

Instituto Tecnológico de Costa Rica
Área Académica de Ingeniería Mecatrónica



**Implementation of a low-cost IoT system for a Hermle
UWF900E CNC milling machine**

Informe de Proyecto de Graduación para optar por el título de
Ingeniero en Mecatrónica con el grado académico de Licenciatura

Daniel Berrocal Campos

Cartago, 2 de abril de 2018

Declaro que el presente Proyecto de Graduación ha sido realizado enteramente por mi persona, utilizando y aplicando literatura referente al tema e introduciendo conocimientos propios.

En los casos en que he utilizado bibliografía he procedido a indicar las fuentes mediante las respectivas citas bibliográficas. En consecuencia, asumo la responsabilidad total por el trabajo de graduación realizado y por el contenido del correspondiente informe final.

Daniel Berrocal Campos

Cartago, 2 de abril de 2018

Céd: 4-0226-0718

Instituto Tecnológico de Costa Rica
Área Académica de Ingeniería Mecatrónica
Proyecto de Graduación
Tribunal Evaluador

Proyecto de Graduación defendido ante el presente Tribunal Evaluador como requisito para optar por el título de Ingeniero en Mecatrónica con el grado académico de Licenciatura, del Instituto Tecnológico de Costa Rica.

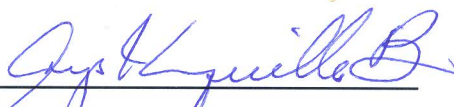
Miembros del Tribunal



Msc. Yeiner Arias Esquivel
Profesor Lector



Msc. Carlos Adrián Salazar García
Profesor Lector



Ing. Arys Carrasquilla Batista, M.C.
Profesora Asesora

Los miembros de este Tribunal dan fe de que el presente trabajo de graduación ha sido aprobado y cumple con las normas establecidas por el Área Académica de Ingeniería Mecatrónica.

Cartago, 2 de abril de 2018

Abstract

This inform summarizes the work executed for designing, building and testing a custom, low-cost monitoring system for a CNC milling center model Hermle UWF900E. The structure proposed follows the general topology of an IoT application, with the intention of making access and data collection easier. The variables monitored were chosen as a combination of basic working conditions, safety parameters and environmental factors that could affect the operation. The stages involved in the project development included theoretical research, to find the adequate protocols and components to use; electrical circuit design, including sensor calibration and controller programming; and Internet communication deployment, involving both a web-based database storage and web-page working as user interface. Each stage included a series of tests to prove individual performance of the parts involved. After installing the whole system, real operative data was recorded during a period of 2 weeks, allowing to both asses the efficiency of the IoT system and to discover some relationships in the observed magnitudes. A meaningful correlation between spindle speed and high-frequency vibrations was found, with Spearman coefficients over 0.7, proving the ability of the system installed to generate enough data for functional analysis of the machine.

Keywords: CNC machines, condition monitoring, Fast Fourier Transform, Internet of Things, Platform as a Service, sensor conditioning, serialization protocols

Dedico este trabajo a mis padres, con todo cariño

Agradecimientos

En primer lugar, quisiera agradecer a mi familia, cuyo apoyo incondicional durante todos los años de estudio, y especialmente durante el período en el que trabajé en este proyecto, fue indispensable para seguir hacia adelante. Extiendo igualmente mis más sinceros agradecimientos al personal del Departamento de Ingeniería Mecánica del DHBW sede Karlsruhe, especialmente a Dr. Clemens Reitze, Prof. Mathias Metzner y Prof. Gunter Schafer, quienes no solo me brindaron la oportunidad de usar las instalaciones de la institución para implementar mis ideas, también me brindaron valiosas recomendaciones sobre diversos aspectos de las mismas. A los miembros de mi tribunal evaluador en el ITCR, Ing. Arys Carrasquilla, Msc. Carlos Salazar y Msc. Yeiner Arias les agradezco sus importantes aportes sobre el trabajo realizado, que me permitió mejorar en gran medida este reporte y afinar muchos detalles que de otra manera no hubiera considerado. Finalmente, quisiera agradecer a todos los profesores y compañeros con los que compartí durante mis años de carrera, ya que sin la experiencia que adquirí junto a ellos me hubiera sido imposible siquiera encarar el reto que supuso el proyecto.

Daniel Berrocal Campos

Cartago, 2 de abril de 2018

Contents

List of Figures	v
List of Tables	vii
List of symbols and abbreviations	ix
1 Introduction	1
1.1 History and perspectives of the DHBW Karlsruhe	1
1.2 The challenge of monitoring mechanical equipment	2
1.3 Brief overview of the monitoring system design	3
1.4 Objectives and document structure	4
2 Theoretical references	7
2.1 CNC machines: characteristics and requirements	7
2.1.1 General definition	7
2.1.2 Maintenance of CNC machines	7
2.1.3 Examples of modern maintenance	9
2.1.4 Hermle UWF900E parameters	9
2.2 Internet of Things and cloud-hosted services	10
2.2.1 What is IoT?	10
2.2.2 Cloud hosting and Platform as a Service (PaaS)	12
2.2.3 Databases: SQL vs NoSQL	12
2.2.4 Basics of web design	13
2.3 Communication protocols for fast processing	14
2.3.1 Serialization protocols	14
2.3.2 Google Protocol Buffers	15
2.4 Filtering and data analysis on the digital domain	17
2.4.1 Processing filters	17
2.4.2 Timing and digitization errors	19
2.4.3 Frequency analysis: Fast Fourier Transform (FFT)	19
3 IoT Monitoring System for CNC: Sensors and other electronic components	23
3.1 Summary of the design process	23
3.2 Component selection	24

3.2.1	Variables to be monitored	24
3.2.2	Sensor selection	24
3.2.3	Microcontroller and other ICs	26
3.3	Microcontroller programming	29
3.3.1	General structure and timing	29
3.3.2	Serial communications	30
3.3.3	Measurement processing	31
3.4	Physical installation	32
3.4.1	Electronic circuits	32
3.4.2	Installation on the CNC machine	34
4	IoT Monitoring System for CNC: Internet communication	35
4.1	Summary of the design process	35
4.2	Software selection	36
4.2.1	Database	36
4.2.2	Server side management (PaaS)	37
4.2.3	Programming language and additional libraries	38
4.3	Processing services on the SBC	39
4.3.1	Sensor data parsing and database updates	39
4.3.2	Machine status and frequency analysis	41
4.3.3	Watchdog	42
4.4	Web based services	43
4.4.1	Web page main layout	43
4.4.2	Graphics	45
4.4.3	Security options	46
5	Results and analysis	47
5.1	Sensor calibrations	47
5.2	Bluetooth and Protocol Buffers	53
5.3	Web page with made-up data	56
5.4	Web page with real-time data	58
5.5	Analysis of operative data	66
5.6	Watchdog tests	69
5.7	Costs involved in the project	70
6	Conclusions and future work	73
	Bibliography	77
A	Example of the Cooley-Tukey FFT algorithm	83
B	Installation and user guide	85
B.1	Quick Setup	85
B.2	Modifying existing software	86

B.2.1	Installing NodeJS and libraries	86
B.2.2	Modifying Raspberry Pi services	87
B.2.3	Linking a new Bluetooth module	88
B.2.4	Creating new server repository	89
B.2.5	Deploying changes to the webpage	89
B.2.6	Administering database users	90
B.2.7	Programming the MCU	90
B.2.8	Changing the Protobuf message	92
C	Electronic circuit in detail	93
	Index	95

List of Figures

1.1	CNC milling machine Hermle UWF900E. Source: [4]	2
1.2	General structure implemented in the project. Source: Own elaboration	5
2.1	Failure tree elaborated for a 3-axis CNC milling machine. Source: [8]	8
2.2	Example of the topology for an IoT system. Source: [16]	11
2.3	Comparison between serialization protocols. Source: [28]	16
2.4	File size for different serialization protocols. Source: [26]	17
3.1	PCB design. Source: Own elaboration	34
4.1	Login page for user interface. Source: Own elaboration	44
4.2	Home page for user interface. Source: Own elaboration	44
4.3	Error message for user interface. Source: Own elaboration	45
4.4	Example of graph view for user interface. Source: Own elaboration	46
5.1	Calibration curve for pressure sensor. Source: Own elaboration	51
5.2	Calibration curve for microphone. Source: Own elaboration	52
5.3	Time required for sending a full array-type message. Source: Own elaboration	55
5.4	Time required for a single ADC conversion. Source: Own elaboration	56
5.5	Results obtained for made-up data: “Acceleration”. Source: Own elaboration	57
5.6	Results obtained for made-up data: “Vibrations”. Source: Own elaboration	57
5.7	Results obtained for made-up data: “Temperature”. Source: Own elaboration	58
5.8	Results obtained for real data on 1st try: Noise. Source: Own elaboration	59
5.9	Results obtained for real data on 1st try: Acceleration. Source: Own elaboration	59
5.10	Results obtained for real data on 1st try: Spindle speed. Source: Own elaboration	60
5.11	Operational data: Spindle-speed. Source: Own elaboration	61
5.12	Operational data: Pressure. Source: Own elaboration	61
5.13	Operational data: Temperature. Source: Own elaboration	62
5.14	Operational data: Vibrations. Source: Own elaboration	63
5.15	Operational data: Vibrations frequency. Source: Own elaboration	63
5.16	Operational data: Acceleration. Source: Own elaboration	64
5.17	Operational data: Acoustics. Source: Own elaboration	65

5.18	Operational data: Acoustics frequency. Source: Own elaboration	65
5.19	Correlation analysis for vibrations: Pearson. Source: Own elaboration . . .	67
5.20	Correlation analysis for vibrations: Spearman. Source: Own elaboration . .	67
5.21	Correlation analysis for temperatures: Pearson. Source: Own elaboration .	68
5.22	Correlation analysis for temperatures: Spearman. Source: Own elaboration	68
5.23	Correlation analysis for vibrations frequency: Pearson. Source: Own elab- oration	68
5.24	Correlation analysis for vibrations: Spearman. Source: Own elaboration . .	69
5.25	Message for setup alarm. Source: Own elaboration	69
5.26	Message for repeated alarm. Source: Own elaboration	70
5.27	Example of wrongly activated alarm. Source: Own elaboration	70
B.1	Circuit for Atmega 328 programming. Source: [55]	91
C.1	Main electronic circuit	94

List of Tables

2.1	CNC Hermle UWF900E Specifications	10
3.1	Sensors used in the project	27
3.2	Comparison between microcontroller models	28
3.3	Other integrated circuits used in the project	29
3.4	Current consumption in the system circuits	33
4.1	Comparison between different PaaS providers	38
5.1	Calibration tests for light sensor	48
5.2	Measurements for linear accelerometer calibration	48
5.3	Results for lineal accelerometer after processing	49
5.4	Measurements for vibrations sensor calibration	50
5.5	Test results for vibration sensor after processing	50
5.6	Measurements in pressure sensor calibration	51
5.7	Test results for pressure sensor after processing	51
5.8	Comparison of microphone performance	53
5.9	Results for frequency analysis tests	54
5.10	Test results for communication with Bluetooth and Protobufs	54
5.11	Costs of the project by category	72

List of symbols and abbreviations

Abbreviations

BSON	Binary JavaScript Object Notation
CM	Condition Monitoring
CNC	Computer Numerical Control
COTS	Commercially off-the-shelf
DFT	Discrete Fourier Transform
DHBW	Duale Hochschule Baden-Württemberg (Baden-Württemberg Cooperative State University)
DSP	Digital Signal Processing
FFT	Fast Fourier Transform
GUI	Graphic User Interface
HTML	Hyper Text Markup Language
IaaS	Infrastructure as a Service
IC	Integrated Circuit
IoT	Internet of Things
IP	Internet Protocol
IT	Information Technologies
JSON	JavaScript Object Notation
LAN	Local Area Network
MCU	Microcontroller Unit
MEMS	Micro Electro-Mechanical System
PaaS	Platform as a Service
PCB	Printed Circuit Board
RMS	Root Mean Square
RTC	Real Time Clock
SaaS	Software as a Service
SBC	Single-Board Computer
SPL	Sound Pressure Level
SQL	Structured Query Language
URI	Uniform Resource Identifier
WSN	Wireless Sensor Network

General notation

$a[n]$	n -th element of the a series
\mathbb{C}	Set of the complex numbers

e	Euler's number, base of the natural logarithm
$\text{Im}(z)$	Imaginary part of complex number z
j	$j = \sqrt{-1}$
$\text{Re}(z)$	Real part of complex number z
$\sum_{i=n}^m a[i]$	Summation of the elements of the sequence a , starting with the n-th element and ending with the m-th element

Chapter 1

Introduction

1.1 History and perspectives of the DHBW Karlsruhe

The State of Baden-Württemberg is widely recognized as one of the most industrial-oriented zones in Germany, and it is the home of several important companies with historical impact in the country's economic development, such as Daimler, Bosch, and IBM Deutschland. The demand of qualified workers has boosted a privileged culture for the growth of mechanical and electrical engineering, and high-precision metal machining. An even more interesting fact is that most of the business located in this region are classified as medium or small-sized, so in many cases the focus of the local I&D centers is shifted towards these markets [1]. The economic context was a key factor in the creation of the Baden-Württemberg Cooperative State University (Duale-Hochschule Baden-Württemberg, DHBW). Since 1974, this institution has been a pioneer in the application of the dual-education concept, integrating formal academic studies with practical work directly in enterprises. For this reason, the different campus need to adjust their academic plans to the industries real-world inquiries, and therefore require facilities that are up to the latest technologies [2].

