Development of a data acquisition system for educational laboratory chemical reactors

Luana Riffel Bremm

Dissertation presented to the School of Technology and Management of Bragança to obtain the Master Degree in Industrial Engineering.

Work oriented by:

Prof. Dr. José Luís Sousa de Magalhães Lima

Prof. Dra. Maria Olga de Amorim e Sá Ferreira

Prof. Dr. Rafael Cardoso

Bragança

2021-2022

Development of a data acquisition system for educational laboratory chemical reactors

Luana Riffel Bremm

Dissertation presented to the School of Technology and Management of Bragança to obtain the Master Degree in Industrial Engineering.

Work oriented by:

Prof. Dr. José Luís Sousa de Magalhães Lima

Prof. Dra. Maria Olga de Amorim e Sá Ferreira

Prof. Dr. Rafael Cardoso

Bragança

2021-2022

Dedication

First and foremost, I want to thank my parents, Dirlei and Leonir Bremm, for their unconditional support of my academic career and for always being there for me when I needed them the most. Because of their unfailing love, encouragement, and assistance. This work is dedicated to you.

To IPB and its faculty members for welcoming me as a foreign student and providing all the assistance I required throughout my stay in Portugal. To UTFPR and its academic members for consistently sharing their knowledge.

To my supervisors, thank you for your time and effort in putting this work together.

To my friends and my boyfriend, I want to thank you for your unwavering support, strength, love and assistance.

To all those who directly or indirectly were part of my academic training, thank you very much.

Abstract

Data acquisition systems aim to acquire data related to a process, whether chemical, physical or biological, and show it clearly to the user. In chemical processes, the visualization of data referring to the process can be the key point for the success or failure of the same. Thereby, the work presents the development of a data acquisition system that is already implemented in the Chemical Processes Laboratory of Instituto Politécnico de Bragança (IPB) in Bragança, Portugal. It allows to support students in real-time acquisition and recording of chemical reactor variables, such as conductivity and flow-rate. This system, based on a common 8-bit microcontroller and an application developed in Lazarus, is in operation, allowing the validation of the proposed objectives.

Keywords: Data acquisition system; Chemical reactors.

Resumo

Os sistemas de aquisição de dados têm por objetivo adquirir os dados relativos a um processo, seja ele químico, físico ou biológico e mostrá-los de forma clara para o usuário. Em processos químicos, a visualização dos dados referentes ao processo podem ser o ponto chave para o sucesso ou a falha do mesmo. Dessa forma, o trabalho apresenta o desenvolvimento de um sistema de aquisição de dados que já se encontra implementado no Laboratório de Processos Químicos do IPB em Bragança, Portugal. Este permite apoiar os alunos na aquisição em tempo real e registro de variáveis de um reator químico, tais como a condutividade e o fluxo. Este sistema, baseado num microcontrolador de 8 bits comum e numa aplicação desenvolvida em Lazarus encontra-se em funcionamento permitindo validar os objetivos propostos.

Palavras-chave: Sistema de aquisição de dados; Reatores químicos.

Contents

List of Tables

List of Figures

Acronyms

ADC Analog-to-Digital Converter.

BR Batch Reactor.

CSTR Continuous Stirred Tank Reactor.

CSV Comma-Separated Values.

DAQ Data Acquisition.

GUI Graphical User Interface.

I/O Input/Output.

IDE Integrated Development Environment.

IIR Infinite Impulse Response.

IPB Instituto Politécnico de Bragança.

LabVIEW Laboratory Virtual Instrument Engineering Workbench.

PC Personal Computer.

PFR Plug Flow Reactor.

RAD Rapid Application Development.

SCADA Supervisory Control and Data Acquisition.

USB Universal Serial Bus.

VIM International Vocabulary of Metrology.

Chapter 1

Introduction

This chapter discusses the work's motivation, objectives, and document structure.

1.1 Motivation and framework

The study reported in this thesis was prompted by a problem that arose at the chemical processes laboratory of IPB in Bragança, Portugal. A chemical reactor is commonly utilized by IPB students and teachers in its facilities. There was originally a signal conditioning system and a data collecting system in place to allow users to receive needed data from the operation being done.

Because the chemical reactors system is about twenty years old, the functions of the respective data acquisition system no longer meet the educational goals of the laboratory.

The only data being gathered are the conductivities as a function of time. The system contains four reactors, and only one reactor's conductivity may be monitored at a time. Furthermore, there is no system that permits data to be stored, thus everything must be recorded manually.

1.2 Objectives

This document will look into the chemical reactors data collection challenge and the need for additional features in it.

The data acquisition system must be capable of capturing the voltage values leaving the reactor, sending them to a microcontroller, converting them into a digital form that can be used by a developed application, and then converting them back into the original variables, which in this case are conductivity and flow-rate. This must be accomplished in the most economical way possible. As a result, there must be no replacement of reactor parts, and it is preferred to use equipment available at the university and free software.

It is intended that the system would provide a useful and reliable solution for monitoring and saving data so that it can be used later.

1.3 Document structure

This document is divided into six chapters, beginning with the current one, which contains the work proposal, contextualization, and objectives.

Chapter 2, entitled "State of the art and study of tools", presents a bibliographic review, that addresses the essential topics required to understand the concepts and to carry out the work.

Chapter 3, entitled "System architecture" offers an overview of the chemical reactor's system, illustrating its link with the hardware and software.

Chapter 4, entitled "Development work" displays the development work of the hardware and software, as well as some tests performed on the reactor's outputs.

Chapter 5, entitled "Results" describes the tests performed using the built software to ensure its functionality.

Finally, Chapter 6, entitled "Conclusion and Future Work" summarizes the work and identifies future work.

This document also has two appendices, Appendix A presents the Arduino coding and

Appendix B describes the application structure and coding.

Chapter 2

State of the art and study of tools

This chapter will give a literature review on measuring systems, sensor elements, signal analysis, Data Acquisition (DAQ) systems, and chemical reactors. In addition, tools and softwares utilized in data collecting and DAQ systems are provided, with advantages and downsides for each.

2.1 Measurement techniques

The term signal refers to something that conveys information and is represented by a function of independent variables [1]. An analogue signal is one that varies continuously throughout time. A discrete-time signal is defined as a series of samples obtained sequentially over a certain time period [2]. Signal analysis is significant in many fields, including medicine, engineering, and agronomy. Although the numbers or measures that form the signal may appear random, they have the capacity to provide crucial information about the item or location of investigation. Signal analysis and processing are used to extract information from the signal that may be used for other reasons.

Instrumentation techniques must be used to determine the standards of each piece of equipment in order to acquire data and create the signal. According to Bentley [3], a process is a system that generates information, such as a chemical reactor, an automobile, or a weather system. Measurement is the foundation of the experimental process in

processes that must be regulated. Before beginning the measurement, it is necessary to understand everything about the physical amount that will be researched. Furthermore, a technique of measurement must be determined, not only to ensure that the measurement is accurate, but also to ensure that it can be replicated by anybody [4].

According to the International Vocabulary of Metrology (VIM), quantity is a "property of a phenomenon, body, or substance, where the property has a magnitude that can be expressed as a number and a reference". The goal of measuring is to ascertain the value of a quantity using specified methods of measurement. The measurement method is made up of logic sequences of operations used to perform the measure, which are defined in the measurement procedure, which includes a complete version of all measurement principles [5]. Measurement systems are generally used to measure physical and electrical quantities such as mass, temperature, pressure, level, flow, capacitance and voltage, and they are frequently employed as part of a control system [6]. A common structure of a measurement system is show in Figure 2.1.

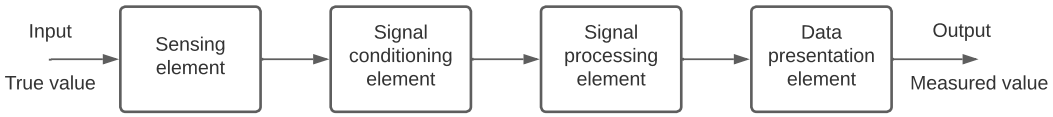

Figure 2.1: Structure of a measurement system. Adapted from [3].

Sensing elements, as seen in the picture, are the initial component of a measuring system. They are used to capture information and value from the input and form signals. Transducers are commonly used sensing devices in measurement systems to transform an input (any physical, chemical, or biological variable) into an output represented by an electric signal (current, voltage or power). According to the VIM, a measuring transducer is a "device, used in measurement, that provides an output quantity having a specified relation to the input quantity".

Sensors are categorized into several categories based on what they are used to measure, their technological aspects, conversion processes, materials and fields of use, among other factors. The conversion phenomenon is the most often used categorization, in which

sensors are classed as physical, chemical, or biological [7].

The output of a measurement system's sensor has small amplitude and it is noisy, therefore it must be processed before being displayed to the observer or used in a control system. To do this, a signal conditioning circuit is used to filter and amplify the sensor output, converting it into a format that can subsequently be used to present the original data to the observer in the form of a display or a pointer, as well as record it. This is referred to as signal processing [8].

The majority of sensors produce analogue electrical output. The output must be transformed to digital form before it can be utilized in other systems, like DAQ systems [8]. In order to do this, an Analog-to-Digital Converter (ADC) is used. Its purpose is to turn analogue quantities that characterize real-world occurrences into digital language [9].

DAQ systems are used to acquire information from some physical phenomena. The main step is to sample the signals that transform the sensor's analog value (electrical signal) to a digital value that can be manipulated by a computer [10]. A basic DAQ system is shown in Figure 2.2.

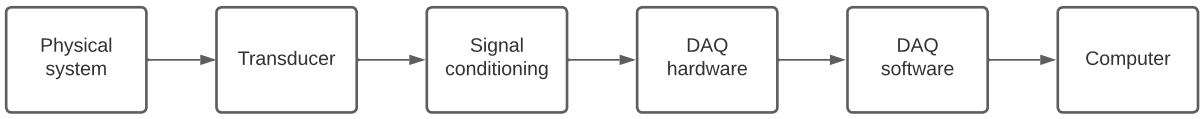

Figure 2.2: Structure of a DAQ system. Adapted from [10] and [11].

The DAQ hardware is, in general, the interface between an analog signal and a computer. It might take the shape of modules that link to the computer through serial and Universal Serial Bus (USB) ports, or cards that plug into a slot on the mother board. The DAQ software is the most important component of a DAQ system and is required for the DAQ hardware to communicate with a Personal Computer (PC). DAQ software can be created in a range of languages, including C, Python and Pascal, and tailored to the specific application in design. Alternatively, there are a variety of proprietary DAQ software programs that may be used, like LabView [10].

2.2 Chemical reactors

Chemical reactors are important pieces of equipment in the chemical industry because they turn raw materials into finished products. There are several types of chemical reactors, including Batch Reactor (BR), Plug Flow Reactor (PFR) and Continuous Stirred Tank Reactor (CSTR).

The BR is the most basic commercially utilized reactor. Its basic premise is to contain the reactants and allow them to react over time until the end product is achieved. These reactors typically have ports for injecting reactants, for removing the end product, and, in some cases, a stirring mechanism. Because certain processes require precise temperature or pressure, batch reactors might feature temperature and pressure controls [12]. An example of a BR is shown in Figure 2.3.

