project contributes in a technical way to achieve this improvement.

Bibliography

[1] H. Karbasian and A. Tekkaya, “A review on hot stamping,” *Journal of Materials Processing Technology*, vol. 210, pp. 2103–2118, Jul. 2010.

[2] H. Güler and R. Ozcan, “Comparison of hot and cold stamping simulation of usibor 1500 prototype model,” *Indian Journal of Engineering and Materials Sciences*, vol. 21, pp. 387–396, Aug. 2014.

[3] *Fairlawn tool inc. - hot vs cold metal stamping*, `https://www.fairlawntool.com/blog/hot-vs-cold-metal-stamping/`, [Online; accessed in 2019].

[4] N. Rodrigues, P. Leitão, and E. Oliveira, *Self-interested service-oriented agents based on trust and QoS for dynamic reconfiguration.* Nice, France, 2015, vol. 594. DOI: `10.1007/978-3-319-15159-5{_}20`.

[5] P. L. N. P. Lorenzo Stroppa Nelson Rodrigues, “Quality Control Agents for Adaptive Visual Inspection in Production Lines,” in *IEEE Industrial Electronics Society (IECON12)*, 2012.

[6] M. Abdallah and O. Elkeelany, “A survey on data acquisition systems daq,” *International Conference on Computing, Engineering and Information*, Apr. 2009.

[7] V. J. Brusamarello and A. Balbinot, *Instrumentação e Fundamentos de Medidas, V. 2*, 1st ed. Rio de Janeiro: LTC, 2007, ISBN: 978-85-216-1563-7.

[8] V. J. Brusamarelo and A. Balbinot, *Instrumentação e Fundamentos de Medidas, V. 1*, 2nd ed. Rio de Janeiro: LTC, 2010, ISBN: 9788521617549.

[9] N. Instruments, *What is signal conditioning*, `https://www.ni.com/pt-pt/innovations/white-papers/09/what-is-signal-conditioning-.html`, [Online; accessed in 2019].

[10] B. Carter and R. Mancini, "Chapter 14 - interfacing a transducer to an analog to digital converter," in *Op Amps for Everyone (Fifth Edition)*, B. Carter and R. Mancini, Eds., Fifth Edition, Newnes, 2018, pp. 167–175, ISBN: 978-0-12-811648-7. DOI: `https://doi.org/10.1016/B978-0-12-811648-7.00014-5`. [Online]. Available: `http://www.sciencedirect.com/science/article/pii/B9780128116487000145`.

[11] T. Schulz, "Válvulas proporcionais em malha fechada de controle de gases inertes na automação de processos," Tech. Rep., 2015.

[12] R. Abbasi *et al.*, "The icecube data acquisition system: Signal capture, digitization and timestamping," *Nuclear Instruments and Methods in Physics Research A 601 (2009) 294–316*, Jan. 2009.

[13] U. Hansmann, L. Merk, M. S. Nicklous, and T. Stober, *Pervasive Computing*, 2nd ed.: Springer, 2003, ISBN: 978-3-642-05525-6.

[14] A. Pereira, N. Rodrigues, J. Barbosa, and P. Leitão, "Trust and Risk Management Towards Resilient Large-scale Cyber-Physical Systems," in *Proceedings of the IEEE International Symposium on Industrial Electronics (ISIE 2013)*, Taipei, 2013.

[15] P. Marwedel, *Embedded System Design.*: Springer, 2006, ISBN: 9780387300870.

[16] N. Rodrigues, E. Oliveira, and P. Leitao, "Decentralized and on-the-fly agent-based service reconfiguration in manufacturing systems," *Computers in Industry*, vol. 101, pp. 81–90, Oct. 2018, ISSN: 01663615. DOI: `10.1016/j.`

compind.2018.06.003. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0166361517306991%20https://linkinghub.elsevier.com/retrieve/pii/S0166361517306991.

[17] A. B. Bondi, "Characteristics of scalability and their impact on performance," *Proceedings of the 2nd international workshop on Software and performance*, Jan. 2000.

[18] B. C. Neuman, "Scale in distributed systems," *Readings in Distributed Computing Systems. IEEE Computer Society Press*, 1994.

[19] I. O. Porto, "Padroes e diretrizer arquiteturais para escalabilidade de sistemas," Tech. Rep., Sep. 2009.

[20] J. Traub, J. Hülsmann, S. Breß, T. Rabl, and V. Markl, *Sense: Scalable data acquisition from distributed sensors with guaranteed time coherence*, 2019. arXiv: 1912.04648 [cs.DB].