The Karlsruhe campus of DHBW has an important catalog of engineering careers, including departments for Electronic, Mechanical, Computer and Mechatronics Engineering [3]. Each department has industrial level equipment which can be used by students in lessons and practices. The Mechanical Engineering department, for example, has turbines, ovens and different milling machines and lathes. One of the most important devices, in terms functional complexity, is the Hermle CNC milling machine, model UWF900E, which can be found in the figure 1.1.

The maintenance of this and the rest of the machines is a very important considering the frequency of their usage, which expose them to a more accelerated degradation process. Bearing in mind the whole population of students is about 3000 people [3], to monitor the use of every single piece of institutional equipment becomes a total challenge. That

is why some of the professors involved with the machines have begun their own research towards the development of tools that ease this task and allow them to comprehend in a better way how is everything working in their own laboratories.



Figure 1.1: CNC milling machine Hermle UWF900E. Source: [4]

1.2 The challenge of monitoring mechanical equipment

The Mechanical Engineering Department of the DHBW Karlsruhe has many different machines and equipment. The professors would like to know about the work conditions these machines are really facing, but obviously, they cannot be watching over them the whole day. Therefore, there is a need to monitor these devices remotely, showing the results in such a manner any member of the faculty could see them and understand them easily. Moreover, to maximize the usefulness of such monitoring system, it is convenient to allow simultaneous access to all the personnel interested in the situation, so they can complete any further analysis over the collected data.

Taking into consideration that installing measurement systems over the whole department inventory would require a long time, it was a better option to test the idea only in one of the machines. Then, analyzing its own set of features and the overall importance it has for the university, the Hermle UWF900E milling machine was chosen as starting point. CNC machines are highly complex electromechanical devices, so to register their functional state, several physical variables need to be analyzed. Each of the variables lies on a specific range of values that can be considered good enough for the CNC regular operation. Also, in many cases, visualizing relationships between two or more variables is necessary to establish the real state of the whole group. Although most of the times the internal controllers of the CNC manage these datasets, access by user side is usually protected or directly blocked.

The personnel at Mechanical Engineering Department has found some critical variables required to understand the behavior of their milling machine, which include: hydraulic

pressure and temperature for the cooling system; open/closed state of the access door for safety; rotational, lineal speeds and vibrations in the working area for tool positioning; and current consumption for electrical power. The control section relies entirely on the internal computers and software, which does not get degraded with usage, so there is no utility in keeping them under constant vigilance. There is space for monitoring any other operative conditions that could complement the functional profile of the device.

In synthesis, the problem this project solved was the need to verify that the machines of DHBW Karlsruhe Mechanical Engineering Department are working in optimal conditions.

1.3 Brief overview of the monitoring system design

To keep a complete, real-time, remote access to all the information registered, besides a group of sensors and electronic controllers, a full working communication system is necessary. Therefore, the plan for this project was to focus on a system based on Internet of Things (IoT). In simple words, the final design integrates web-based communications and data storage with mechanical and electronic devices.

Right now, there are fully developed options available in the market for IoT applications in factories and other industrial environments. Nevertheless, implementing one of these solutions is usually expensive. For academic environments and low-scale enterprises is better to design tailored systems, using open-source elements and COTS sensors, which are cheaper and have more possibilities for the user. Also, this approach is optimal for scaling up solutions to other CNCs or different kind of machines, as intended by the Mechanical Engineering Department. Thus, the project was developed within this vision. Moreover, one of the first considerations made was that just showing raw data on a screen makes the analysis of usage tendencies and performance more difficult for the people interested in it. Hence, the solution developed includes a simple GUI, allowing the visualization of time-series registers in different graphs.

The general working scheme used in this project was:

- **Theoretical research about the machine and possible materials to use:** To select which characteristics are important enough to be monitored, which of them can be measured without too much intervention in the CNC, and what kind of sensors could get those measurements, a brief review of technical documents and discussions with the professors of the DHBW were conducted. Also, it was necessary to choose a micro- controller, a SBC and all the software packages for communication and GUI, always collecting the right information before getting a conclusive choice.
- **Selection and calibration of electronic components:** Once all the sensors were bought, it was necessary to verify the output of each one under controlled stimulus. The same tests allowed to evaluate accuracy and precision. In the end, it was possible to generate a set of experimental curves, which were the basis for further measurement pre-processing programming.

- **Micro-controller programming:** It was required to establish a connection between all the sensors and the controller selected, to process the measurements and set an appropriate flow of data. This include the use of filters and the creation of timing algorithms that assured no interference between components. After that, a new communication section was tested, this time to send data to the SBC, and the heaviest parts of data processing, which could not be implemented in the controller, were developed in this other device.
- **Cloud database programming:** A full working data structure, able to manage incoming data from all the sensors, was designed and implemented online. Then, the required queries tested the functionality of the database. Finally, permissions and access keys were added. The whole database system was connected to a remote server, via a PaaS provider.
- **GUI programming:** A specific format for the downloaded data in each query was established, according to common use protocols. Then, over the cloud-based server platform, a web page including graphs for each variable monitored was created. The design includes the ability to refresh information, so every time the page is loaded, the complete set of data is retrieved from the database, including the newest uploads.
- **Results evaluation:** When the whole system was ready, information was collected over a period of two weeks, expecting to detect any tendency in the CNC operation and giving special attention to variables correlation. A statistical analysis software was used for this purpose.

Figure 1.2 shows a high level representation of the whole installed system. Notice the wired connections between the sensors and the micro-controller work over a specific communication protocol, which would be properly explained in further chapters.

1.4 Objectives and document structure

The main objective pursued with this project was to develop a remote monitoring system for operational proficiency of a CNC milling machine, applying IoT concepts and low-cost components, in way that allows easy access and usage to the people in charge of the equipment. To clarify the way to achieve this general goal, the reach of the work done was divided in the following specific objectives:

1. To design the external measurement circuit required to supervise the machine, considering the critical variables for its operation.
2. To implement the necessary algorithms for communicating the electronic micro-controllers with a cloud-stored database, using informatics safety elements
3. To design a graphical interface able to show all the measurement results in a clear way, over any computer provided with access to the database

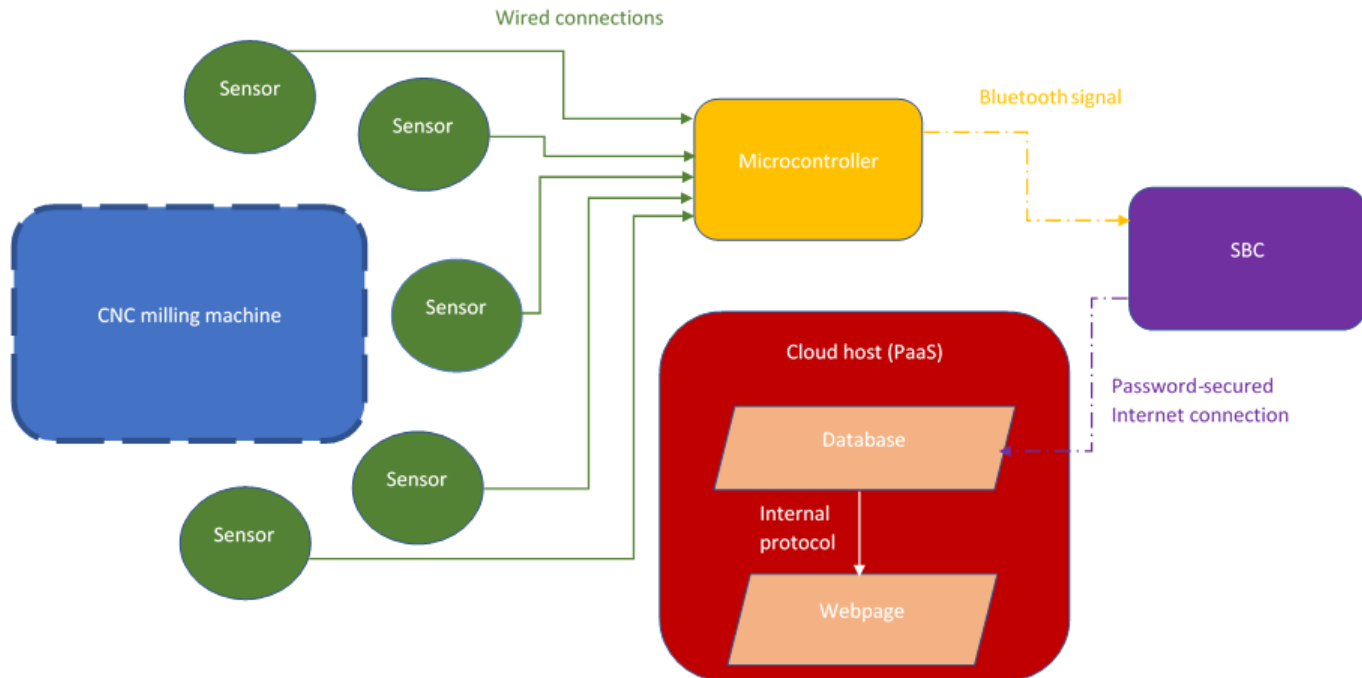


Figure 1.2: General structure implemented in the project. Source: Own elaboration

4. To evaluate the performance of the milling machine analyzing tendencies over a short period of time (1-2 weeks) through the data registered by the IoT system

This inform explains how all these objectives were fulfilled in a sequential order. The next chapter covers the theoretical basis applied in the project, and some references consulted as a source of ideas for the structure designed. In chapter 3, the sensor electronics and programming of such design are explained in detail, commenting all the decisions made for each section of the system, the reasons behind them and corresponding data to back them up when it was considered necessary. Correspondingly, in chapter 4 the same is done with the Internet communication segment. Chapter 5 shows a brief collection of the experimental results achieved in the end of the implementation, and some analysis done over this information. Finally, on chapter 6, a list of the main conclusions obtained with this work and some recommendations for future research in the area is presented. Some related data, as demonstrations for algorithms and user guides, is included in the appendix section at the end of the document.

Chapter 2

Theoretical references

2.1 CNC machines: characteristics and requirements

2.1.1 General definition

Traditional machining processes are characterized by the presence of an operator who leads the motion of the working area in the machine to create the desired shape on a piece [5]. This method has several disadvantages: the output rate is heavily limited by the capacity of the operators; the accuracy is very dependent on the level of skill the person has and the human factor affects the repeatability of dimensional accuracy [5]. Moreover, most of the pieces produced nowadays have requirements in shape and surface accuracy so extensive they cannot be reached with only one-dimensional movements [6], and modern industry privileges the concepts of flexible and tailored design, which imply constant change in the used set of tools [7]. Therefore, it is impossible for single operator to do the whole job accurately by himself [5]. CNC machining centers solve the problem.

In simple words, CNC (Computer Numerical Control) refers to the method of directing the operation of a machining equipment introducing directly the wanted numeric parameters via a computerized environment, instead of guiding the movements manually [5]. Like other manufacturing machines, these centers always count with a *table* (which holds the machined pieces) and a *spindle* (which holds the tools). According to spindle position and operation, the three main types of machining centers are vertical mills, horizontal mills and turning (or lathing) centers, which include rotation in the table too. [6].