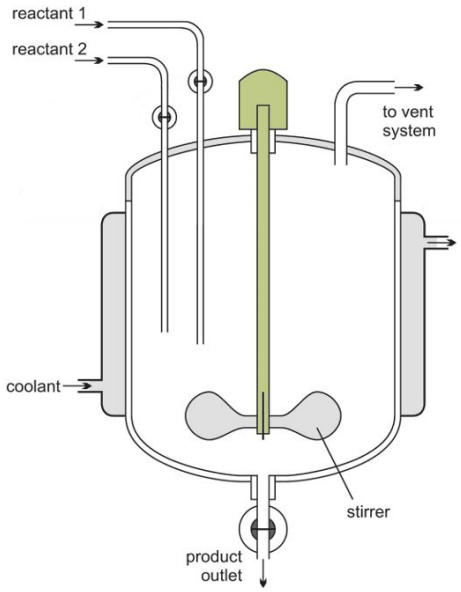

Figure 2.3: Basic BR system. Adapted from [13].

The PFR is a reactor designed with a hollow pipe or tube carrying the reactants, which can have varying diameters depending on the application, rolled inside a tank. Water circulates within the tank to keep the reactants at a constant temperature. The final product continually flows out of the reactor [14]. An example of the basic PFR system is shown in Figure 2.4.

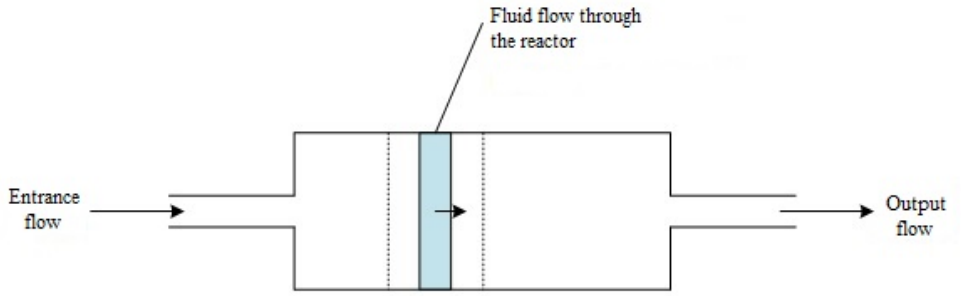

Figure 2.4: Basic PFR system. Adapted from [15].

The CSTR consists in a reactor in which the reactants are free to enter and the end product is free to escape, implying that reactants are constantly entering and the final product is constantly being withdrawn. The reactor's conditions do not vary inside the reactor, and it is termed perfectly agitated. When the reaction is too slow or the liquids demand a greater agitation rate, systems with more than one CSTR are employed [16]. An example of a CSTR is shown in Figure 2.5.

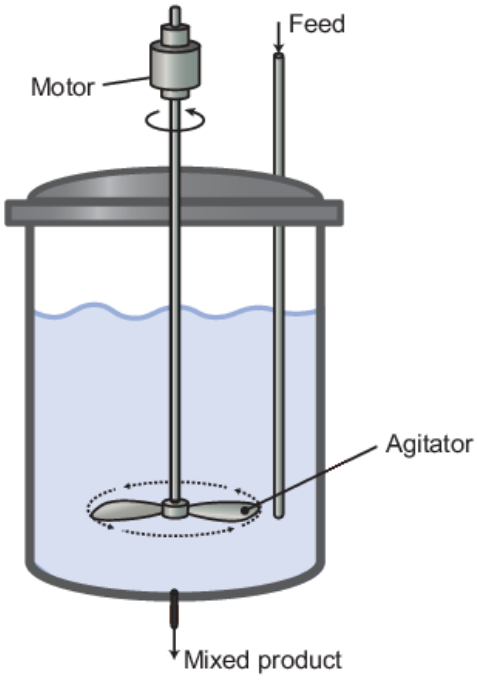

Figure 2.5: Basic CSTR. Adapted from [17].

In summary, the important variables in this type of equipments include temperature,

pressure, flow-rate and concentration of products and reactants, since these are the variables that affect the procedure.

2.3 Tools

A DAQ system consists of sensors used to capture the changing of the required variable that is being measured, signal processing circuit used to convert the signal to the appropriate form for the hardware, data acquisition hardware to collect analogue form signals and convert them to digital form for transfer to the computer, and data acquisition software to analyze and display data.

In this section, various possibilities of hardware and software that may be employed are indicated, as well as their characteristics, benefits, and disadvantages.

Given that the DAQ system's function has already been established, the hardware and software chosen must be suitable. Furthermore, it must be considered that the system will not be used on a large scale, and hence a huge investment is not desired.

2.3.1 DAQ devices

A data acquisition system must include hardware that collects analogue signals from sensors and converts them to digital format. The signal can then be passed to software and stored, processed, or displayed [8]. Depending on the software, many types of hardware, such as Arduino, Raspberry Pi, other microcontrollers, and other types of DAQ boards, can be utilized. A basic hardware used in DAQ is shown in Figure 2.6.

The analogue inputs are accessible by the circuitry through a multiplexer, as indicated. The signal is amplified before being sent to an ADC. The signal is linked to a computer once it passes through the control parcel. In practice, both the ADC and the control component are handled by a microcontroller or a DAQ board.

Each hardware can work with different softwares. Some property softwares may only work with its own specialized hardware, while open source and free softwares often use cheaper microcontrollers.

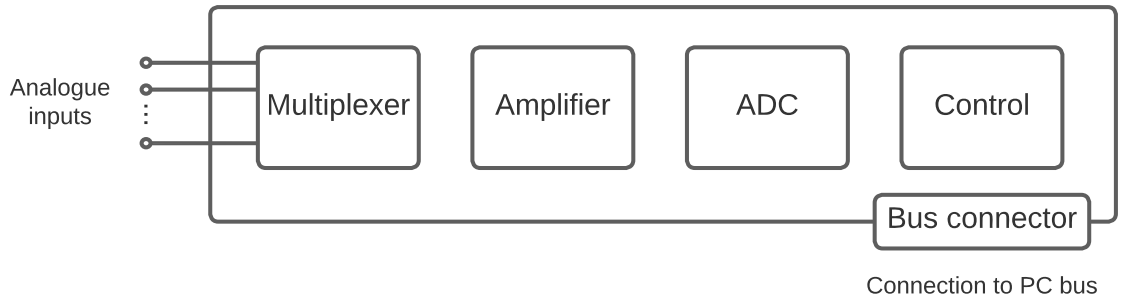

Figure 2.6: Basic elements of a DAQ hardware. Adapted from [8].

National Instruments' boards, such as the NI USB-6008, are multifunction Input/Output (I/O) devices, that provide a combination of analog I/O, digital I/O and counter/-timer functionality. They have an analog input range of ±10V, through eight analog input channels. They cost on average $200, despite being excellent DAQ boards.

A microcontroller is a microcomputer that joins hardware and software. The general microcontroller consists of the processor, the memory, serial ports and peripherals, such as timers and counters. It is commonly utilized in minor applications such as measurement and control systems. Raspberry Pi and Arduino are two of the most widely known and cheaper microcontrollers available in the market.

The Arduino Uno, one of the Arduino's models, has six analog inputs. The voltage reference can be set as default (3,3 or 5V), internal (1,1 or 2,56V) and external (voltage applied in AREF pin). It is programmed in its own Integrated Development Environment (IDE), in a language similar to C. Furthermore, it is compatible with the majority of data acquisition software.

The Arduino Uno was chosen to this work because of its adaptability, which allows it to work with a variety of software and expands the software development possibilities, and because it is less expensive than the other options.

2.3.2 Development tools

Some DAQ softwares are free and open source, others are non-free and proprietary. Some examples of softwares that can be used are LabVIEW, Lazarus, Elipse E3, MyOpenLab, Itom and FlowStone.

Laboratory Virtual Instrument Engineering Workbench (LabVIEW) is a software for monitoring via DAQ systems. It is a graphical programming environment for the visual programming language "G" from National Instruments. The G language has data flow as its principle, which makes this language fit to DAQ, instrument control and industrial automation. Some of LabVIEW benefits are that it can interact with devices or sensors and has many libraries related to Supervisory Control and Data Acquisition (SCADA) [18]. Otherwise, it is a non-free software, and in order to acquire signals there is the need of a DAQ board compatible with the software, such as the NI USB-6008 mentioned before and it generally doesn't run in microcontrollers. This software will not be utilized in this project because the Arduino Uno will be used as the hardware.

Lazarus is a Delphi compatible cross-platform IDE for Rapid Application Development (RAD) that uses a Free Pascal compiler. It is used mainly to create Graphical User Interface (GUI) applications for desktop [19], and using its packages, it can interact with most DAQ boards and microcontrollers in a simple way. Some of Lazarus benefits are that it is a free software that can be used for commercial applications, it uses visual programming, making it easier to develop the desired program and it can interact with serial devices and sensors used in DAQ systems, and also, many examples can be found online [20]. Additionally, the Arduino Uno can be used in a program built with this software. One of its disadvantages is the fact that it uses the Pascal programming language, which has some restrictions, such as the fact that for large projects, the resultant executable can be heavy for certain PCs, and it is not the easiest language to learn.

Elipse E3 is a SCADA tool, owned by Elipse Software, for monitoring and controlling processes, offering scalability and constant evolution for different types of applications. It is compatible with the majority of DAQ equipments, including Arduino Uno. It has its

own libraries, and offers advanced programming capabilities for real-time administration of industrial processes. By linking with other products, it is possible to see and operate the system using tablets and smartphones. [21]. Elipse E3 is not a free software, which makes it more difficult to make improvements to the program if they are needed and there are no skilled persons available to do so.

MyOpenLab is a free development software, that is built using visual components. It is used to create control applications that can be executed on Raspberry Pi, Arduino boards, Android, and other devices that support serial connection. It is based on the Java programming language, and its general principle is event-driven programming, in which events determine the program's flow. This software has a plethora of online examples, making it easy to create a program that does the needed purpose [22]. One of its disadvantages is that it is not widely recognized software, therefore learning to use it may be more difficult.

Itom is an open source software, maintained end extended by the ITOM community, based in Python scripting language. It is employed in the operation of measuring systems, laboratory automation and data evaluation. Its principal use is the construction and operation of sensor and measurement systems, such as in a laboratory setting. As a result, the software must be able to interface with a broad variety of hardware devices and provide a diverse and full set of assessment and data processing methods. Itom's pros is that since is a Python based software, there are multiple packages available and even on less capable computational platforms, such as outdated laptops or Raspberry Pi, it runs quickly, and there is a wealth of documentation, examples, and getting started materials accessible [23]. Its cons are that it is not a widely known software, and there is no one in IPB that has prior familiarity with it.

FlowStone is a software owned by DSP Robotics, and it is based on Ruby graphical programming language. It uses a combination of graphical and text based programming. Applications are programmed by linking together functional building blocks. FlowStone can communicate with a wide variety of external devices, such as for USB devices, I/O cards, webcams and audio hardware. Exported executables generated in the software can

be run on any PC, laptop, or even on a embedded hardware. A pro of this software is that it has built in support for many popular data acquisition devices [24]. Otherwise, it is proprietary software, making updates difficult if necessary.

Lazarus was chosen as software for this job among the softwares compatible with Arduino Uno, which was chosen as hardware and because of the benefits already mentioned, such as being free software and having numerous examples available.