[21] P. Freemantle, *A reference architecture for the internet of things*, WSO2 White paper, 2014.

[22] H. Gabbar, A. Zidan, and M. Xiaoli, "Chapter 16 - data centers for smart energy grids," in *Smart Energy Grid Engineering*, H. A. Gabbar, Ed., Academic Press, 2017, pp. 433–452, ISBN: 978-0-12-805343-0. DOI: https://doi.org/10.1016/B978-0-12-805343-0.00016-4. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9780128053430000164.

[23] *Atis telecom glossary*, https://web.archive.org/web/20130313182217/http://www.atis.org/glossary/definition.aspx?id=6587, [Online; accessed in 2019].

[24] Y. Hu and K. Liu, "Chapter 5 - devices and technology for monitoring transmission lines," in *Inspection and Monitoring Technologies of Transmission Lines with Remote Sensing*, Y. Hu and K. Liu, Eds., Academic Press,

2017, pp. 281–508, isbn: 978-0-12-812644-8. doi: https://doi.org/10.1016/B978-0-12-812644-8.00005-9. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9780128126448000059.

[25] R. Bago and M. Campos, "16 - smart meters for improved energy demand management: The nordic experience," in *Eco-Friendly Innovation in Electricity Transmission and Distribution Networks*, J.-L. Bessède, Ed., Oxford: Woodhead Publishing, 2015, pp. 339–361, isbn: 978-1-78242-010-1. doi: https://doi.org/10.1016/B978-1-78242-010-1.00016-1. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9781782420101000161.

[26] S. B. Navathe and R. Elmasri, *Sistemas de Banco de Dados - Fundamentos e Aplicações*, 4th ed. Lapa, SP, Brasil: Pearson Education do Brasil Ltda., 2002.

[27] *Banco de dados - relacional vs não relacional*, https://blog.totalcross.com/pt/banco-de-dados-relacional-nao-relacional/, [Online; accessed in 2019],

[28] E. Plugge, P. Membrey, and T. Hawkins, *The Definitive Guide to MongoDB - The NoSQL Database for Cloud and Desktop Computing.* Apress, 2010, isbn: 978-1-4302-3052-6.

[29] *About python*, https://www.python.org/about/, [Online; accessed in 2019].

[30] Z. Parker, S. Poe, and S. Vrbsky, "Comparing nosql mongodb to an sql db," Apr. 2013. doi: 10.1145/2498328.2500047.

[31] R. Cruz Huacarpuma, R. T. De Sousa Junior, M. T. De Holanda, R. De Oliveira Albuquerque, L. J. García Villalba, and T.-H. Kim, "Distributed data service for data management in internet of things middleware," *Sensors*,

vol. 17, no. 5, 2017, ISSN: 1424-8220. DOI: 10.3390/s17050977. [Online]. Available: https://www.mdpi.com/1424-8220/17/5/977.

[32] *Raspberry pi 3 model b+*, https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/, [Online; accessed in 2019].

[33] *Raspbian official website*, https://www.raspbian.org, [Online; accessed in 2019].

[34] A. Silberschatz, P. B. Galvin, and G. Gagne, *Operating System Concepts*, 7th ed. John Wiley & Sons, 2004, ISBN: 9780471694663.

[35] *Nosql databases*, https://nosql-database.org, [Online; accessed in 2020].

[36] L. Pouzin, "Cigale, the packet switching machine of the cyclades computer network," *Proceedings of IFIP, Stockholm, pp 155-159.*, Aug. 1974.

[37] P. Leitao, N. Rodrigues, C. Turrin, A. Pagani, and P. Petrali, "GRACE ontology inteGrating pRocess and quAlity Control," in *IECON 2012 - 38th Annual Conference on IEEE Industrial Electronics Society*, 2012, pp. 4348–4353. DOI: 10.1109/IECON.2012.6389189.

[38] *Introduction to mongodb*, https://docs.mongodb.com/manual/introduction/, [Online; accessed in 2019].

[39] *Mongodb sharding*, NI White Paper - https://docs.mongodb.com/manual/sharding/, [Online; accessed in 2019].

[40] Wireshark, *About*, https://www.wireshark.org/, [Online; accessed January-2020].

Appendix A

Code Developed

The full developed code for the concentrator can be accessed in `https://github.com/j-girardi/Tese.git`.