2.1.2 Maintenance of CNC machines

The main disadvantages CNC machines have when compared to traditional ones are higher initial costs; personnel training with more specific requirements, which involves knowledge not only on common machining, but programming and design as well [5] [7]); and elevated costs of long term maintenance. The last one deserves a special attention,

as its effects span over a longer period. Some of the augmented expenses come from the greater complexity of such devices, which integrate electrical, mechanical, pneumatic, and hydraulic sections [7] through coordinated movements in 3 or more axes. Also, the software controls introduce another layer of possibilities whenever a failure occurs [5]. An example of a complete, traditional maintenance plan for CNC centers is presented in [8]. It divides the possible failures on the machine in the following areas: cooling subsystem, positioning subsystem, computer control subsystem, electrical power subsystem and safety subsystem. Figure 2.1 shows this categorization, alongside the most common reasons for failure found by the work. The consequences lousy or badly planned inspections bring

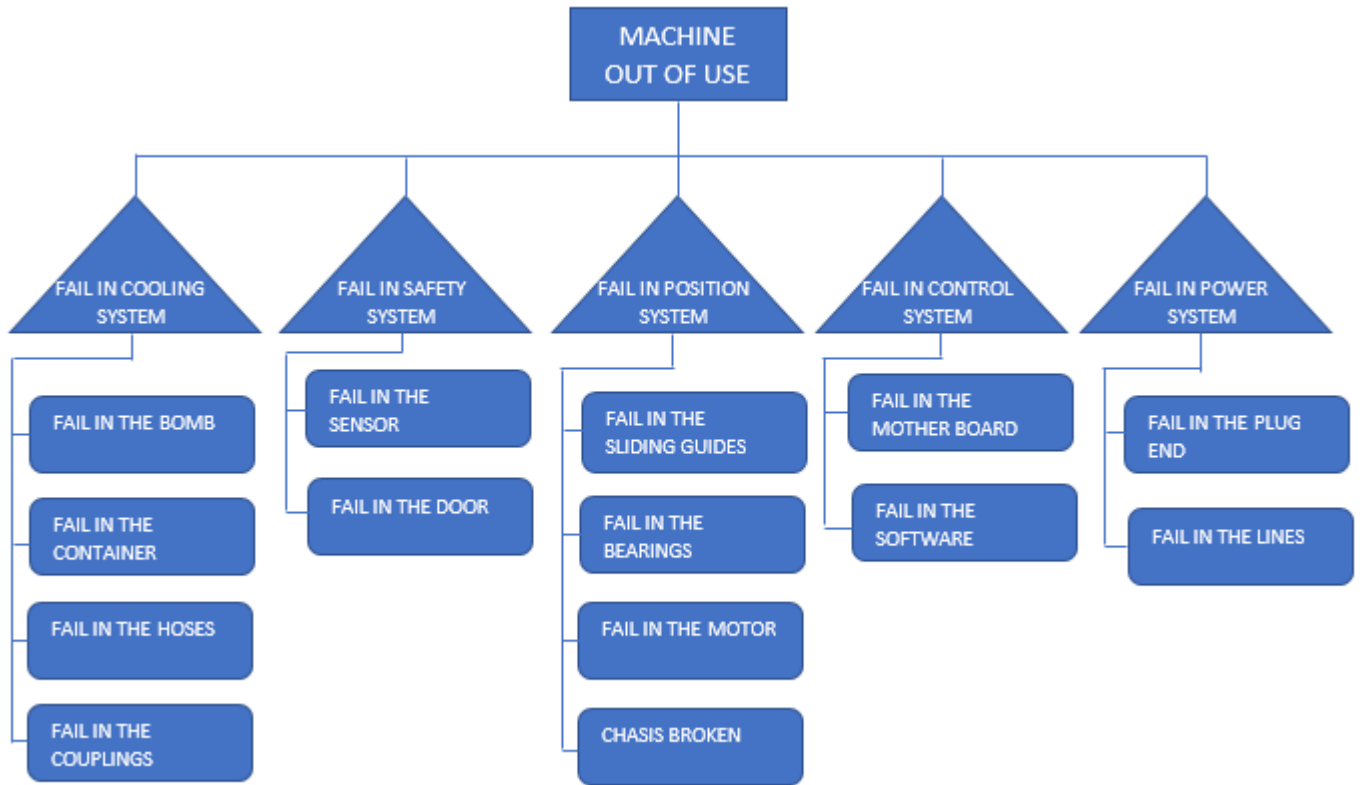


Figure 2.1: Failure tree elaborated for a 3-axis CNC milling machine. Source: [8]

over the machine can go from a detrimental in the quality of pieces produced to a total operative damage. One common way machining devices show internal unbalance is by excessive vibrations. Wrong cutting parameters or not optimal selected tools are usual sources for that problem. [6]. Another case is thermal distortion on tools and piece, caused by several different situations, such as insufficient bearing lubrication or wrong cooler fluid application while cutting [6]. Both conditions show the relevance of extending CNC maintenance to more modern techniques, involving a close understanding of the operative conditions.

2.1.3 Examples of modern maintenance

Condition monitoring (CM) and *predictive maintenance* are two trending concepts in industrial maintenance. The first one relates to continuous collection of operative data from a given device to identify states in it. Predictive maintenance, by other side, seeks to analyze the information to generate future state estimations on each variable, which set predictions over failure before it happens [9]. Most of the times a CM strategy is applied to a machining center, the focus is to monitor one or two magnitudes, and then use analysis tools and historic data to compliment it. The work of [10] offers an example of this situation. The authors analyze the procedures of a producer dealing with tolerances of microns in his lots. To assess the real ability of the positioning system, a sensor measures the radius of circular paths executed by the spindle. With only one variable registered, the whole set of balancing parameters can be determined. The personnel environment collects temperature and humidity too, to create correlations with movement accuracy.

Going even further, the authors of [9] investigated the effectiveness of retrieving vibrations to determine deterioration on cutting tools. Using only vibration sensors attached to 3-axis CNC mill, with a sample rate about 1 kHz, the team could recognize properly the wear state of the tool, after cutting test pieces. On the same topic, [11] shows the use of a micro-controller with USB interface and a relatively cheap accelerometer to detect vibrations on spindle. With a cost under \$200, the authors were able to distinguish the signals reported for brand new tools and clearly damaged tools. One of their conclusions is that frequency analysis offers more accurate results than amplitude analysis when evaluating wear conditions.

A last example is shown in [12]. The authors based their research on the idea that cutting pieces emit acoustic signals between 2 kHz and 4 kHz, along with regular vibrations, but the former can be easier to detect with cheap sensors. They registered data for 4 different variables, at a sample rate of 44100 kHz: cut depth, feed rate, RPMs, and RMS acoustic level near the spindle, calculated as follows:

$$S_{RMS} = \sqrt{\frac{s_1^2 + s_2^2 + s_3^2 + \dots + s_n^2}{N}} \quad (2.1)$$

The authors were capable of mark some distinction between brand new and used tools, and a very remarked difference with worn out tools.

2.1.4 Hermle UWF900E parameters

The monitored machine for this project is classified as universal milling machine with 3 axis of movement. According to [4], the total travelling distances for its axis are 600 mm on X axis, 400 mm in Y axis and 420 mm on the Z one. The model that is used at DHBW Karlsruhe campus has a Heidenhain model TNC 407 controller, programmed with MasterCam software. The user manual indicates that this controller is on the low end of specifications (limited graph use, longer block processing time, etc.), but features

Year of production:	1995
Table total dimensions:	900 mm x 458 mm
Spindle model:	SK 40
Spindle travel:	75 mm
Number of places in tool magazine:	30
Tool rotational speed:	20-4000 RPM
Feed rate:	1-5000 mm/min
Power voltage:	400 V, 3-phase, 50 Hz
Control voltage:	34 V
Maximum power consumption:	7 kW
Net weight:	3200 kg
Cooling system:	Compressed air, max. pressure 7 bar
Other details:	Full protection cabin with transparent 3 door access, machine light included inside the cabin (lights on when powered)

Table 2.1: CNC Hermle UWF900E Specifications

communication through RS-232 and RS-422 protocols [13]. In table 2.1, there is a list of other important machine characteristics.

2.2 Internet of Things and cloud-hosted services

2.2.1 What is IoT?

The concept of Internet of Things refers to an upcoming tendency in communications technology. Explained in simple terms by [14], it consists in the integration of Internet-based connections not only to personal computers and related devices, but to almost commo, from household devices to work-place machines. The intention is to share device operative or environmental information in an extensive way, instead of maintaining the data and the control of it in internal-networks[15]. IoT development was possible due to the convergence of several factors: designers interest in recovering data from real source to help them with their work; better, cheaper sensor technologies, easy to integrate on other devices; ubiquity of Internet in common life, specially through wireless, broad band protocols; and the creation of new services for transmission and management of huge amounts of information (*Big Data* applications) time [14] [16] [15].

A general IoT application structure can be seen on figure 2.2. The sensors, which provide the data for the whole system, are located on the starting side. These elements

can be connected through wired protocols, or through Wireless Sensor Networks (WSN), although the latter are limited most of the times to an internal level of communication [15]. A gateway, a device with stable Internet connection and available ports for the sensors protocols, is usually added to link both sides. Considering the limited functions they need to display embedded options are preferred [14]. Once a gateway has been established, the data is within the local area network (LAN). To communicate at global scale, a dedicated *server* needs to run the required processes. Many IoT applications use third-party providers for this task, which allocate the programs in big, software-based company servers [14]. The most popular services provided through IoT include: database generation and maintenance, analysis of data, remote control of the machines, data display and prediction reports [16]. Some IoT systems, include security option for the collected information, such as user profiles, or *firewalls* in the gateway.

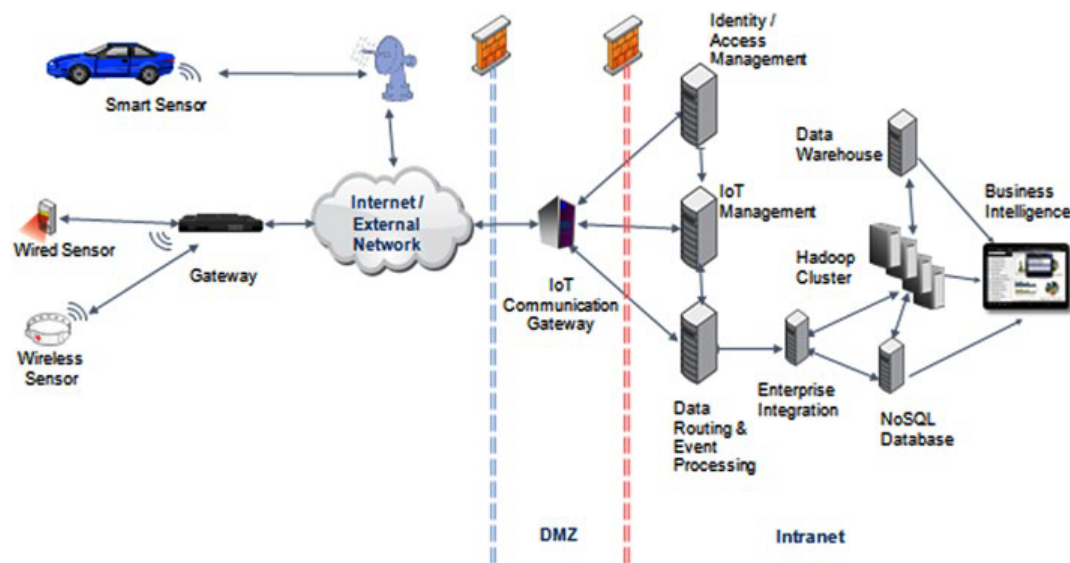


Figure 2.2: Example of the topology for an IoT system. Source: [16]

The main advantages of installing IoT systems are: the easiness to control direct functions or update machines control programs remotely; the possibility to keep real-time monitoring over wide conditions sets; the deep analysis that can be applied automatically over operational information; and the opportunity to extend user experience and get better designs at the same time [14]. Referring to manufacturing industry, it has been reported IoT is especially useful for managing automated process, such as CNC ones manu-handbook. On the other side, some of the biggest challenges to overcome when designing or implementing IoT are securing sensible information from unauthorized access; providing a reliable transmission path for all the collected data; and choosing the right sensors and analysis combination to provide valuable system perception [15] [16].