Chapter 3

System architecture

This chapter will describe the chemical reactors utilized in this work, as well as its components and functionalities. It is also discussed how the reactors communicates with the hardware and software.

3.1 Chemical reactor

The stirred tank reactors in series are depicted in Figure 3.1, whose diagram was retrieved from the user handbook. In Figure 3.2, the real reactor is shown.

In the Figure 3.1, represented by the numbers, there are:

1. Rigid standpipes that provide the suction to the reactor feed pumps;
2. Reagent tank module;
3. Feed pump;
4. Feed pump;
5. Propeller stirrer;
6. Conductivity probe;
7. Dead time coil;

8. Conductivity probe of the solution leaving the dead time coil;

9. Conductivity meter;

10. Selector switch that selects which conductivity is shown on the meter;

11. Data output sockets.

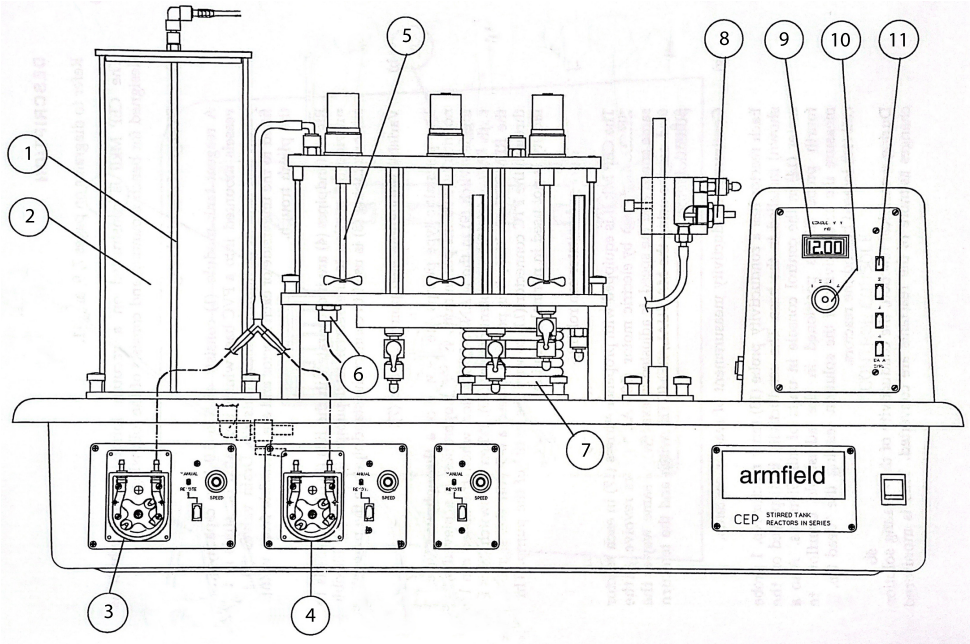

Figure 3.1: Stirred tank reactor in series. Adapted from [25].

Figure 3.3 shows a more detailed representation of the feed pumps. In the Figure 3.3, represented by the numbers, there are:

1. Toggle switch, used to select the operational mode: if manual is selected, the speed can be selected by the 10-turn potentiometer. If the remote mode is selected, it is possible to control the speed by the PTC connector, applying a 0 to 5V signal;

2. 10-turn potentiometer;

3. PTC connector, that can be used for control or data logging.

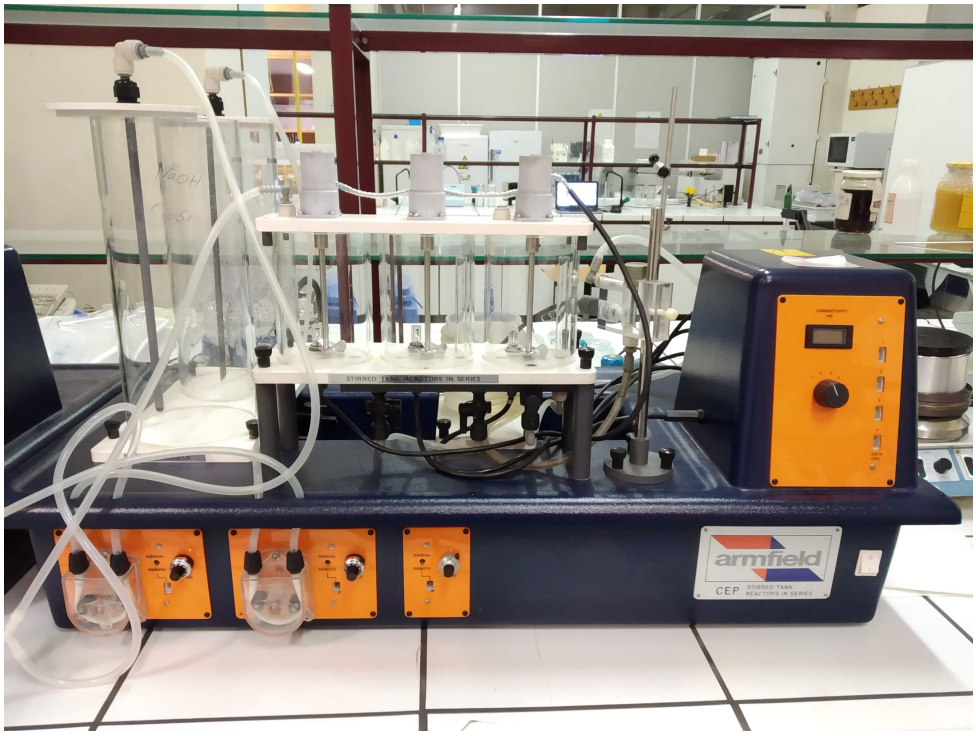

Figure 3.2: Real stirred tank reactor in series.

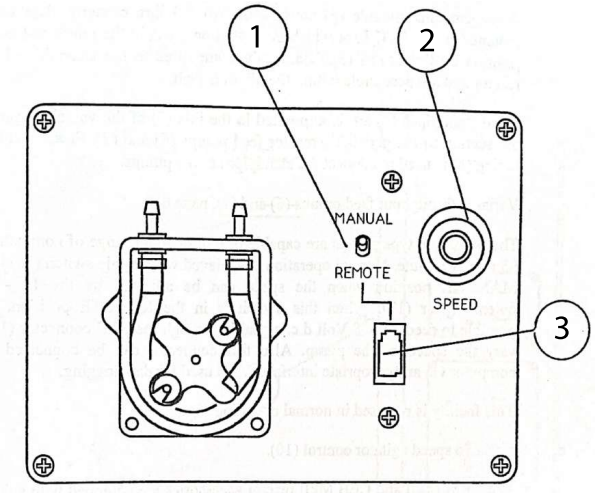

Figure 3.3: Feed pump. Adapted from [26].

Aside from the sensors, the reactor has a signal conditioning circuit that converts the signal from each sensor into a voltage that is shown in each output. The output voltage range varies depending on the sensor. The four conductivity sensors produce a voltage range of 0 to 1V, while the two flow-rate sensors output a voltage range of 0 to 5V. This

circuit is operational. The reactor originally came with a data collecting system, which is now outdated. As a result, a new data collecting system will be developed.

Coming from the data logging outputs, cables compatible to the reactor are used. Each of them has six pins, and Figure 3.4 shows a schematic of these pins and their functions.

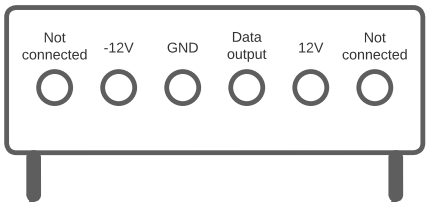

Figure 3.4: Six pin data logging cables.

The analog signal from these wires will be utilized to do the data acquisition. Because the data output is the only signal of interest, just two pins will be used: pin 3 for ground (0V) and pin 4 for the data output signal.

Depending on the voltage range of the output signal, a conditioning circuit must be utilized to properly adjust the output signal to the microcontroller input. As a result, a calibration will be performed to determinate the voltage range of the data output signal.

3.2 Hardware and software connection

There are six data signals from the reactors' four conductivity data and two flow-rate data. Each of these signals will be supplied to the microcontroller via entering it's analog inputs. Furthermore, each ground from the wires will be linked to the microcontroller's ground. The block diagram of this connection is shown in Figure 3.5.

When an analog signal is sent into the analog input, it is transformed to a digital signal via a 10 bit ADC. Following then, the sensor's value will range from 0 to 1023. The Arduino is going to be connected to a PC via USB port.

The Arduino Uno's ports are configured using the Arduino IDE. Following that, the

USB port is utilized in an application developed in Lazarus software to obtain information from the signal, multiply it by a constant in order to translate the voltage back into conductivity and flow-rate, and display it in a graph. A filter is also used to remove noise and determine the true value of the variables.

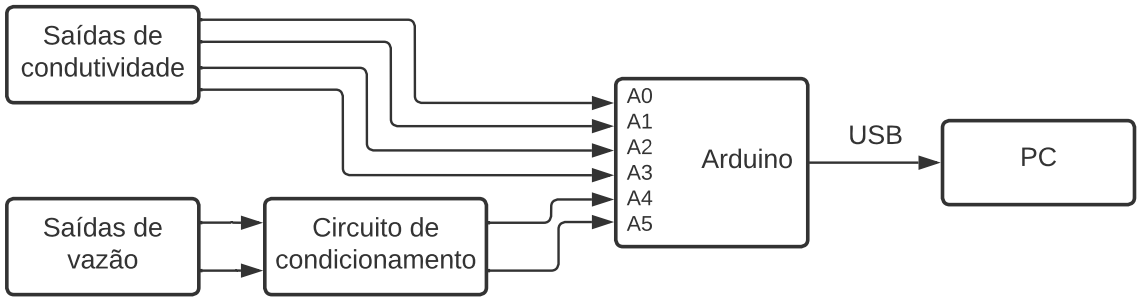

Figure 3.5: Block diagram that represents the system.

Chapter 4

Development work

The development work of the hardware and software will be detailed in this chapter, with the most important elements highlighted.

4.1 Output calibration

A calibration test was performed in order to acquire the voltage range of the data logging outputs and accurately calibrate the conditioning circuit leading to the microcontroller's inputs.

To determine the voltage range and flow rate of the feed pumps, a calibration with distilled water was performed. After adjusting the potentiometer (represented by number 2 in Figure 3.3) in all ten positions, the water was placed in a graduated cylinder. The voltage was measured with a multimeter in pins 3 and 4 of a cable equal to the one represented in Figure 3.4, which was connected to the data output of the feed pumps, represented by number 3 in Figure 3.3. A one-minute timer was set, and the final volume of water after one minute was verified. This test was carried out for both feed pumps. The flow-rate calibration values of the first feed pump are presented in the Table 4.1 and the values of the second feed pump are presented in Table 4.2.

From the tables, the flow-rate was obtained, with the value of approximately 0 to 145 mL/min, and the voltage range approximately from 0 to 5V.

Table 4.1: Flow-rate and voltage values for the first feed pump.

Potentiometer position	**Voltage (V)**	**Flow-rate (mL/min)**
1	0.518	0
2	1.028	0
3	1.538	16
4	2.049	27
5	2.563	48
6	3.070	74
7	3.581	90
8	4.100	110
9	4.610	126
10	5.020	145

Table 4.2: Flow-rate and voltage values for the second feed pump.