2.2.2 Cloud hosting and Platform as a Service (PaaS)

IoT solutions are commonly implemented through *Cloud Computing* tools. This concept is defined as a software development method where every program and file handled to the end user is ultimately stored in a remote server memory space and accessed through network communications[14]. The origins of cloud computing are linked with the rise of big web-centered companies. By the nature of their own work, these companies need huge clusters of Internet-ready database computers in their own facilities. Data centers usually have memory space that is not required at full-capacity, so the source company can get profit renting that extra space to other IT companies that are needing it. [17].

Depending on the layers of responsibility assigned to the provider, cloud services can be classified, in Infrastructure as a Service (IaaS), Platform as a Service (PaaS) or Software as a Service (SaaS). For the first one, the provider only offers access to the memory and processor of their own servers. In a PaaS, the provider gives the client the opportunity to use programming language environments, dedicated web page display, pre-configured databases, and other utilities. This is intended to allow software developers flexible tools without caring too much for maintenance. Finally, a SaaS provider has a full functioning program or application running on the cloud, and the clients can use it with their own files [18].

The advantages of using cloud computing are avoiding the costs and technical difficulties of installing an own cluster of servers and giving it maintenance; the ability and flexibility to use only the amount of processing that is really needed in each moment[18]; and the ubiquity of information, that can be reached from different devices and users without losing track of the latest changes [14]. The main challenge that remains for cloud computing is to provide enough security to the clients about their information [18]. The authors of [19] made a security analysis for the three types of service layers, and the conclusion is that PaaS clients need to interact thoroughly with the provider safety procedures, because it is their own job to call them correctly inside the programs.

Choosing a specific provider for cloud services depends on the reach of the desired project. The authors [20] offer a well-based guide to it, with an evaluation framework to compare providers based upon user environment and needs. The full extent of the method is no matter of discussion for this inform, but the general structure is a good reference to consider. The paper evaluates each provider on four attributes, divided in four sub-attributes each, and those ones divided themselves in metrics, that can be discretely assigned numeric values. The attributes used were agility, security, performance, and usability.

2.2.3 Databases: SQL vs NoSQL

Database deployment and maintenance is a key element in IoT design process. The different infrastructures that provide such service are mainly classified in two categories.

The first one is relational databases, also recognized by the usage of structure query language or SQL. These databases appeared around 40 years ago, when information traffic and query requirements were lower [21], so they focus on what is called ACID compliance: atomicity, consistency, isolation, and durability [16]. SQL databases are constructed through structured data, that means that every element stored is part of an explicit and pre-defined structure declared in the database management section. Common view for SQL databases is the one of a table, with sets of relationship rules established if a connection between tables is needed. Notice that, because the schema is used to validate the queries, SQL does not accept missing spaces when uploading a new data point [21].

For quick-update, parallel-scalable, Big Data applications, SQL have some performance troubles. Thus, the second database category, NoSQL databases, tackles this problem using partially non-structured data big-data. There are at least four main categories of NoSQL databases: key-value pairs, column-based, graph-based, and document-based. The first one is a very basic method, where values are attached only to a correspondent tag, but no further relationships are necessary. Column-based type adds the possibility to group different key-value pairs to be identified as whole object of data, although they still manage relationship freedom. A document-based database allows grouping in a little bit more structured way, including generating hierarchical subgroups. Graph-based service works with nodes of information and multiple relationships stored for each one [16].

The main advantage of NoSQL is flexibility, so if the application is known to have remarkable difference from data point to data point, getting one of those database is valuable. Relational bases are still useful when ACID characteristics are necessary (NoSQL are usually faulty in one or various of the attributes). The authors of [21] offer an extensive series of recommendations to choose the most appropriate type in different scenarios. Document databases, for example, are suit for web applications, with small and continuous reads and writes.

2.2.4 Basics of web design

Web page visualization is based in the Hypertext Transfer Protocol or HTTP, which defines the rules to execute hyperlinks and other main functionalities. HTTP works by sending method requests, each one related to different resources. The two most used methods in web design are GET and POST. The GET method asks the server for the file; the POST method covers the opposite function: to send some information from the client side to the server [22]. The HTTP scheme uses Hypertext Markup Language (HTML) documents to establish the specific layout and file paths of a given page[23]. To execute programming functions beyond visualization, like accessing databases, the servers have programs interacting with HTTP requests. The same programs that manage a web page can be easily integrated with other Internet-based functions, like e-mail, because they share a common basis in their procedures. In the specific case of mail, the protocol is called SMTP [24].

Some actions, mainly interactivity ones, need to run on the client side. HTTP cookies are one particularly useful example. These are small pieces of data, which are stored by the browser. Whenever the server requires, it can access to them or modify them. The core usage of cookies is to store static settings defined by the user for the web page navigation [22]. The programming language dedicated to this task is almost universally JavaScript, a high-level, weakly-typed and interpreted language[23]. Nowadays, JavaScript can also be executed in server side, primarily with the NodeJS environment. Node is recognized by web developers because its working scheme (event driven and with asynchronous execution of functions) is vastly efficient for the high input/output requirements servers must deal usually [25].

2.3 Communication protocols for fast processing

2.3.1 Serialization protocols

In systems that interact with heavy information flow, like most IoT applications, collected data should be used by all the parts involved with minimum effort. This requirement includes both user interactions, where human-recognizable formats are needed, and machine-to-machine communications [26]. For the latter, information is usually formatted into byte arrays to transmit it over connection lines in an easier way, a process known as *serialization*. Consequently, the receiving end must reconstruct the whole data structure from the array, or *deserialize* it [27]. The main issue with serialization is how to preserve internal data structure relationships all along the whole procedure; several different encoding protocols have been created to tackle the problem [28].

A first approach is to use XML, a format similar to HTML which is widely popular because it natively multi-platform, plain-text based and therefore, human-readable. Nevertheless, the use of tags, redundant by nature, increases the size of serialized byte array tends to several times the original object, and reduces clarity[27]. A similar alternative is JSON, the notation used by default in JavaScript to create objects. The structure of a JSON file is also made of plain-text, but it distinguishes different elements with simple symbols and writing order. The evidence collected in academic works suggests JSON serialization produces smaller arrays than XML, making it better for storage and faster to send [27].

Because computers work with binary representation, the logical step to maximize the usage of memory is to convert the full object into a binary file. BSON, for example, is an adaptation of JSON to a binary format. It is self-describing, meaning that machines do not require extra files to deserialize messages, and it also has internal hierarchy for complex structures [26].The other common self-describing protocol is MessagePack, which is also derived from JSON [29]. Another popular protocol is Avro. Although it is not completely self-describing (because it uses a predefined schema), it is classified as dynamically generated [28]. Avro was designed for the Apache Foundation and it is mainly used in big data applications [27]. Other serialization protocols rely entirely on a schema

file read before executing any serialization. The protocol needs to provide a compiler for every programming language it is intended to be used. The two more popular options within this category are Protocol Buffers (Protobufs), developed by Google for their own archives, and Thrift, with a similar situation for Facebook [28].

The references consulted point out that binary protocols produce more compact files, and therefore, they are more efficient in terms of storage and speed [27]. In [27], the authors run a benchmark test between JSON, XML, Avro, Protobufs and Thrift implementations for the .NET environment, both with small and big number of objects. The analysis shows after 100 objects there is a noticeable difference between plain text and binary formats, as binary protocols can be even 5 times faster and around 3 times smaller. Protobufs is the protocol with the best results in each test. The authors of [28] made a similar test run. Their results are more diverse, but they show again that plain text is always a step behind binary, with the worst performances being JSON implementations. The best options, according to this work, are: Avro in terms of arrays size, Protocol Buffers in terms of serialization time and Thrift for deserialization time. Finally, both [26] and [29] made a comparison between BSON and Protobufs. The conclusion is that BSON performance is on the same level of JSON, with sizes 5 times bigger and times twice the ones Protocol Buffers can achieve. Figures 2.3 and 2.4 shows some of the commented results.

Although BSON is the worst binary option in the papers referenced, its features tend to benefit storage and query functions. The files generated by BSON have a characteristic known as traversability, the ability of quick indexing and searching for specific values inside them. To achieve it, the encoding method add extra information in every document, creating a memory overhead, but it pays off during search requests. In some cases, all this overhead makes BSON files bigger than JSON equivalents, but keeping in mind that BSON goal is not set on size but accessibility, it turns out to be a good option for storage [30].

2.3.2 Google Protocol Buffers

Google designed Protocol Buffers to deal with their indexed servers requests. It was created as a more efficient alternative to XML with a fast, lean serialization system able to accept new data fields without upgrading servers, defined through easy to read schemes, multi-language, and multi-platform [31]. A specific Protobufs implementation starts with the schema file, called *.proto* file, which defines a generic message structure with the proper elements in it. To apply it into a program, the *.proto* file needs to be compiled for the desired language, be it C++, C#, Java, Python, Ruby, Go, PHP or JavaScript. The result of compilation process is a reference script that can be invoked inside other source codes. A programmer only would need to create the desired data structure in the preferred language, and call `serialize` or `deserialize` methods; that is, it does not get involved with the binary encoding itself [32].

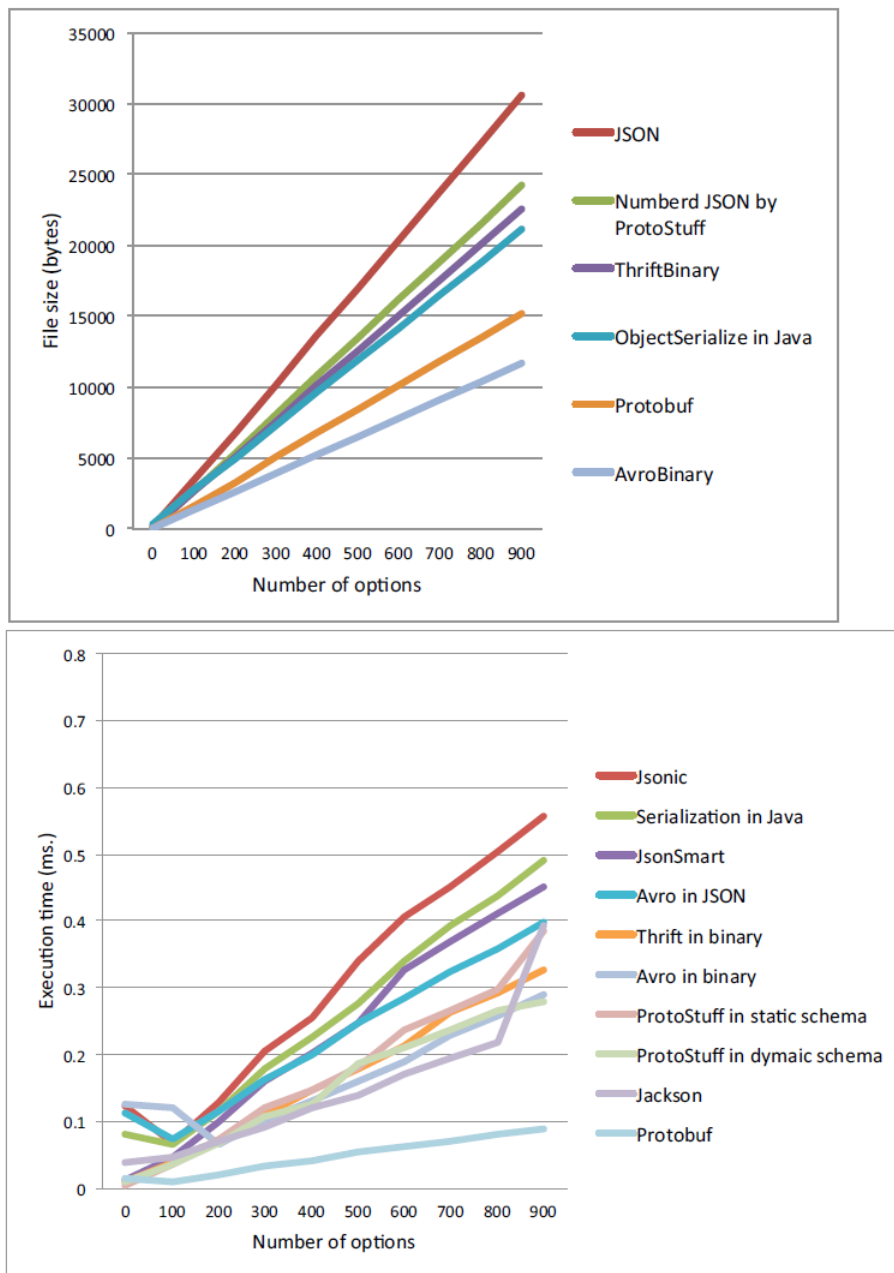


Figure 2.3: Comparison between serialization protocols. Source: [28]

When working with integer values, the serialization encoding executes adjustments to use only the necessary space, although some tagging is required for each variable. For floating point numbers and strings, there is no conversion protoencode. Protobufs can treat numerical arrays like strings, to avoid individual tags; and it has an option to encode signed integers, preventing the appearance of very large representations of negative numbers. The main disadvantage of Protobufs messages is they never include neither the total amount of bytes sent, nor any end sequence. Adding these consideration is a task left for the programmer [31].