Potentiometer position	**Voltage (V)**	**Flow-rate (mL/min)**
1	0.520	0
2	1.068	10
3	1.583	20
4	2.092	30
5	2.607	52
6	3.122	68
7	3.631	76
8	4.150	102
9	4.670	125
10	5.050	135

With this, in order to obtain a tendency line that represents the relation between flow-rate and voltage, Excel was used. The two graphs with its own tendency line are shown in Figure 4.1. Because the flow values are not variables that require such precision when displayed to the user, a linear function was chosen rather than a quadratic function.

From this, the line equation that relates the flow-rate with the voltage for the first feed pump is shown in Equation (4.1), and for the second feed pump is shown in Equation (4.2), in which Q represents the flow-rate in mL/min and V represents the voltage in V.

$$Q_1 = 34.37 \cdot V - 32.901 \tag{4.1}$$

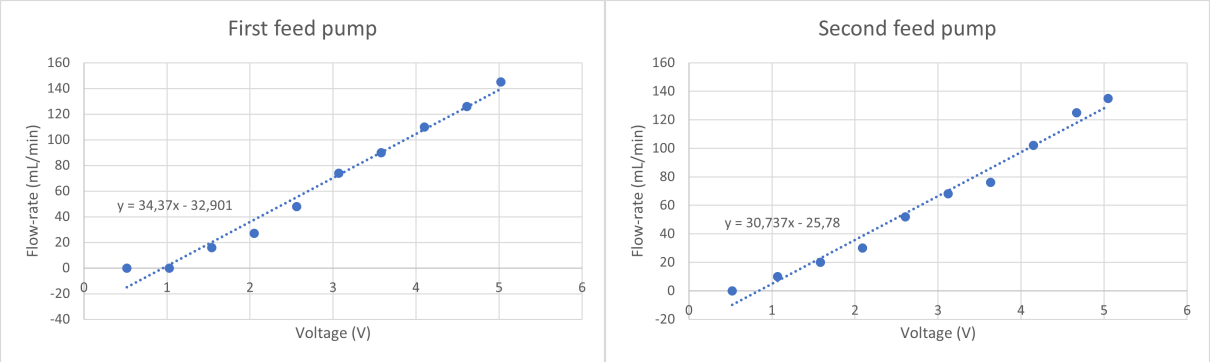

Figure 4.1: Flow-rate versus voltage: experimental data and linear fit.

$$Q_2 = 30.737 \cdot V - 25.78 \tag{4.2}$$

In order to realize the conductivity calibration, six aqueous solutions prepared with different concentration of potassium chloride KCl, ranging between 0.0 and 0.025 mol/L.

To obtain the real conductivity, a benchtop conductivity meter WTW-inoLab was used, with special conductivity measuring cells TetraCon 325. The ambient temperature was 21.1°C. The conductivity of each solution is shown in Table 4.3.

Concentration of KCl (mol/L)	**Real conductivity (mS/cm)**
0.0	0.016
0.0025	0.361
0.005	0.707
0.01	1.371
0.02	2.670
0.025	3.310

Table 4.3: Conductivity of the aqueous solutions of KCl.

Initially, each solution was deposited into the three stirred tank reactors. The reactors were then turned on, and the stirrers (represented by number 5 in Figure 3.1) initiated. The conductivity displayed for each reactor in the conductivity meter (number 9 in Figure 3.1), selected by the selector switch (number 10 in Figure 3.1), has been noted. A suitable wire equal to the one in Figure 3.4 and a multimeter were used to measure the voltage of each conductivity output. The measure was taken in pins 3 and 4 of the cable, and the

setup is shown in Figure 4.2. The results are presented in Table 4.4. Figure 4.3 shows the experimental data of the results, as well as the respective tendency line.

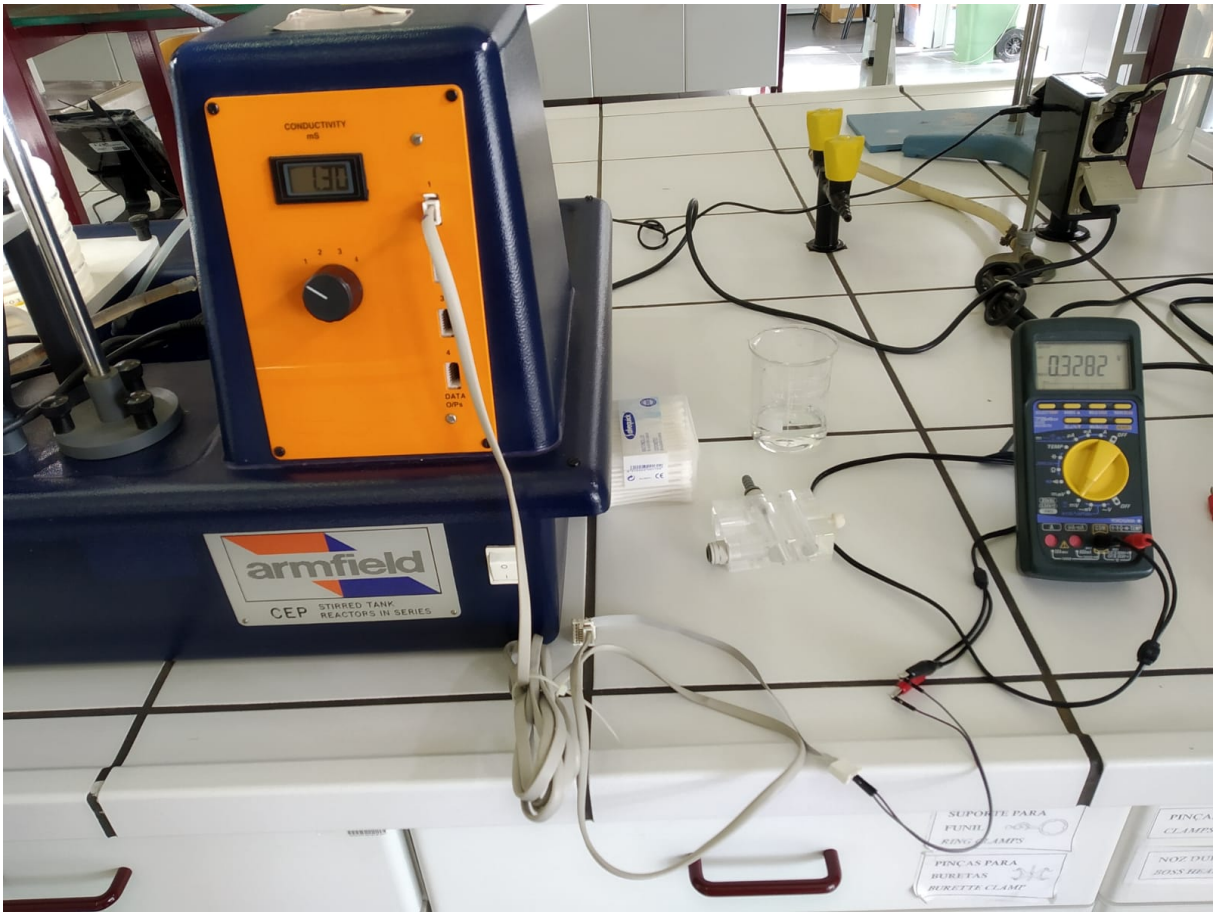

Figure 4.2: Measurement setup.

Based on the results of the testing, it was found that the voltage range of the four conductivity outputs ranges from 0 to 1V, and the voltage varies linearly with the conductivity, as shown in Figure 4.3. This way, the equation that represents it is shown in Equation (4.3), being σ the conductivity in mS/cm and V the analog voltage in V.

$$\sigma = 3.9622 \cdot V \tag{4.3}$$

Table 4.4: Measured voltages as a function of the conductivity for the aqueous solutions of KCl.

Concentrations of KCl (mol/L)	**Displayed conductivity (mS/cm)**		**Output voltage (V)**
0.0	Reactor 1	0.06	0.0143
	Reactor 2	-0.11	-0.0239
	Reactor 3	0.06	0.0146
	Dead time coil	0.08	0.0187
0.0025	Reactor 1	0.35	0.0877
	Reactor 2	0.21	0.0530
	Reactor 3	0.39	0.0984
	Dead time coil	0.38	0.0959
0.005	Reactor 1	0.67	0.1697
	Reactor 2	0.54	0.1355
	Reactor 3	0.73	0.1835
	Dead time coil	0.73	0.1817
0.010	Reactor 1	1.30	0.3284
	Reactor 2	1.16	0.2940
	Reactor 3	1.37	0.3453
	Dead time coil	1.37	0.3466
0.020	Reactor 1	2.48	0.6279
	Reactor 2	2.35	0.5936
	Reactor 3	2.57	0.6504
	Dead time coil	2.55	0.6444
0.025	Reactor 1	3.09	0.7799
	Reactor 2	2.85	0.7204
	Reactor 3	3.18	0.8049
	Dead time coil	2.95	0.7465

4.2 Hardware assembly

After the testing was completed, the hardware assembly could commence. The six voltages, two representing the flow-rates, obtained from the data output of the feed pumps (represented by number 3 of Figure 3.3) and four representing conductivities, obtained from the four data output of the reactors (represented by number 11 in Figure 3.1), needed to be connected to the microcontroller. To accomplish this, suitable cables, such as those shown in Figure 3.4, were utilized. Each cable provides a voltage signal, two signals ranging from 0 to 5V and four signals ranging from 0 to 1V.

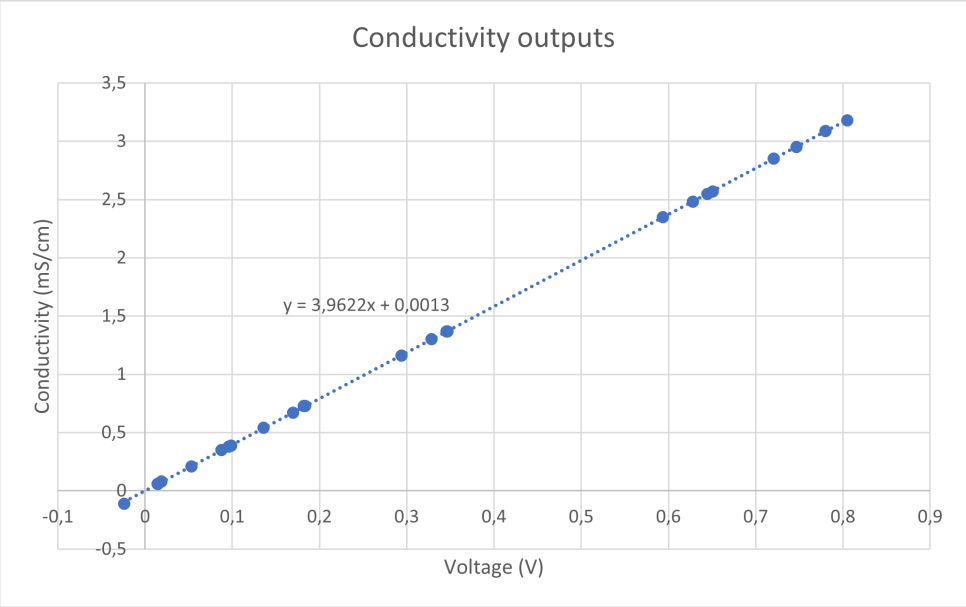

Figure 4.3: Conductivity versus voltage: experimental data and linear fit.

Given that there were four voltages with 1V maximum associated to the measurement of the conductivity and two voltages with 5V maximum associated to the measurement of the flow-rate, it was interesting to lower the 5V to 1V in order for the Arduino to use the six signals using the same voltage reference. To accomplish this, a voltage divider circuit was designed to convert the 5V to approximately 1V. Figure 4.4 illustrates the circuit, while Figure 4.5 shows the built circuit plugged into Arduino Uno. The remaining four volts are sent straight to the Arduino ports.

As a result, six voltages reach the analog ports of the Arduino Uno, each with a maximum value of about 1V. Because of this, the Arduino's reference voltage is set to 1,1V in order to obtain resolution. This is done in line 45 of the Arduino code, as shown below. The complete code can be seen in Appendix A.

```
analogReference(INTERNAL);
```

The four conductivity measures are entered in A0, A1, A2, and A3, while the two flow-rate measures are entered in A4 and A5. The sampling interval is defined by the user. The section of the Arduino code (line 78-91) that configures the inputs and sends them to the Lazarus application is shown below.

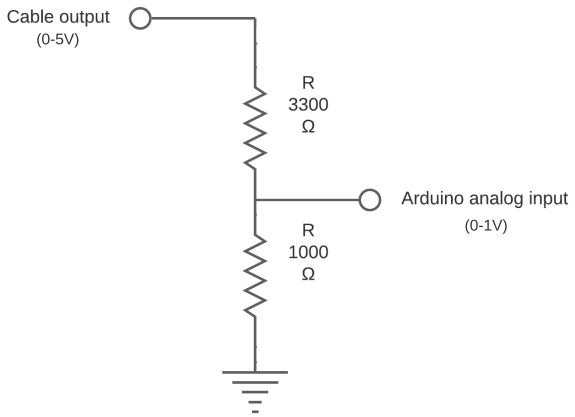

Figure 4.4: Voltage divider circuit.

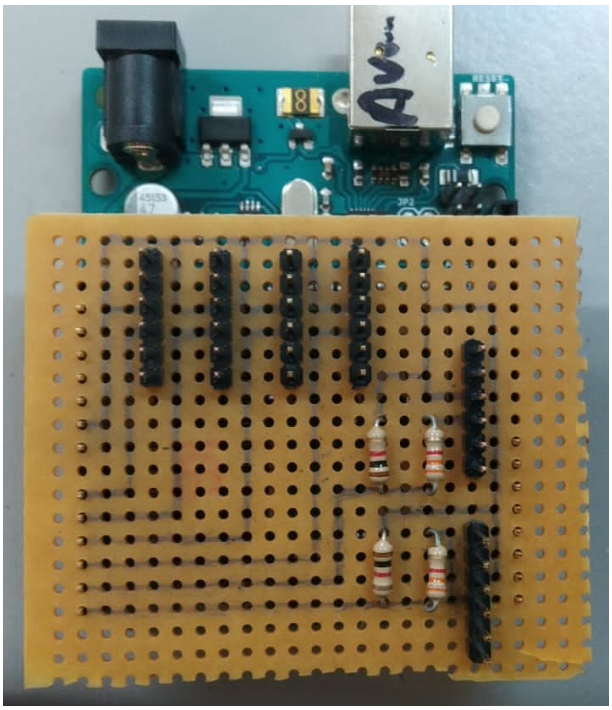

Figure 4.5: Built voltage divider circuit.

```
sample++;
    v0 = analogRead(A0);
    v1 = analogRead(A1);
    v2 = analogRead(A2);
    v3 = analogRead(A3);
```

```
v4 = analogRead(A4);
v5 = analogRead(A5);
serial_channels.send('M', v0);
serial_channels.send('N', v1);
serial_channels.send('O', v2);
serial_channels.send('P', v3);
serial_channels.send('Q', v4);
serial_channels.send('R', v5);
serial_channels.send('S', sample);
```

Because of the ADC functionality, the voltages are transformed to a value ranging from 0 to 1023 after being processed on the Arduino, and they can be used in the Lazarus application as digital signals rather than analog signals. The Arduino is attached to a USB port for this purpose, and its signals are used in the software. Each signal is send to the program represented by a letter. M, N, O and P represents the four conductivity signals, Q and R represent the two flow-rate signals.

4.3 Software development

After passing through the Arduino code, the signals are now configured to operate with the built program in Lazarus. As previously stated, each signal is routed to the application denoted by a letter. The signal is processed in the application, and the voltage is converted back to the original variable recorded by the sensors.

To do this, the numeric values ranging from 0 to 1023 must be multiplied by a factor after going through the Arduino to return to the measured voltage and then to the original variable. In the application, this is represented by the letter "m".

The relations between the flow-rate and the analog voltage were 34.37 and 30.737, respectively, as shown in Equations (4.1) and (4.2), but because the voltage is now digital, this relations should also be converted to digital by dividing it by 1023. The new flow-rate-voltage relations are represented by equation (4.4). As shown in Equation (4.5), the same should be done with the offset represented by the letter "b".

$$\begin{aligned} m_1 &= \frac{34.37}{1023} = 0.0336 \\ m_2 &= \frac{30.737}{1023} = 0.03 \end{aligned} \tag{4.4}$$

$$\begin{aligned} b_1 &= \frac{-32.901}{1023} = -0.0322 \\ b_2 &= \frac{-25.78}{1023} = -0.0252 \end{aligned} \tag{4.5}$$

This way, the new equations for the flow-rate are shown in Equations (4.6) and (4.7).

$$Q_1 = 0.0336 \cdot V - 0.0322 \tag{4.6}$$

$$Q_2 = 0.03 \cdot V - 0.0252 \tag{4.7}$$

As indicated in Equation (4.3), the relation between conductivity and analog voltage was 3.9622, but because the voltage is now digital, this relation should also be transformed to digital by dividing it by 1023. Furthermore, because this voltage spans from 0 to 1V and the Arduino's voltage reference is 1.1V, the analog voltage should also be divided by 1.1V. Equation (4.8) represents the new relation between conductivity and voltage.

$$\begin{aligned} m &= \frac{3.9622}{1023 \cdot 1.1} \\ m &= 0.003521 \end{aligned} \tag{4.8}$$

Thus, the line equation for the four conductivity signals is:

$$\sigma = 0.003521 \cdot V \tag{4.9}$$

Furthermore, a Infinite Impulse Response (IIR) filter is utilized to remove noise from the measured signal, resulting in a smoother signal. It has a value ranging from 0 to 1. When it is set to its minimal value, the signal is not filtered and is displayed as measured;

when set to its highest value, the signal is as filtered as feasible. The filter value is multiplied by the incoming signal as well as the existing array values representing prior conductivity or flow rate. The variable "q_filt" represents this filter value, ranging from 0 to 1, which is defined by the user.

After the implementation of this, the new line equation for the conductivity is presented in Equation (4.10).

$$\sigma = [(m \cdot V + b) \cdot (1 - q\_filt)] + (q\_filt \cdot \sigma_{previous}) \tag{4.10}$$

The same happens with the flow-rate, as shown in Equation (4.11).

$$Q = [(m \cdot V + b) \cdot (1 - q\_filt)] + (q\_filt \cdot Q_{previous}) \tag{4.11}$$

The conductivity value and the flow-rate value calculation is performed in lines 336-433 of the Lazarus application code (full code in Appendix B). It uses a library called `channels.h`, that communicates the Arduino signals with the Lazarus application. Below, one of the calculations is represented, and this is repeated for every analog input.

```
//MemoDebug.Text := MemoDebug.Text + channel;
if channel = 'X' then begin
  // if the arduino was reset ...

end else if channel = 'M' then begin
  A0 := value;
  V[0] := (m[0] * A0 + b[0]) * (1 - q_filt[0]) + (q_filt[0] * V[0]) ;
  StringGrid1.Cells[1,1] := floattostr(V[0]);
  StringGridCalib.Cells[5,1]:= floattostr(V[0]);
  StringGridCalib.Cells[4,1]:= inttostr(A0);
```

With this, the final values of conductivity are shown in a graph. The code that adds these values in the graph is performed in lines 340-356 of the Lazarus application code (full code in Appendix B), and it is shown below. Because flow-rate is normally a constant number, it is not represented on the graph; instead, its value for both feed pumps is only

shown in a box at the same time as measured.

```
begin
  actualaqtime := actualaqtime + AqPeriod / 1000;
  if Serial.Active then begin
    if CBgoaqc.Checked then begin
      LabelTime.Caption:='Aq time (s)='+FloatToStr(actualaqtime);
      CLA0.AddXY(actualaqtime,V[0]);
      CLA1.AddXY(actualaqtime,V[1]);
      CLA2.AddXY(actualaqtime,V[2]);
      CLA3.AddXY(actualaqtime,V[3]);
      //CLA4.AddXY(actualaqtime,V[4]);
      //CLA5.AddXY(actualaqtime,V[5]);
    end;
    if ((actualaqtime > totalAqTime) and CBgoaqc.Checked) then begin
      CBgoaqc.Checked:=false;
      SaveLog();
    end;
end;
```

Aside from the graph, which may be saved as an image, a Comma-Separated Values (CSV) file can be exported and viewed in Excel. This is essential for future data processing.

Chapter 5

Results

To verify the functioning of the produced application, two tests given in the user manual will be carried out.

The first test procedure is explained below.

1. Turn on the CEP MK-II reactor;
2. Fill one of the reagent feed vessels with distilled water;
3. Pour distilled water into each of the reactors until they are full to the standpipe overflow levels;
4. Start the water feed pump;
5. Start the stirrers;
6. Weigh 1 g of potassium chloride (KCl) and dissolve it in approximately 10 mL of distilled water. Put the solution in a syringe;
7. Quickly inject the content of the syringe into the first reactor. Simultaneously start the acquisition of conductivity data in each reactor;
8. The test will end when the conductivity of the water in reactor 3 becomes approximately zero;

9. End the test and the data acquisition.

Some frames of how the experiment was performed is shown in Figure 5.1.

Figure 5.1: Experiment procedure.

The values used in order to calibrate the conductivity and flow-rate are shown in Table 5.1. S1, S2, S3 and S4 represents the four signals of conductivity, and B1 and B2 represents the two signals of flow-rate.

Table 5.1: Calibration values used in the experiment.

	m	b	q_filt
S1	0,00352	0	0,9
S2	0,00352	0	0,9
S3	0,00352	0	0,9
S4	0,00352	0	0,9
B1	0,03360	-0,0322	0,9
B2	0,03000	-0,0252	0,9

The outcome of this experience is depicted in the graph in Figure 5.2. The green line represents the first reactor's conductivity, the red line represents the second reactor's conductivity, and the blue line represents the third reactor's conductivity. The fourth conductivity sensor, which represents the dead time coil conductivity, was not used in the first experiment. The test was carried on during one hour.