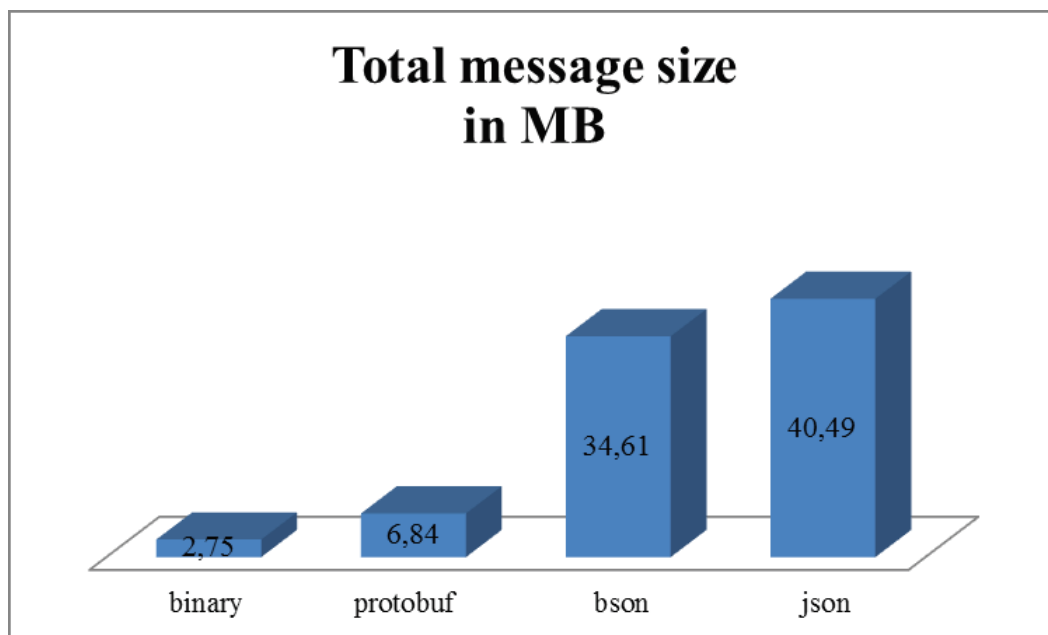


Figure 2.4: File size for different serialization protocols. Source: [26]

2.4 Filtering and data analysis on the digital domain

2.4.1 Processing filters

Although almost all modern technologies (including Internet protocols) are based on digital computing, many of the signals a monitoring system catches are still analog by nature. Thus, conversions are required for each of them. To assure the data stored in last term is representative of reality, Digital Signal Processing (DSP) techniques are necessary. DSP studies computing algorithms that manipulate analog signals after digital conversion, with the goal of enhancing their quality, recognize patrons or make them more suitable for transmission [33].

Noise is a basic concept in this area. When acquiring a measurement, physics phenomena can get involved in the generation and transmission of data, provoking an erroneous value to be registered. The range of perturbances includes environmental elements such as humidity, temperature, or magnetic field, but also unexpected situations that deviate the monitored condition instantaneously [34]. Also, high frequencies required in both control and transmission ICs introduce resonance effects contributing to signal distortion [35]. These alterations are considered completely random and for that reason, they cannot be totally avoided, only countered with the help of filters.

A common analog filter is normally composed of passive components (resistors and capacitors) and operational amplifiers. The basic design is one pole circuit, named after its transference function on the frequency domain:

$$H(s) = \frac{1}{1 + \frac{s}{\omega_c}} \quad (2.2)$$

The given function works for low-pass filters, with a cut frequency on ω_c . Noise is usually perceived as a high frequency element, so low-pass filters are a convenient election. Single component RC circuits behave as one-pole filters, as the transference function of the capacitor voltage can be written as:

$$V_c(s) = \frac{1}{1 + RCs} \quad (2.3)$$

Comparing equations 2.2 and 2.3, the formula for the cut frequency is obtained:

$$f_c = \frac{1}{2\pi RC} \quad (2.4)$$

Analog filters present some disadvantages for real-life applications. In first place, the passive elements introduce a source of uncertainty to the system, as the tolerance rates on commercially available versions can reach even 10%. Also, the same high frequency issues that affect paths of connection act over resistors and capacitors too. Finally, as [33] explains, the analog to digital conversion itself adds more random noise, that would not be perceived by the filter. Analog filters are still necessary to reduce the amount of noise that reaches the digital stage, but software filters should be added in any large-scale project. A digital filter never deals with the physical limitations analog filters have, and so its performance is several times better[35].

One characteristic that digital filters exploit to reduce noise is that, when generated by natural sources, its probability distribution can be well approximated with a Gaussian function. Furthermore, the Central Limit Theorem proves that even when the random values come from non-Gaussian sources, the total sum of them would tend to behave as a Gaussian function [33]. Because all the noise added to a signal can be interpreted in this way, simply averaging the measurements over a predefined period is effective in reducing the alterations and recovering the original values. The so-called moving average filter is then the first solution that should be tested when facing the problem of noise in digital systems. The mathematical expression of this filter is:

$$y[i] = \frac{1}{M} \sum_{j=0}^{M-1} x[i + j] \quad (2.5)$$

Where y is the output signal for the filter, x the input, i is every consecutive measurement taken by the system and M is a design parameter. Although its simplicity, moving average filters are commonly the optimal solution against noise [33]. In fact, for a given edge sharpness there are no other filters that reduce noise as much as averaging. The general rule is using larger values for M gets a better performance in noise reduction (in the order of \sqrt{M}), but lowers edge resolution.

Moving average is effective in time-domain situations. For frequency-domain situations where a specific bandwidth needs to be filtered out, other options (Chebysev, Butterworth, Bessel) are more suited [35]. Other statistical measurements (median, for example) can be used for time-domain filters, but the theory and practice predict results equal or worse than moving average [33].

2.4.2 Timing and digitization errors

Transforming a full analog signal into digital information involves 3 stages: sampling, quantization, and digitalization. The signal is always function involving an independent variable (time, frequency, or distance) and a dependent variable (the magnitude of interest: temperature, speed, etc.). In an analog environment, both are continuous. Digital systems, by the other hand, can only store data in discrete versions. In the first stage, sampling, the independent variable is converted. To achieve this, sample-and-hold components are used. They catch the instant value of the signal periodically and then hold it until the next measurement. There is a clear loss of information in this step, as the intermediate values between every request are never registered. This is the sampling error [34].

Next stage applies discrete conversion to the dependent variable. The analog-to-digital units have a predefined number of levels (expressed in bits) that can be taken by the output. Hence, the sampled signal is interpreted by ranges, assigning them the nearest lower or higher value inside ADC. Again, such process results in an accuracy loss, because many intermediate values are interpreted as the same. The situation is known as quantization error. In the last step, the levels are simply translated to binary numbers and stored in memory. The conversion is executed one-to-one, so no error is introduced in this stage [35].

The quantization error can be assumed as a deviation added to the real value of the signal, taking the maximum value of $1/2$ LSB (least significant bit). If that signal is not static, the difference between original signal and registered levels is effectively interpreted as another source of Gaussian noise, with standard deviation of $\frac{1}{\sqrt{12}}$ LSB. Sampling error cannot be treated this way. Most of the times, this error is simply ignored if the signal changes slower than the sample rate, because there is enough information to reconstruct the original data. But if the signal frequency is too high, there is no way to find the real original signal. The error is known as “aliasing”, and to avoid it, the Nyquist Theorem establishes the sample rate should be at least twice the higher frequency component measured. [33]

Another sampling error can occur when it is assumed that collected data represents one period of the given signal, a principle used in several algorithms. That forces the number of samples to cover the lowest frequency expected. In a general case, to avoid analysis errors:

$$N * f_s \geq \frac{1}{f_{min}} \quad (2.6)$$

Where N is the number of samples.

2.4.3 Frequency analysis: Fast Fourier Transform (FFT)

The Fourier Transforms are a family of mathematical tools useful in the frequency domain analysis. For digital computing, that work with discrete variables and finite amount of

data, the transform can be applied assuming one series of samples represent a period. The Fourier transform used in digital devices is thus based on discrete, periodical signals, and it is called Discrete Fourier Transform (DFT) [35].

The basic Fourier theory not only states a signal can be formed from sine and cosine waves, but tells the frequencies of all those waves are entire multiples of a base frequency. In the DFT the base frequency is established as the one covered by one array of samples. Then, the basic transformation functions are given by:

$$c_k[i] = \cos(2\pi ki/N) \quad (2.7)$$

$$s_k[i] = \sin(2\pi ki/N) \quad (2.8)$$

Where N is the number of samples. A DFT decomposition is made of $N/2+1$ sine waves and the same amount of cosine waves, always starting with $k=0$. The indexed series of each wave amplitude is called the frequency spectrum [33]. Another way of interpreting the output of DFT is treating the frequency spectrum as a series of phasors containing information of each frequency component, where the cosines are the real part of each phasor and sines (properly, the opposite values) are the correspondent imaginary part. [34].

The most common algorithm for DFT uses the correlation, an operation that shows how similar is a given signal to a predefined target. For discrete sampled signals, this can be done multiplying point by point both signals. In DFT, the targets are the sine and cosine bases. Then, the mathematical expression for this algorithm is:

$$\text{Re}(X[k]) = \sum_{i=0}^{N-1} x[i] \cos(2\pi ki/N) \quad (2.9)$$

$$\text{Im}(X[k]) = \sum_{i=0}^{N-1} x[i] \sin(2\pi ki/N) \quad (2.10)$$

Where x is the input [33]. A more compact way to express this formula is to refer to the phasor interpretation and use the Euler identity for numbers in \mathbb{C} :

$$X[k] = \sum_{i=0}^{N-1} x[i] e^{-\frac{2\pi ki}{N}j} \quad (2.11)$$

The previous algorithm is not efficient when executed by a computer. J.W. Cooley and J.W. Tuckey introduced in 1965 a method for calculating the transformation that can be even two orders of magnitude faster. Accordingly, it was called Fast Fourier Transform (FFT) [33]. The method is intended to be used with the original definition of the DFT, where the output has N elements, instead of $N/2+1$. However, for a purely real input it is possible to prove that:

$$X[k] = X[k + N/2] \quad (2.12)$$

So, in the end, only $N/2$ complex outputs are necessary. The main concept behind the algorithm is to divide the whole series of samples in one group of even-indexed elements and one of odd-indexed. Then, it is possible to state that:

$$X[k] = \sum_{i=0}^{N/2} x[2i]e^{-\frac{2\pi k}{N}2ij} + \sum_{i=0}^{N/2} x[2i+1]e^{-\frac{2\pi k}{N}(2i+1)j} \quad (2.13)$$

From the “odd indexed” group, one can extract one e factor element, defining both sums in a similar form. Comparing them to the equation 2.11, it is clear that each one corresponds to the DFT of their respective half. That can be expressed as:

$$X[k] = E[k] + e^{-\frac{2\pi j}{N}k}O[k] \quad (2.14)$$

Where $E[k]$ is the transformation for even-indexed elements and $O[k]$ for odd-indexed. Then, recurring to equation 2.12 and using the Euler’s Formula, the basic equation for FFT is derived:

$$X[k] = E[k] + e^{-\frac{2\pi j}{N}k}O[k]; \quad (2.15)$$

$$X[k + N/2] = E[k] - e^{-\frac{2\pi j}{N}k}O[k] \quad (2.16)$$

$E[k]$ and $O[k]$ calculation can be done recursively applying the same formula to each subgroup [34]. If the number of samples is a power of 2, the base case would be when one point composes each group. In that case, the frequency spectra are equal to the points themselves [33].