As the solution was deposited into the first reactor, the conductivity quickly increased, but soon began to decrease as distilled water entered the reactor. With some delay, the

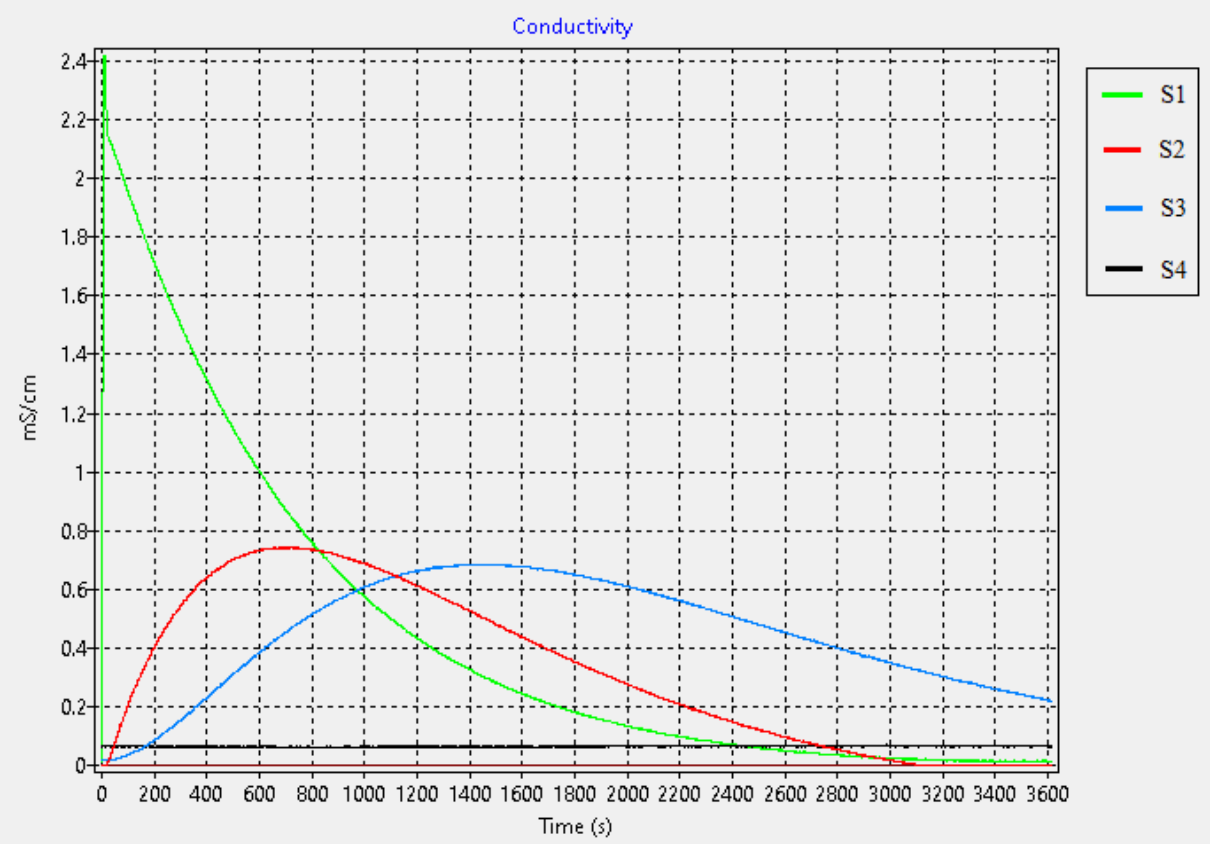

Figure 5.2: Conductivity graph of the first experience.

second reactor's conductivity began to rise as the solution from the first reactor moved to the second. The same happened with the third reactor.

The graph depicts the predicted outcome.

The second test protocol is identical to the first, except that 2 grams of potassium chloride (KCl) were used instead of 1 gram.

The graph in Figure 5.3 depicts the outcome of this experience. The green line indicates the conductivity of the first reactor, the red line represents the conductivity of the second reactor, the blue line represents the conductivity of the third reactor, and the black line represents the conductivity of the dead time coil. The test was run for 1300 seconds, until all conductivities reached zero.

As expected, the conductivity out the exit of the dead time coil was approximately the same as the third reactor, with a delay.

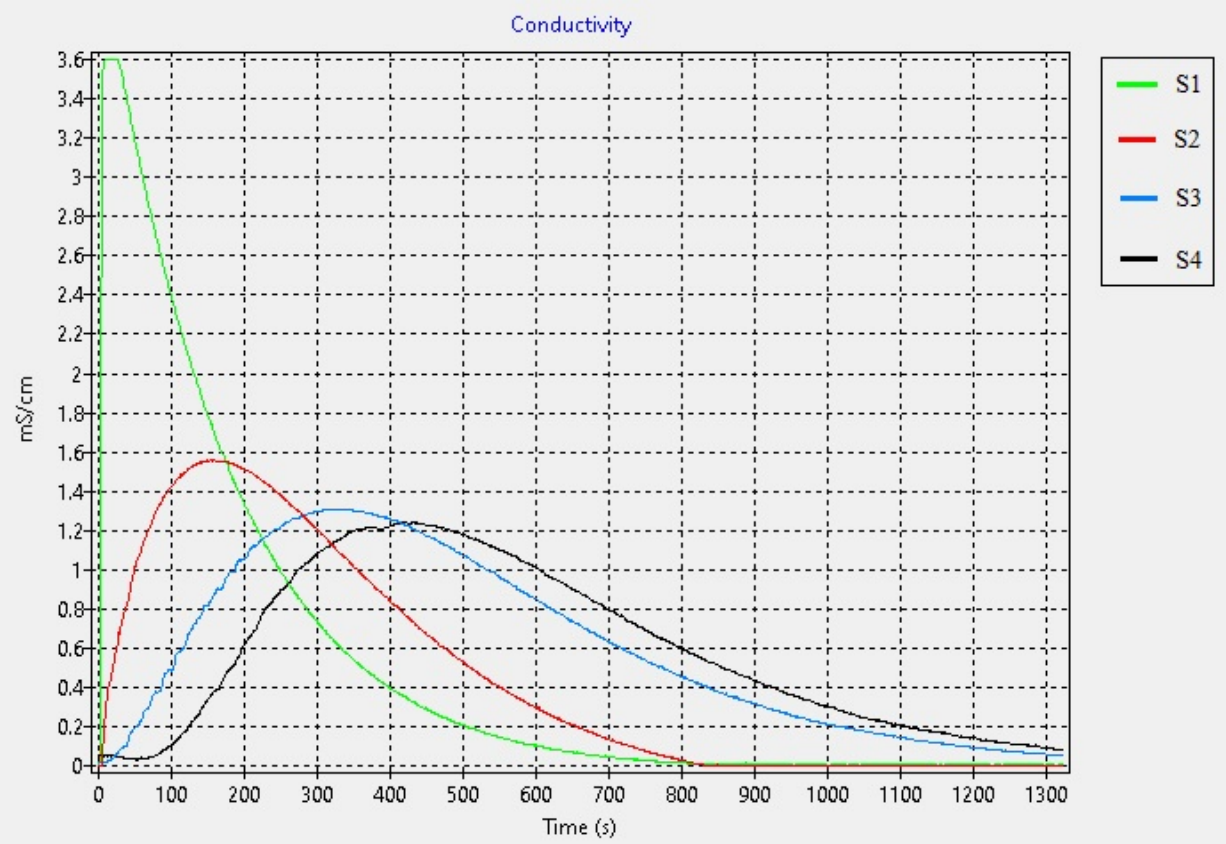

Figure 5.3: Conductivity graph of the second experience.

In Figure 5.4 it is shown how the flow-rates are displayed in the interface.

	Value
S1	0.071613152
S2	0.046565048
S3	3.849575640
S4	0.005278939
B1	130.1826903
B2	130.2547285
s	14995

Figure 5.4: Flow-rate as shown in the software.

Chapter 6

Conclusion and Future Work

A data acquisition system is an appropriate solution for dealing with data analysis and data usage problems, particularly in chemical applications where the variables must always be controlled. The current work developed a data acquisition system to obtain conductivity and flow-rate data from a chemical reactor.

The old data collecting system was studied to design this system, the cables were checked to discover which pin produced the desired signal, and several hardware and softwares were analyzed to determine their compatibility for the purpose of this work.

This document presents a solution to the data collection problem by creating a inexpensive data acquisition system that allows the user to observe data in real time, as well as paste data in a graph and a work sheet.

The implementation of a data acquisition system for educational chemical reactors was successful after the tools were chosen and based on the experimental results presented in this work. The system performed as expected after tests were carried out.

While the system did not prove to be fault-free, it did reveal to be suitable for internal use in the chemical processes laboratory in IPB.

6.1 Future work

Some ideas for future development are provided below.

- Development of a control system for the feed pumps;
- Development of a control system for the stirrers;
- Creation of a calculus system additionally to the data acquisition system in order to convert conductivity data into molar concentration;
- Adjustment of the current data acquisition system to other chemical reactors presented in the laboratory.

Bibliography

[1] A. V. Oppenheim, *Discrete-time signal processing*, 2nd ed. Pearson Education India, 1999, ISBN: 978-0130834430.

[2] V. Ingle, S. Kogon, and D. Manolakis, *Statisical and adaptive signal processing*. Artech, 2005, ISBN: 978-1580536103.

[3] J. P. Bentley, *Principles of measurement systems*, 4th ed. Harlow, England: Pearson Prentice Hall, 2005, ISBN: 0130430285.

[4] A. Balbinot and V. J. Brusamarello, *Instrumentação e fundamentos de medidas*, 2nd ed. Livros Técnico e Cientıficos Editora, 2010, vol. 1, ISBN: 978-8521617549.

[5] "International vocabulary of metrology–basic and general concepts and associated terms," *Chemistry International – Newsmagazine for IUPAC*, vol. 30, no. 6, 2008. DOI: 10.1515/ci.2008.30.6.21.

[6] R. B. Northrop, *Introduction to instrumentation and measurements*, 3rd ed. CRC Press, 2017, ISBN: 9781138071902.

[7] R. M. White, "A sensor classification scheme," *IEEE Transactions on ultrasonics, ferroelectrics, and frequency control*, vol. 34, no. 2, pp. 124–126, 1987.

[8] W. Bolton, *Instrumentation and Control Systems*. Oxford: Newnes, 2004, ISBN: 978-0750664325.

[9] W. Kester, "Section 2-1 - coding and quantizing," in *Data Conversion Handbook*, W. Kester, Ed., Burlington: Newnes, 2005, pp. 57–72, ISBN: 978-0-7506-7841-4.

[10] M. Di Paolo Emilio, *Data Acquisition System: from fundamentals to applied design.* Jan. 2013, ISBN: 978-1461442134.

[11] *Types of data acquisition systems*, visited on 31/05/2022. [Online]. Available: https://daqifi.com/types-of-data-acquisition-systems/.

[12] *Batch reactor*, visited on 29/10/2021. [Online]. Available: https://encyclopedia.che.engin.umich.edu/Pages/Reactors/Batch/Batch.html.

[13] *Chemical reactors*, visited on 14/07/2022. [Online]. Available: https://www.essentialchemicalindu org/processes/chemical-reactors.html.

[14] *Plug flow reactor*, visited on 29/10/2021. [Online]. Available: https://encyclopedia.che.engin.umich.edu/Pages/Reactors/PFR/PFR.html.