The advantage in terms of operations of the FFT over regular DFT is explained by the type of loops a program must run to execute one algorithm or the other. In common DFT, each value of the spectrum has to recover the whole array, and there are exactly N values of the spectra, so the amount of operations is proportional to N^2 . In the FFT, each stage has to cover the whole number of values of the array, but there are only $\log_2 N$ stages [35].

Chapter 3

IoT Monitoring System for CNC: Sensors and other electronic components

3.1 Summary of the design process

The starting point in this project was establishing a definitive to-be-monitored variable list, with the help of datasheets, data plaques on the machine, research papers revisions, CNC personal inspections and discussions with the Mechanical Engineering department personnel. The main criterion to select which variables would enter in the list was the possibility of measuring them without modifying the structure of the machine or interrupting its functionality. Later, it was possible to clarify the requirements for sensors and the microcontroller that would collect their measurements, including limitations on physical dimensions, operative ranges, resistance to external conditions and the total amount of components to be used. With this information, a research among common, low-cost options available in the local market was conducted, choosing the best option for each one of the variables. Then, knowing the type and quantity of signals that would be generated, an appropriate controller was chosen.

After getting all selected components, the next task was to execute some proof measurements, leading to a calibration process based on experimental data. Then, the microcontroller was programmed with a periodic measurement routine, considering the filters that were deemed necessary according to the empiric results, and maximizing programming effort and memory usage. Database connection and refreshment was delegated to a single-board computer (SBC), because it was too demanding in terms of processing power for the microcontroller. The work assuring MCU-SBC connection kept the information unaltered and without losses was done while the control programs were being created, and it was tested thoroughly after. Finally, the sensors and circuits set was installed on the CNC machine.

3.2 Component selection

3.2.1 Variables to be monitored

An initial approach to CNC maintenance programs was found in *citecnc-maintenance*. Such work was estimated appropriate, because its end goal is also a medium-size, 3-axis milling machine. Based on the author considerations, the following variables were remarked as relevant to the CNC performance: sliding-guides position and speed on each axis, CNC controller status, spindle motor speed, cooling bomb pressure output, security door state and power consumption on motors, bomb, and main electrical feed.

Direct monitoring of any motor condition was discarded from the beginning, because the position of these and other pieces like bearings or gears would force dismounting operations when installing any sensor. The decision was to keep an eye only on the secondary actions: speed and pressure of the bomb. The dedicated controller of the CNC works under proprietary software and all its interface ports are input-only, so operative data seems to be blocked by the manufacturer. Hence, the option of getting feedback directly from the CNC was pruned and software control status was eliminated from the variable list too. Power input for the CNC is high voltage and therefore, the feed cables on the workshop were installed underground. Even if an external sensor did the measurement, there is not enough space available to install it without danger. This magnitude was also ignored because the evident difficulties

Considering the deep level of analysis the authors of [11], [12], and [9] achieved in an environment similar to the Hermle CNC, vibrations and acoustic emission measurements were included in the project. Following the method applied in the references, these variables were monitored both in magnitude and frequency. Vibrations required 2 measure points, because the column and the table of the milling machine are separate structures. In the same way, the theory explained in [6] and [11] suggested working-zone temperatures observation could compliment the other measurements. In this case, 3 contact points were selected: tables, as “environment reference”; spindle, as working area; and cooling hose, to test the respective system effects. Finally, cabin light status was added to the variable catalogue, mainly because its role in safety conditions.

The final list of selected monitoring variables is: axis feed speed, spindle speed, cooler pressure, door status, light status, vibrations, vibrations frequency, noise, noise frequency and temperature.

3.2.2 Sensor selection

The sensor selection for each of the selected magnitudes was based on three factors: machine characteristics, price, and availability. The main features of the Hermle UWF900E are displayed on table 2.1. In terms of price the idea was to keep the expenses as low as possible, without sacrificing too much performance, so a total materials budget of

250 € was expected. Thus, a 20 € limit was chosen for electronic components. About availability, the intention was to make easier the ordering procedures for the DHBW administration, given that they oversaw approval of the purchase. Therefore, only local, online shop options were considered.

For the door status, the most suitable idea was to use reed contact switches. The features studied when selecting the specific model for the project were basic. The voltages and currents used for the contact were on digital control levels (5 V and few miliamperes), quantities that almost any commercial switch easily withstand. The passive state of the switch (normally closed or open) did not matter. The other detail were dimensions. The CNC door has a gap of about 2 cm when totally closed. A sensor with a slightly lower width was chosen.

In the light status case, the solution approach was to use phototransistor. The first searched condition was polarity: NPN transistors are easier to integrate in the role of switches to digital circuits. Voltage and current limitations were barely considered, because common transistors are designed for digital voltages. About size LED-type encapsulation was preferred, considering the sensor must be attached directly to a lamp. Finally, the sensitivity range was selected according to fluorescent lamps spectrum, which peaks around 500 and 600 nm. There are no transistor options with maximum sensitivity near these values; the better approximation peaks on 900 nm and its working range is between 400 nm and 1100 nm [36].

The acoustic measurement needs a simple and small type of microphone, to connect it without using extensive circuits to a microcontroller and install it as near as possible to the spindle. Electret mics are the recommended type. The frequency range was selected following the results of [12], where data lied between 2 kHz and 4 kHz. For the microphone sensitivity, values between -30 dB and -40 dB were accepted. A low impedance value was preferred, to counter the loss involved in the long installation cables. The model selected has a range between 100 Hz and 10kHz, -40 dB sensitivity and 2.2 k Ω impedance [37].

Linear speed measurements are quite hard to acquire with low-cost sensors. A better option is to derive them from acceleration, a magnitude that can be detected with cheap MEMS sensors, which have good sensitivity and small size. The principal requisite for the sensor was to distinguish between positive and negative movements in the 3 different axis. Following this, sensitivity range was estimated knowing maximum speed is around 0.08 m/s and system response fall in the order of 5 milliseconds. Then, maximum expected acceleration is around 2 g. Ideally, the sensor should be also able to detect the smallest speed changes, around 1/5000th of the maximum value. Even working with a 12-bit ADC, this cannot be done. Therefore, the search was limited to the upper limit. For directional sensors, the best sensor found has a range of around 3 g, and sensitivity of 300 mV/g [38].

Vibration measurements are basically acceleration transducers on smaller scales. The work done in [9] suggests that for heavy usage, values do not surpass 0.8 g, so a sensor with a 1 g range should be enough. The sensor selected has precisely this range, detects vibrations in every direction and has a sensitivity of 5 V/g, (experimentally estimated),

which is an improvement of 16 times over the directional sensors.

For bomb pressure, a gauge type sensor was required, as work load evaluation is typically done related to atmospheric level. MEMS integrated sensors are available for this task and they are the cheapest option. Working with a limit pressure of 7 bar (101 psi), it was estimated that a precision of 0.1 bar should be enough for the project intents. With a 5 V control voltage and 10 bits in ADC, this represents a sensitivity of at least 40 mV/bar or 2 mV/psi. The sensor selected has a range up to 100 psi and a sensitivity of 40 mV/psi [39], which is in turn 20 times the minimum required.

To make the temperature sensor calibration process less complicated and to reduce the amount of analog ports needed on the microcontroller, a digital output sensor was the main idea. The big advantage of these options is they process and compensate the signal, so data recorded does not need further treatment. The range expected for the sensors was between 15^o C and 50^o C, enough to cover CNC conditions. About sensitivity, knowing temperature changes over a disperse surface can be slow, a resolution of 0.5^o C was requested. The option chosen was Maxim DS18S20 model, which uses a communication protocol (OneWire) based on only one digital pin [40], highly beneficial regarding MCU limitations.

Finally, to measure spindle speed, the best option should be adapting an optical encoder to the corresponding shaft. However, such option was discarded because it is too invasive. The alternative was to apply the same concept of the encoder with existing structures. The spindle tool grabbing piece has two sections that stand out 5 millimeters. When rotating, detecting these sections can give an indirect measurement of speed. To execute the detection, an optical sensor is needed. Discrete output distance sensors are adequate for the task. The distance between the outer diameter of the spindle and the side of the column (where the sensor could be attached) is 5 cm. However, no commercial options for distance sensors operate on that range. The closest option found was 15 cm [41]. On table 3.1, there is a summary of all the selected sensors.

3.2.3 Microcontroller and other ICs

The features expected from the microcontroller were mainly determined by the operational characteristics of the sensor set. Bearing in mind 7 of the selected sensor outputs are analog, an ADC with that number of entries was necessary; to cover the most demanding measurement, units of at least 10 bits should be used. The microcontroller should also be able to handle at least 3 different externally triggered interrupts: door sensor, light sensor, and spindle speed measurement. Finally, another digital input is required for the temperature sensors. The general structure considered was 8-bit architecture, because computing load was estimated to be low.

To execute frequency analysis, adequate sample rates should be set in the ADC. The stable sample frequencies for ADC in 8-bit MCUs is around 200 times slower than the clock frequency, and the highest expected frequency in the measurements is around 10

Variable	Manufacturer	Model	Quantity	Price by piece ({ € })
Door Status	Sygonix	18734Y1 Normally Open	1	6.99
Light Status	Everlight	Opto PT331C	1	0.36
Noise	Kepo	KPCM-94H65L- 40DB-1689	1	0.63
Linear acceleration	Analog Devices	ADXL335	2	13.59
Vibrations	Iduino	801S-SE040	2	10.49
Fluid pressure	Honeywell	40PC100G2A	1	9.99
Temperature	Maxim Integrated	DS18S20	3	3.54
Distance (for spindle speed)	Sharp	GP2D150A	1	26.09

Table 3.1: Sensors used in the project

kHz. On a conservative basis, the minimum clock frequency accepted should be 8 MHz; hence, values from 16 to 20 MHz were searched. For timer interrupts, the idea then was to use only one and distribute the measurements. However, because some variables would change slowly (temperature) or be very stable (bomb pressure), the time range between timer interrupts was estimated in the order of seconds. With a clock frequency of 16 MHz and the standard maximum pre-scaler of 256, only 16-bit timers can reach these values.

Sample arrays for frequency analysis were estimated to contain between 128 and 512 measurements. The software structure that hold them needs to allocate memory statically on the MCU RAM. Worst-case scenario the amount of memory required would be 2 KB. The program itself was also foreseen to be relatively large. In that sense, practical experience indicated that 28 or 32 KB for program memory should give enough space to work without any problem. Finally, the only communication protocol port needed was UART, wich provides serial communication with the SBC with huge flexibility, especially regarding wireless systems.

A research of models that fulfilled the conditions was made in a main MCU producer webpage[42]. From all the available options, the best 3 ones are shown in table 3.2. The abilities of all of them seem almost the same. The Atmega 328P has a pin count slightly lower. In that way, it seem less oversized. There is also another advantage of the 328P model: it can be easily programmed and debugged through the development platform Arduino. The Mechanical Engineering Department already has several of these devices and no other dedicated MCU programmer, so this characteristic was relevant. For those reasons, the Atmega328P was chosen.

Although the regular version of the 328P has effectively 8 ADC channels, the DIP encapsulation bought only includes 6 analog pins, missing a channel for one of the measurements. To solve the issue, an analog multiplexer was added. A 2 to 1 switch with TTL level on selection channels and transition times under 1 ms (to avoid errors in the ADC measure-

Model	Atmega 328P	Atmega32A	Atmega324PA
Program memory	32 KB	32 KB	32 KB
RAM	2048 bytes	2048 bytes	2048 bytes
Pin count	32	44	44
Max CPU speed	20 MHz	16 MHz	20 MHz
RAM	2048 bytes	2048 bytes	2048 bytes
ADC channels	8	8	8
ADC resolution	10 bits	10 bits	10 bits
Timers	2 x 8 bits, 1 x 16 bits	2 x 8 bits, 1 x 16 bits	2 x 8 bits, 1 x 16 bits
Communications	1 UART, 2 SPI, 1 I2C	1 UART, 1 SPI, 1 I2C	2 UART, 3 SPI, 1 I2C
External Inter-rupts	2 predefined, 3 Pin Change options	3 predefined, 4 Pin Change options	3 predefined
Price	3.39 €	4,87 €	3,97 €

Table 3.2: Comparison between microcontroller models

ments) should be enough. For availability reasons, the IC acquired in the final version is a 4 to 1 multiplexer, with a transition time of it is typically 130 ns [43].

All components selected, except the directional accelerometer, work with 5 V power input. The best idea to power the system is to keep a main line on this voltage, and add a DC-DC level converter, with an output of at least 3V (per accelerometer manual [38]). It was intended to work with a fully integrated structure and a minimal output current 10 mA without heatsink. The IC chosen goes up to 500 mA without heating problems, and the output (considering ripple) can go from 3.275 to 3.325 mV [44], so it can manage the sensors supply. In SBC selection, the best price/performance ratio was found for Raspberry Pi models. A Bluetooth connection between SBC and MCU was the choice, as this protocol is one of the most reliable low-power, wireless formats. The Raspberry Pi model 3 includes in its structure an integrated Bluetooth module. This characteristic gave it the edge over the other models

Electret microphones sensitivity is low. To obtain a useful ADC measurement, an amplifier must be applied. In audio applications, the model LM386 is the most popular option. Having a default 26 dB gain [45], it was good enough for the project needs. Finally, during MCU-SBC communication, the controller side needed a corresponding module. There are two ICs available in the market that can execute the serial-Bluetooth conversion: HC-05 and HC-06. The main difference is that HC-06 can only be used as slave module, while HC-05 can be both master and slave. For debugging tests, the second one was deemed more adequate. Table 3.3 shows a list of the selected ICs.

Function	Manufacturer	Model	Price by piece({ €})
microcontroller	Atmel	Atmega328P DIP package	3.39
Analog multiplexer	Analog Devices	ADG409	9.25
Buck DC-DC converter	ROHM	BP5275-33	4.90
Single Board Computer (Gateway)	Raspberry Pi	Model B version 3	34.99
Audio amplifier	Texas Instruments	LM386	1.09
Bluetooth module	ALLNET	HC-05 6 pins	15.30

Table 3.3: Other integrated circuits used in the project

3.3 Microcontroller programming

3.3.1 General structure and timing

The microcontroller program takes care of scheduling individual samples for each monitoring variable, executes signal processing, formats data points according to the elected communication code and finally sends the information through the UART port. The protocol chosen for the serial communication was Protocol Buffers, because its performance is better than similar options, as explained on chapter 2. General structures for the messages are created at the start, and the data slots are adapted whenever a specific data sending takes place.

For most of the variables (temperatures, linear accelerations, spindle speed, noise, and vibrations magnitudes) it was deemed enough to send only one data point per minute. Using smaller time frames simply saturates the database with extra information that could be not useful. The cooler pressure was set to a smaller rate, one data point every 10 seconds, because there seem to be small oscillations that were important to register. Finally, for the noise and vibration data, although the intention was to save them every minute, frequency analysis (executed in the SBC) requires a data exchange every 5 seconds. Anyways, the timer was set to 1 second, to cover the needs of filtering methods for multiple samples.

The door and light status are only registered when a change happens (for that reason, their sensors are attached to an external interrupt). To correctly asses condition value, a variable reads the initial state of the interrupt pins, and its value is toggled by the interrupt routine. The changes are heavily exposed to quick oscillations, then a bounce-rejection code is included in each call. By rule of thumb, interruption routines should be kept as short as possible to avoid process collision in the controller; and data transmission

procedures are lengthy. Hence, the information about status is sent with the next scheduled measurement. The interrupt merely sets a flag and it is cleared when the change information is sent. Spindle speed monitoring also uses interrupts, through a counting method. In this case, an auxiliary variable keeps a revolutions count, until the schedule cycle calls to send the value (once every minute) and resets the variable.

The timer interrupt keeps an internal count, which is used to send the different measurement information in a staggered fashion. The idea is that, in worse-case scenario, two measurements are sent in a 1 second period, to avoid data collision and loss in the interrupt-driven measurements. The data transmission process can take as long as 100 ms, so the scheduling scheme applied assures a 90% of accuracy in such magnitudes. A special situation happens around temperature registers. They interact with the MCU through OneWire protocol and the dedicated Atmel library. To retrieve a single measurement, the protocol sends first a command to start a conversion, which lasts 100 ms to be completed [40]. To avoid the mentioned data loss, in any scheduled communication, the temperatures measurements use 2 different slots of 1 second, one only to start conversion, and the other one to receive and transmit it.

Other important settings are the UART baud rate, fixed to 115200 by empirical tests, and the ADC frequency. By Nyquist, a sample rate of 20 kHz should be enough for it. However, as a safety factor, a value around 3 times was selected. By datasheet [46], the fixed frequency should be 70 kHz, but different runtime operations slow it down to 60 kHz.

3.3.2 Serial communications

To implement Protocol-Buffers in the MCU, a C-based version dedicated to low-memory applications (<10 kB), known as Nanopb, was applied. In this case, the base data structure for the protocol is a C struct type. The encoding procedure uses an auxiliary data type, the “stream”, that preserves the information of the process. Basically, the stream takes an array and fills it with the encoding coming from a given struct and the compiled reference files, and then saves the number of written bytes. One useful Nanopb feature is that every variable size data element can be limited to a maximum, allowing static allocation.

Two different types of message were defined: one to manage the arrays of samples needed in frequency analysis, and another one for the rest of one-value measurements. There are several reasons for marking a distinction. The sample arrays for both noise and vibrations can use the same number of elements, so this option can be defined only once in the program. If the same message was used for all measurements, constant changes should be done in runtime. Another detail is that, given its high memory size, the frequency-oriented message was thought to include only integers as sample (which Protobufs compresses better), but one-value data points include floating-point variables. Finally, the difference makes slightly faster the SBC-side processing.

The one-value message consists of 3 required spaces and 3 optional. The required ones are: monitored machine name, controller version number and monitored condition. One optional space, named “sensor”, is used to distinguish cases where one variable has different contact sites. The other 2 spaces are for the measurement itself, one integer and one floating point. The strings inside message were limited to a maximum of 20 characters. The array message has the same 4 first spaces one-value, but it has a *repeated* space, to store measurement samples. The intention was to cover frequencies starting in 100 Hz, which in turn requires a sample number of 512, considering a 60 kHz and equation 2.6. However the highest array size allowed by the RAM space of the Atmega 328p is 128 samples, which works for a minimum frequency around 400 Hz, value that is still useful following the ranges used in [11] and [12].

The controller program only initializes one “stream” and one-byte array. The stream is reset every time a new data point is transmitted. The byte array is limited to 300 bytes (an extreme case for the regular messages used). After each byte array is written to the UART, a series of four “255” valued bytes is sent, serving as separator sequence. Such sequence was chosen because it is impossible for it to appear in the Protobuf encoding using 4-byte floating point numbers.

3.3.3 Measurement processing

The variables door status, light status and spindle speed do not need any further processing, as they state discrete changes in the machine. The temperatures are processed by the pre-calibrated circuit inside the sensors. The tests done with them show a very stable output, with acceptable accuracy, so no other operation was applied over them. For the rest of the signals, different levels of noise were detected during the calibration tests. Following the recommendations given by [33], an average filter was the first option tested. In the MCU, the averaging is done first with the raw ADC results, and then respective conversions to right units are applied.

The pressure sensor required a not especially large averaging. During lab tests, an average with the 10 measurements was enough to achieve a measurement almost constant. The initial deviation in the linear accelerometer was smaller. Consequently, using a filter with 10 measurements provided again a smooth output in first place. After additional noise was perceived in experimental environment, the quantity of measurements was extended up to 60. For the acoustic magnitudes, a double step averaging is executed. In the first stage, RMS value of the sampled signal is extracted every time an array is registered, following the same procedure employed in [12]. Still, there is too much noise perceived between the successive magnitudes. An second, 12-element averaging was the solution implemented. The vibration sensors were by far the ones with worse performance. Noise could be reduced by an order of magnitude with a moving-average filter of 100 elements, but performance did not really improve with larger filters. Other variants of statistical filters (median filter, for example) were tested, with similar results. In the end, the

decision was to repeat the same method used for acoustics) and try to reduce external noise with a simple analog filter.

The frequency analysis done with microphone data seemed to have heavy components of distortion in some cases, and no deviation in others. This was caused by outliers in some of the sample lots. To prevent any further problem, the same averaging that was being used for the magnitudes was applied to frequencies. There was no lab test done for vibration frequency, because the lab lacked reliable references, but with the existent experience with the sensors, it was highly probable that noise would also affect it.

To correctly detect acoustics frequency main component, microphone offset (previously detected through calibration) is subtracted from the signal before sending the data. In this way, the first base frequency, zero, is ignored. The vibrations sensor does not have a clear offset, though, because it is multidirectional. The solution applied is to subtract the average value of the array from each sample.

3.4 Physical installation

3.4.1 Electronic circuits

All the sensors outputs are connected to one corresponding pin in the microcontroller. However, each sensor required different conditioning circuits. The simplest sensors are the ones used for door and the light status, which are just switches. The only circuit they need is a pull-up resistor connected in series. For the door switch, an arbitrary value of 1 k Ω was selected, merely to guarantee a few milliamperes current. The phototransistor, however, have voltage drops in the emitter-collector junction. As the collector current gets higher, the leakage voltage in the junction grows. With a 1 k Ω , the generated current is enough to drop around 3 V, outputting a “high” value too low for TTL levels. Augmenting the resistance to 10 k Ω solves the problem, reducing the leakage to less than 500 mV.

The distance sensor used for spindle speed operates under open-collector technology. Therefore, it needs a pull-up resistor in the output pin. The value was set to 10 k Ω , to avoid voltage drop problems. The temperature sensors also require a pull-up on the communication wire, according to the datasheet [40]. The recommended value is 4.7 k Ω , but 1 k Ω resistors were employed successfully.

The microphone involves a larger circuit. It uses a resistor in series to a power source, to generate the output signal, which later needs to be amplified. To couple the mic output with the amplifier input, a 10 nF is connected between them. The amp IC does not require any additional components to use the default gain, only the usual ground and power connections. An additional RC filter was added between output and ground lines to filter out noise. Datasheet recommendations, set the filter components to 10 Ω and 50 nF [37]. According to equation 2.4, this filter should block frequencies over 300 kHz. It was necessary to limit this frequency even more, to around 30 kHz, so a 500 nF

capacitor was put in place. An additional filtering capacitor for the power supply was placed between a 5 V and ground. Noise from this source should not be especially high, so an arbitrarily low capacitor (4.7 nF) was deemed enough.

Vibration sensors outputs pass through the inputs of the multiplexer. On the output of this piece, a grounded capacitor was placed, to filter out high frequencies. The value was estimated to follow microphone filter design (considering both should face the same frequency analysis). However, the multiplexer already provides an ohmic resistance between 100 and 170 Ω . Using the equation 2.4 again, the final value for the capacitor was 47 nF, which works for frequencies between 20 and 33 kHz, as intended.

The MCU is connected to other 2 components. A 16 MHz crystal with paired capacitors (10 pF) is used for clock generation. And the Bluetooth module is connected to the UART ports. Between MCU Tx pin and Bluetooth Rx pin, a protection resistor was placed, because the logic levels on the module work with 3.6 V. A full-circuit diagram can be seen on appendix C.