[15] *Plug flow reactors (pfr) : Mass balance expression*, visited on 14/07/2022. [Online]. Available: https://myengineeringtools.com/Chemical_Reactions/Plug_Flow_Reactor_Mass_Balance.html.

[16] *Continuous stirred tank reactor*, visited on 29/10/2021. [Online]. Available: https://encyclopedia.che.engin.umich.edu/Pages/Reactors/CSTR/CSTR.html.

[17] Y. Lu, Z. Fang, and C. Gao, "Stabilization of (state, input)-disturbed cstrs through the port-hamiltonian systems approach," *arXiv preprint arXiv:1707.01560*, 2017.

[18] *What is labview?* visited on 01/02/2022. [Online]. Available: https://www.ni.com/pt-pt/shop/labview.html.

[19] *About lazarus project*, visited on 06/12/2021. [Online]. Available: https://www.lazarus-ide.org/index.php?page=about.

[20] *Why use lazarus?* visited on 06/12/2021. [Online]. Available: https://www.lazarus-ide.org/index.php?page=whyuse.

[21] *Elipse e3*, visited on 27/04/2022. [Online]. Available: https://www.elipse.com.br/produto/elipse-e3/.

[22] *Myopenlab*, visited on 10/05/2022. [Online]. Available: https://myopenlab.org/inicio/.

[23] *Features - itom*, visited on 11/05/2022. [Online]. Available: `https://itom.bitbucket.io/features.html`.

[24] *Flowstone - overview*, visited on 26/04/2022. [Online]. Available: `http://www.dsprobotics.com/flowstone.html`.

[25] *Instruction manual cep mk ii - stirred tank reactors in series*, Armfield Limited, Nov. 1998.

[26] *Instruction manual cex - chemical reactors service unit*, Armfield Limited, Nov. 2000.

Appendix A

Arduino coding

The complete Arduino code is presented below.

```
#include <Arduino.h>
//#include <util/atomic.h>

#include "channels.h"
#include "proj_types.h"

uint16_t v0, v1, v2, v3, v4, v5, sample;

channels_t serial_channels;


unsigned long interval;
unsigned long currentMicros, previousMicros;
byte LED_state;


void serial_write(uint8_t b)
{
  Serial.write(b);
}

```

```
22
23 void process_serial_packet(char channel, uint32_t value, channels_t& obj
       )
24 {
25   //byte i;
26
27   if (channel == 'I') {
28     LED_state = !LED_state;
29     digitalWrite(LED_BUILTIN, LED_state);
30   }
31   else if (channel == 'g')  { // led13
32     obj.send(channel, value + 1);
33     Serial.println(value + 1);
34   }
35   else if (channel == 'P')  { // pwm
36     int x = value & 0xFF;
37     analogWrite(3,x);
38   }
39 }
40
41 void setup()
42 {
43   //uint8_t i;
44
45   analogReference(INTERNAL);
46
47   interval = 100UL * 1000UL; // Periodo em us
48   LED_state = 0;
49
50   pinMode(LED_BUILTIN, OUTPUT);
51   digitalWrite(LED_BUILTIN, LED_state);
52   pinMode(3,OUTPUT);
53
54   Serial.begin(115200);
55   serial_channels.init(process_serial_packet, serial_write);
```

```
  serial_channels.send('X', MCUSR);
  previousMicros = micros();
  sample = 0;
}

void loop()
{

  byte b;

  if (Serial.available()) {
    b = Serial.read();
    serial_channels.StateMachine(b);
    //Serial.write(b);
    //Serial.println();
  }

  currentMicros = micros();

  // THE Control Loop
  if (currentMicros - previousMicros >= interval) {
    previousMicros = currentMicros;
    sample++;
    v0 = analogRead(A0);
    v1 = analogRead(A1);
    v2 = analogRead(A2);
    v3 = analogRead(A3);
    v4 = analogRead(A4);
    v5 = analogRead(A5);
    serial_channels.send('M', v0);
    serial_channels.send('N', v1);
    serial_channels.send('O', v2);
    serial_channels.send('P', v3);
    serial_channels.send('Q', v4);
    serial_channels.send('R', v5);
```

```
91      serial_channels.send('S', sample);
92
93
94
95      // LED_state = !LED_state;
96      //  digitalWrite(LED_BUILTIN, LED_state);
97
98      //Serial.println();
99    }
100 }
```

Appendix B

Lazarus application coding

The complete code of the Lazarus application is shown below. The interface of the program is shown in Figure B.1.

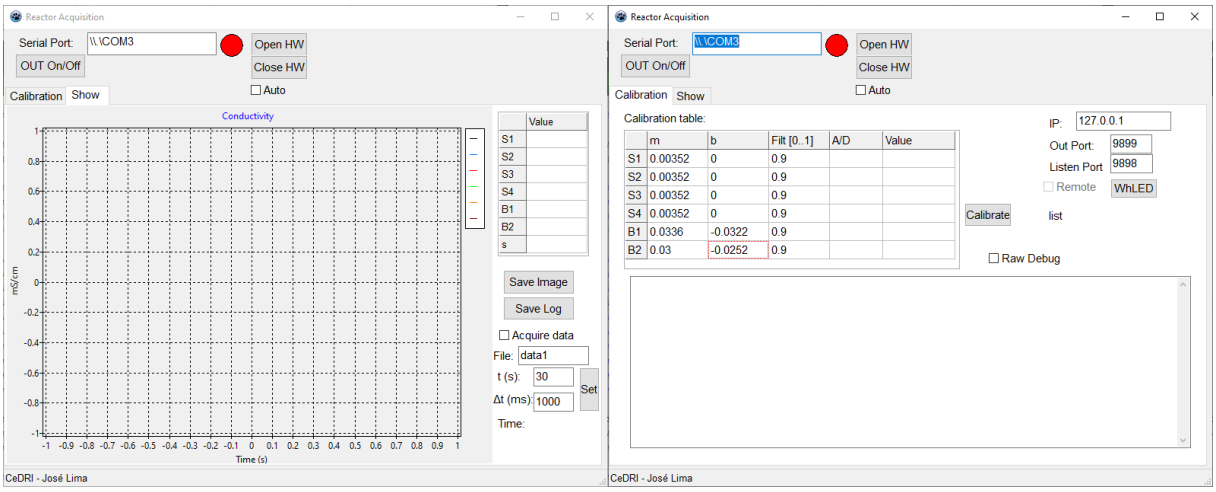

Figure B.1: Application interface.

```
unit main;

{$mode objfpc}{$H+}

interface

uses
```

```
8   Classes, SysUtils, FileUtil, TAGraph, TASeries, Forms, Controls,
     Graphics,
9   Dialogs, StdCtrls, ExtCtrls, IniPropStorage, SdpoSerial, LCLIntf,
     LCLType,
10  ComCtrls, Grids, channels;
11
12 CONST
13   MINPERIOD = 200;
14
15
16 type
17
18   { TFMain }
19
20   TFMain = class(TForm)
21     BOpenSerial: TButton;
22     BCloseSerial: TButton;
23     BSendRaw: TButton;
24     BtSetTimer: TButton;
25     ButtonCalib: TButton;
26     ButtonSaveLog: TButton;
27     ButtonSaveImage: TButton;
28     CBRawDebug: TCheckBox;
29     CBAutoOpen: TCheckBox;
30     Chart1: TChart;
31     CBgoaqc: TCheckBox;
32     CLA5: TLineSeries;
33     CLA4: TLineSeries;
34     CLA3: TLineSeries;
35     CLA2: TLineSeries;
36     CLA1: TLineSeries;
37     CLA0: TLineSeries;
38     Editcsvfilename: TEdit;
39     EditAqTime: TEdit;
40     EditSamplePeriod: TEdit;
```

```
41    EditSerialName: TEdit;
42    IniPropStorage: TIniPropStorage;
43    Label1: TLabel;
44    Label2: TLabel;
45    Label3: TLabel;
46    Label4: TLabel;
47    Label5: TLabel;
48    LabelTime: TLabel;
49    MemoDebug: TMemo;
50    PageControl: TPageControl;
51    SaveDialog1: TSaveDialog;
52    Serial: TSdpoSerial;
53    ShapeSerialState: TShape;
54    StatusBar: TStatusBar;
55    StringGrid1: TStringGrid;
56    StringGridCalib: TStringGrid;
57    TabDebug: TTabSheet;
58    TabConfig: TTabSheet;
59    Timer1: TTimer;
60    procedure BCloseSerialClick(Sender: TObject);
61    procedure BOpenSerialClick(Sender: TObject);
62    procedure BSendRawClick(Sender: TObject);
63    procedure BtSaveDataClick(Sender: TObject);
64    procedure BtSetTimerClick(Sender: TObject);
65
66    procedure BtSaveImgClick(Sender: TObject);
67    procedure Button1Click(Sender: TObject);
68    procedure ButtonCalibClick(Sender: TObject);
69    procedure ButtonSaveImageClick(Sender: TObject);
70    procedure ButtonSaveLogClick(Sender: TObject);
71    procedure CBgoaqcChange(Sender: TObject);
72    procedure CBgoaqcClick(Sender: TObject);
73    procedure CheckBoxSendUDPChange(Sender: TObject);
74    procedure EditSendRawKeyDown(Sender: TObject; var Key: Word;
75      Shift: TShiftState);
```

```
76    procedure FormClose(Sender: TObject; var CloseAction: TCloseAction);
77    procedure FormCreate(Sender: TObject);
78    procedure FormDestroy(Sender: TObject);
79    procedure FormShow(Sender: TObject);
80    procedure SerialRxData(Sender: TObject);
81    procedure Timer1Timer(Sender: TObject);
82    procedure calib();
83  private
84    procedure Debug(msg: string);
85    procedure processFrame(channel: char; value: integer; source:
     integer);
86    procedure processText(s: string);
87    procedure SaveLog;
88
89    { private declarations }
90  public
91    ResetCount: integer;
92    SerialChannels: TChannels;
93    A0, A1, A2, A3, A4, A5, t: integer;
94    V,m,b,q_filt: array[0..5] of real;    // Valores; m e b eq reta
     calibraÃğÃčo
95
96    x,y: real;
97    AqPeriod: integer;
98    sample: integer;
99    totalsamples: integer;
100   totalAqTime: integer;
101   actualaqtime: real;
102
103   procedure ShowSerialState;
104
105   procedure SendMessage(c: char; val: integer);
106   procedure SendMessageFloat(c: char; val: single);
107   procedure sendbyUDP();
108 end;
```

```
109
110 var
111   FMain: TFMain;
112
113
114 implementation
115
116 {$R *.lfm}
117
118 { TFMain }
119
120
121 procedure TFMain.ShowSerialState;
122 begin
123   if Serial.Active then begin
124     ShapeSerialState.Brush.Color := clGreen;
125   end else begin
126     ShapeSerialState.Brush.Color := clRed;
127   end;
128 end;
129
130 procedure TFMain.SendMessage(c: char; val: integer);
131 begin
132   Serial.WriteData(c + IntToHex(Val, 8));
133 end;
134
135 procedure TFMain.SendMessageFloat(c: char; val: single);
136 begin
137   Serial.WriteData(c + IntToHex(PDWord(@val)^, 8));
138 end;
139
140
141 procedure TFMain.BOpenSerialClick(Sender: TObject);
142 begin
143   Serial.Device := EditSerialName.Text;
```

```
144   Serial.Open;
145   ShowSerialState();
146   Timer1.Enabled:=true;
147 end;
148
149
150 procedure TFMain.BSendRawClick(Sender: TObject);
151 begin
152   //Serial.WriteData(EditSendRaw.Text);
153   Serial.WriteData('I00000000');
154 end;
155
156 procedure TFMain.BtSaveDataClick(Sender: TObject);
157 begin
158
159 end;
160
161 procedure TFMain.BtSetTimerClick(Sender: TObject);
162 begin
163   AqPeriod := StrToIntDef(EditSamplePeriod.Text,1000);
164   if AqPeriod < MINPERIOD then AqPeriod:=MINPERIOD;
165   Timer1.Interval:=AqPeriod;
166   EditSamplePeriod.Text:=inttostr(AqPeriod);
167
168   totalAqTime:= StrToIntDef(EditAqTime.Text,60);
169   EditAqTime.Text := IntToStr(totalAqTime);
170 end;
171
172
173 procedure TFMain.BtSaveImgClick(Sender: TObject);
174 begin
175
176
177 end;
178
```

```
179 procedure TFMain.Button1Click(Sender: TObject);
180 begin
181   Serial.WriteData('P00000080');
182 //  LUDPComponent1.Listen(9990);
183 end;
184
185
186 procedure TFMain.SaveLog;
187 var i: integer;
188     SL: TStringList;
189     FormatSettings: TFormatSettings;
190 begin
191   FormatSettings := DefaultFormatSettings;
192   FormatSettings.TimeSeparator := '_';
193   SL := TStringList.Create;
194   try
195     SL.Add('i; t; A0; A1; A2; A3; A4; A5');
196     for i := 0 to  CLA0.Count - 1 do begin
197       SL.Add(format('%d; %e; %e; %e; %e; %e; %e; %e',
198                     [i, CLA0.XValue[i],
199                      CLA0.YValue[i],
200                      CLA1.YValue[i],
201                      CLA2.YValue[i],
202                      CLA3.YValue[i],
203                      CLA4.YValue[i],
204                      CLA5.YValue[i]
205                     ]));
206     end;
207     SL.SaveToFile('DATA\' + Editcsvfilename.text + '.csv');
208 //SL.SaveToFile('teste111.txt');
209     SL.Clear;
210   finally
211     SL.Free;
212   end;
213 end;
```

```
214
215
216
217 procedure TFMain.ButtonCalibClick(Sender: TObject);
218 begin
219  calib;
220 end;
221
222
223
224 procedure TFMain.ButtonSaveImageClick(Sender: TObject);
225 begin
226    if SaveDialog1.Execute then Chart1.SaveToBitmapFile(SaveDialog1.
    FileName);
227 end;
228
229 procedure TFMain.ButtonSaveLogClick(Sender: TObject);
230 begin
231   SaveLog();
232 end;
233
234 procedure TFMain.CBgoaqcChange(Sender: TObject);
235 begin
236   if CBgoaqc.Checked then begin
237     CLA0.Clear;
238     CLA1.Clear;
239     CLA2.Clear;
240     CLA3.Clear;
241     CLA4.Clear;
242     CLA5.Clear;
243     actualaqtime := 0;
244   end
245   else begin
246    SaveLog();
247   end;
```

```
248 end;
249
250 procedure TFMain.CBgoaqcClick(Sender: TObject);
251 begin
252   sample := 0;
253 end;
254
255 procedure TFMain.CheckBoxSendUDPChange(Sender: TObject);
256 begin
257  // LUDPComponent1.Port:=StrToInt(EdListenPort.Text);
258 //  LUDPComponent1.Listen(StrToInt(EdListenPort.Text));
259 end;
260
261 procedure TFMain.EditSendRawKeyDown(Sender: TObject; var Key: Word;
262   Shift: TShiftState);
263 begin
264   if key = VK_RETURN then begin
265     BSendRaw.Click();
266   end;
267 end;
268
269 procedure TFMain.FormClose(Sender: TObject; var CloseAction:
      TCloseAction);
270 begin
271   Serial.WriteData('P00000000');
272   Serial.Close;
273   StringGridCalib.SaveToFile('calibs.ini');
274 end;
275
276 procedure TFMain.FormCreate(Sender: TObject);
277 begin
278   // Lazarus catches WM_SETTINGCHANGE and calls Application.
      IntfSettingChange
279   // which calls GetFormatSettings in SysUtils
```

```
280   // It can be switched off by setting Application.UpdateFormatSettings
       := False;
281   {$IFDEF WINDOWS}
282   Application.UpdateFormatSettings := false;
283   {$ENDIF}
284   DefaultFormatSettings.DecimalSeparator := '.';
285   StringGridCalib.LoadFromFile('calibs.ini');
286   calib;
287   SerialChannels := TChannels.Create(@processFrame, @processText);
288 end;
289
290 procedure TFMain.FormShow(Sender: TObject);
291 begin
292   ShowSerialState();
293   BtSetTimerClick(Sender);
294   if CBAutoOpen.Checked then BOpenSerial.Click();
295   AqPeriod := StrToIntDef(EditSamplePeriod.Text,1000);
296   Timer1.Interval:=AqPeriod;
297 end;
298
299
300 procedure TFMain.FormDestroy(Sender: TObject);
301 begin
302   SerialChannels.Free;
303 end;
304
305 procedure TFMain.calib();
306   var i: integer;
307 begin
308    //StringGridCalib.Cells[0,1] := 'x';
309   for i :=0 to 5 do begin
310     m[i] := strtofloatdef(StringGridCalib.Cells[1,i+1],0);
311     b[i] := strtofloatdef(StringGridCalib.Cells[2,i+1],0);
312     q_filt[i] := strtofloatdef(StringGridCalib.Cells[3,i+1],0);;
313   end;
```

```
314 end;
315
316 procedure TFMain.Debug(msg: string);
317 begin
318   MemoDebug.Text := MemoDebug.Text + msg;
319   MemoDebug.SelStart:=Length(MemoDebug.Text);
320   while MemoDebug.Lines.Count > 50 do begin
321     MemoDebug.Lines.Delete(0);
322   end;
323 end;
324
325 procedure TFMain.SerialRxData(Sender: TObject);
326 var s: string;
327 begin
328   s := Serial.ReadData;
329   if s = '' then exit;
330
331   SerialChannels.ReceiveData(s);
332
333   //MemoDebug.VertScrollBar.Position := MemoDebug.VertScrollBar.Range;
334   if CBRawDebug.Checked then begin
335     debug(s);
336   end;
337 end;
338
339 procedure TFMain.Timer1Timer(Sender: TObject);
340 begin
341   actualaqtime := actualaqtime + AqPeriod / 1000;
342   if Serial.Active then begin
343     if CBgoaqc.Checked then begin
344       LabelTime.Caption:='Aq time (s)='+FloatToStr(actualaqtime);
345       CLA0.AddXY(actualaqtime,V[0]);
346       CLA1.AddXY(actualaqtime,V[1]);
347       CLA2.AddXY(actualaqtime,V[2]);
348       CLA3.AddXY(actualaqtime,V[3]);
```

```
349       //CLA4.AddXY(actualaqtime,V[4]);
350       //CLA5.AddXY(actualaqtime,V[5]);
351     end;
352     if ((actualaqtime > totalAqTime) and CBgoaqc.Checked) then begin
353       CBgoaqc.Checked:=false;
354       SaveLog();
355     end;
356   end;
357 end;
358
359
360
361
362
363
364 procedure TFMain.sendbyUDP();
365 var
366     sbuf: string;
367 begin
368   sbuf:=StringGrid1.Cells[1,1]+';'+StringGrid1.Cells[1,2]+';'+
     StringGrid1.Cells[1,3]+';'+StringGrid1.Cells[1,4]+';'+StringGrid1.
     Cells[1,5]+';'+StringGrid1.Cells[1,6]+';'+StringGrid1.Cells[1,7];
369 end;
370
371 procedure TFMain.BCloseSerialClick(Sender: TObject);
372 begin
373   Serial.Close;
374   ShowSerialState();
375   Timer1.Enabled:=False;
376 end;
377
378
379 procedure TFMain.processText(s: string);
380 begin
381   debug(s);
```

```
382 end;
383
384 procedure TFMain.processFrame(channel: char; value: integer; source:
      integer);
385 var i: integer;
386 begin
387   //MemoDebug.Text := MemoDebug.Text + channel;
388   if channel = 'X' then begin
389     // if the arduino was reset ...
390
391   end else if channel = 'M' then begin
392     A0 := value;
393     V[0] := (m[0] * A0 + b[0]) * (1 - q_filt[0]) + (q_filt[0] * V[0]) ;
394     StringGrid1.Cells[1,1] := floattostr(V[0]);
395     StringGridCalib.Cells[5,1]:= floattostr(V[0]);
396     StringGridCalib.Cells[4,1]:= inttostr(A0);
397
398
399   end else if channel = 'N' then begin
400     A1 := value;
401     V[1] := (m[1] * A1 + b[1]) * (1 - q_filt[1]) + (q_filt[1] * V[1]);
402     StringGrid1.Cells[1,2] := floattostr(V[1]);
403     StringGridCalib.Cells[5,2]:= floattostr(V[1]);
404     StringGridCalib.Cells[4,2]:= inttostr(A1);
405   end else if channel = 'O' then begin
406     A2 := value;
407     V[2] := (m[2] * A2 + b[2]) * (1 - q_filt[2]) + (q_filt[2] * V[2]);
408     StringGrid1.Cells[1,3] := floattostr(V[2]);
409     StringGridCalib.Cells[5,3]:= floattostr(V[2]);
410     StringGridCalib.Cells[4,3]:= inttostr(A2);
411   end else if channel = 'P' then begin
412     A3 := value;
413     V[3] := (m[3] * A3 + b[3]) * (1 - q_filt[3]) + (q_filt[3] * V[3]);
414     StringGrid1.Cells[1,4] := floattostr(V[3]);
415     StringGridCalib.Cells[5,4]:= floattostr(V[3]);
```

```
416     StringGridCalib.Cells[4,4]:= inttostr(A3);
417   end else if channel = 'Q' then begin
418     A4 := value;
419     V[4] := (m[4] * A4 + b[4]) * (1 - q_filt[4]) + (q_filt[4] * V[4]);
420     StringGrid1.Cells[1,5] := floattostr(V[4]);
421     StringGridCalib.Cells[5,5]:= floattostr(V[4]);
422     StringGridCalib.Cells[4,5]:= inttostr(A4);
423   end else if channel = 'R' then begin
424     A5 := value;
425     V[5] := (m[5] * A5 + b[5]) * (1 - q_filt[5]) + (q_filt[5] * V[5]);
426     StringGrid1.Cells[1,6] := floattostr(V[5]);
427     StringGridCalib.Cells[5,6]:= floattostr(V[5]);
428     StringGridCalib.Cells[4,6]:= inttostr(A5);
429   end else if channel = 'S' then begin
430     t := value;
431     StringGrid1.Cells[1,7] := IntToStr(t);
432   end;
433 end;
434
435
436
437 end.
```