A PCB was designed and fabricated to serve as a permanent installation. All sensor connections were inserted via screw secured blocks, that give a mechanical safety level to the connections and are more flexible to use than just soldering. Several blocks were connected to power and ground, to make easier the access to both voltages. The power supply required should provide 5 V DC and at least 500 mA, as proven by table 3.4. A cell phone charger can perform this task, and the lab counted with several of them. To adapt the charger port to the PCB, a micro-USB port was soldered to the board, using only the power and ground pins of it. The diagram of the PCB can be seen on figure 3.1

Component	Current per piece (mA)	Total consumption (mA)
Door sensor	5	5
Light sensor	0,48	0,48
Microphone	0,5	0,5
Accelerometer	0,35	0,7
Vibration sensor	20	40
Pressure sensor	10	10
Temperature sensor	1,5	4,5
Distance sensor	50	50
MCU	10	10
Analog multiplexer	0,5	0,5
Audio amplifier	250	250
Buck converter	0,67	0,67
Bluetooth module	40	40
Total		412,35

Table 3.4: Current consumption in the system circuits

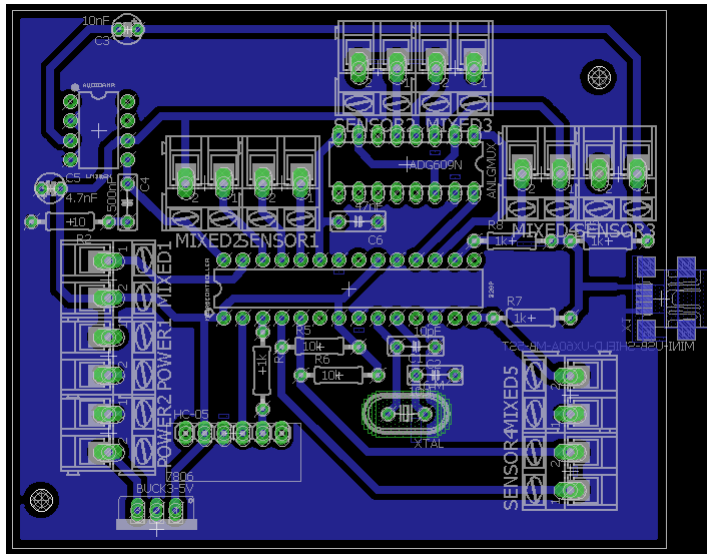


Figure 3.1: PCB design. Source: Own elaboration

3.4.2 Installation on the CNC machine

The sensors require placement all over different places in the CNC machine. To keep a connection with the controller, long (around 6 m) cables are necessary. Thinking about the environmental, and inherent cable noise, shielded, pair twisted, CAT 5 cables were chosen. These cables are designed to have low percentage of losses even in distances over 10 m, their coating reflects external magnetic interference, and the pair structure cancels out the antenna effects on the wires. Each coated piece of CAT 5 cable has inside 4 pairs of single core wires, for a total amount of 8 individual wires. To maximize the usage of this material, the sensors were grouped in the basis of their expected location.

Five sensors need to be placed over the column or overall near the spindle. Those are: the microphone, one vibration sensor, one linear accelerometer (the Z feed is exerted by this piece, not by the table), two temperature sensors (the cooler hoses are kept around the column) and the distance sensor for rotational speed. The required wires include one for each sensor output, one for 5 V, another one for 3 V and one connected to ground. Both temperature sensor can share the output line, effectively making one CAT 5 cable enough for the whole group.

The other group of sensors is placed on the table. This location was chosen for the following components: one temperature sensor, one vibration sensor and one accelerometer with two outputs (for X and Z feeds). Again, lines for 5 V, 3 V and ground are required, so 7 wires are enough for the group. The sensors for door, light and fluid pressure are located far enough from both locations and from each other to be connected in the same cable. Un-coated pieces of wire were used for them. Both the door switch and the phototransistor use only two wires, so they could be properly twisted with each other. The pressure sensor uses 3 connections: 5 V, ground, and output. In this case, a twisted pair was used for output and ground, and the 5 V was an independent piece of wire.

Chapter 4

IoT Monitoring System for CNC: Internet communication

4.1 Summary of the design process

While selecting sensors, a parallel research for cloud database and remote server framework was done. Main characteristics for this section were low-cost and compatibility with open-source software. Also, keeping in mind the system built in the project is intended to expand to other machines, standard protocols use and compatibility with different electronic and software elements were very important. When the entire web platform was selected, a decision for the programming language was made, based on integration with software tools. In this same process, chart generation library was chosen, putting special attention to embedded ability for managing large amount of data and dynamically visualization changes. The options analyzed also had to be on the high-level side, to let the work focus in information retrieval itself and not on the graphics generation.

The system for data classification exposed in chapter 3 was adapted to the communication chain, mainly in database generation. Considering both the receiving protocols in the SBC, and the determined requirements for database uploads, a service routine was programmed to take charge of data transmission stage. The database safety characteristics were exploited to avoid unauthorized information access. Another service routine was deployed to execute basic level, real time analysis with uploaded data, specifically detecting general work states in the machine. Later, web page server management was programmed, using the software packages previously selected. It was optimized to refresh its contents with every browser load, through database queries. Graph visualization was also designed to dynamically change with each restart. At last, the user interactive options, like graph selection, were added. Safety cautions were included since the beginning of web page work

Finally, an internal performance monitoring service was added to the SBC, to look after some critical variables for device right operation. This program, usually known as

“watchdog”, avoids unexpected failures in the communication flow. The version installed for the project was rather simple, just considering five different magnitudes. For each of them, the limits were imposed in a range wide enough to keep the computer working without emergency reboots, but without excessive alarms.

4.2 Software selection

4.2.1 Database

The time frames established for microcontroller-SBC communication are in the order of ten per second. This requirement implies processing involved in every new information package upload to database needs to be relatively fast, to avoid data collisions and additional losses. Then, the database chosen for the project should be reliable, in the sense it is already tested, and not just an experimental environment; and must have quick indexing system. Also, the system should consider data points are extracted from different kind of sensors. According to the proposed design, the value measured for some of them is a discrete variable, and for others it is a numerical continuous value. Some of the monitored conditions are covered through different specific sensors. Nevertheless, all collected information needs to be stored in a unified database, to allow the user quick access to any desired set without adding more interconnections that would turn GUI slow and safety-weak. The general panorama is data entries seem to differ from each other in structure terms, but they should to be stored and indexed in the same way. As discussed in chapter 2, NoSQL databases are especially useful in such situation.

Considering the recommendations made by [21] about appropriate databases for different uses, a document-based one is the optimal configuration for the project needs. Document-oriented databases are also highly popular for web applications, which is convenient for the end user interface. Additionally, these databases are highly unstructured, even by NoSQL standards, which in turn gives enough freedom for uploading very different data points and also opens the possibility to upgrade or change the structures stored without any additional effort. This feature was especially useful during development, as different formats were tested before arriving to the final one. Finally, document-oriented bases are well suited for adapting different pattern recognition and analytic searches. Although this function was not applied in the project, the intention was to make easier further research in the same area.

Once database choices were reduced to document-oriented category, a quick research on theoretic references showed the most commonly used and time-tested options are MongoDB, CouchDB, and DynamoDB. There is a good amount of supporting resources available for all these frameworks; there is also empirical evidence of their proficiency, with user-cases from well-established tech companies (Cisco and Comcast are using MongoDB, for example [47]). Therefore, the three of them complied with the reliability expected. DynamoDB distinguishes itself from the other two by being a proprietary, paid software from

Amazon Web Services. Although it offers a free usage, basic level deal (thus eliminating the costs problem), one of the goals in the whole project was to use as much open-source software as possible, because that work model opens more optimizing possibilities for future designs.

Between Mongo and Couch, the main difference found was data management protocol. CouchDB uses plain JSON for both data communication and information storage. MongoDB includes BSON as its main resource for information exchange. According to chapter 2, one can consider BSON implementations have the cutting edge in terms of database utilities over common JSON applications. The other selection criterion was the offer of cloud-hosted services. The aim for the project was to take advantage of cloud technologies, so database was planned to be host in that way. Henceforth, IaaS structures were analyzed for both database options. CouchDB mostly has beta tests and recently published cloud storage versions, something that MongoDB surpasses slightly with a couple of well-established companies offering the service. In the end, MongoDB seemed to be the best option for the project requirements.

The provider selection was based in developer's popularity, helping resources amount and pricing. The system is intended to be low-cost, and mainly with academic purposes, so free options were highly considered. Most cloud services providers charge for used processing resources, but in small programming load applications, some offer free access with limited functions. Specifically talking about MongoDB, two main options were found: Mongo Atlas, from the development team of the database itself, and mLab, from an independent source. Both have free tier services with up to 500 MB of storage, basic secure access options and user management, and the possibility to select the location of the server (for additional physical security). mLab has more inter-connectivity with other programming tools, and for that reason it was selected.

4.2.2 Server side management (PaaS)

The arguments exposed in chapter 2 show a PaaS is the most adequate framework to adapt the project structure to a web based user interface. Although an IaaS provider could be enough for server programs deployment and maintenance, the limited functions this structure provides to developers were deemed too short for the project goals. The programming tools used for GUI creation were mainly in the high-level end, because networking designs or server management are not part of the project extent. In the other hand, there is not a full working SaaS that could be adapted for graphs generation. SaaS products are also too restrictive for the envisioned design idea, since they do not provide enough access to customizing options.

To have a well-reasoned PaaS choosing process, the evaluation criteria were based in the work of [20]. Considerations about learning and support resources were done too, as the initial deployment of the web page, being a subject particularly different from other sections, required some research beforehand. Just like with the databases, pricing

had also a big weight in the final decision. The contrasted alternatives were Google App Engine, Amazon AWS, Microsoft Azure, Heroku and RedHat OpenShift. Only free tier services were evaluated, to keep a realistic focus on the analysis. Although Google, Microsoft and Amazon offer almost all the paid functions in their free trials, these ones get cancelled after 12 months. Free tier services from the other providers are permanent in principle. Looking to future work expanding the reach achieved in the project indicates temporary options could carry problems in the future. Other important details pondered during evaluation were: web page design is intended to be mostly static and it does not require constant computing; the graphic interface is not thought to be replicated through multiple users; and connectivity with the selected database from the PaaS was completely necessary

Main characteristics of each provider are summarized on table 4.1. Google and Amazon have better agility and security performances. But when specific project requirements are remarked, Heroku is the option with better qualifications. A big role is played by mLab connectivity, a service no other provider offers natively. Heroku also allows full web site management control through terminal commands in any Linux distribution. This possibility is useful in design testing and optimization, and it makes processed like features introduction and crash recovering considerably faster. Hence, the choice was made in favor of this platform.

Provider:	Google	Amazon	Microsoft	Heroku	RedHat
Computing time (monthly):	270h	750h	750h	1000h	540
Code storage size:	1 GB	5 GB	Unlimited	Unlimited	1GB
Agility score:	1.326	1.235	0.978	1.201	0.985
Security score:	1.123	1.578	1.429	0.976	0.884
Performance score:	1.124	1.345	0.967	1.345	1.076
Usability score:	1.432	1.318	0.754	1.546	1.020

Table 4.1: Comparison between different PaaS providers

4.2.3 Programming language and additional libraries

Heroku offers the possibility of programming the server-side management in the following languages: NodeJS, Ruby, Java, Python, PHP, Go, Scala and Cojure. MongoDB has connectivity extensions for most of them, too. Programs to be deployed in the PaaS are server-side web page management routines. As explained on chapter 2, the universally used language for client-side computing is JavaScript, so choosing its server version (Node) was deemed a natural progression. Although all the mentioned languages can manage the

web site design, it was expected that a Node implementation would be simpler and more adjusted to the final product environment. Moreover, MongoDB works with BSON format for its storage, but all the database uploads and queries are interpreted through JSON, the human-readable origin format for BSON. JSON is the native object representation for JavaScript, thus using the language allows a direct management of the database activities without any additional parsing, which keeps the programmed code organized and time-efficient. Finally, NodeJS is widely recognized for its asynchronous approach to task execution, implementing parallel computing without losing track of the newest data loaded. This characteristic could be useful to manage the multiple queries involved each way the web page is requested.

Once NodeJS was selected for web page design, it was considered natural to apply it to the other web communications services. In this manner, all the Internet section would be done with the same parameters and functions, avoiding problems such as incorrect use of exchange methods between languages, and keeping the whole development process more organized and code-compact. Programming all web sections with Node also allowed more efficiency in certain areas. For example, data point upload service, which is locally run by the SBC, but needs database access. The incoming data from the microcontroller is already encoded with Protocol Buffers format and requires an initial parsing, so eliminating other steps in the same vein is highly advantageous.

Most functions on web programming were execute through server-located libraries. NodeJS has a cloud-based, open source, free repository, and package manager for external source libraries, called *npm* [48], operated directly from command line. Installation processes are simply called from it. Two libraries deserve a special recognition. For web page visual style enhancement, some elements *Bootstrap* were recalled; meanwhile, the graph generation was achieved with *Dygraphs*, a high-level, interactive library specialized in this task. Neither of them was installed on the server nor the SBC; the client-side executes them instead. Processing load perceived in the PaaS is lowered, but in exchange, the result quality is highly dependent of the user equipment and Internet browser. Still, computing load for both visual operations was little enough to be managed by most of modern days browsers.

4.3 Processing services on the SBC

4.3.1 Sensor data parsing and database updates

The basic function assigned to the SBC was to act as main database upload gateway, because the function was too demanding for a common microcontroller. Therefore, the program in charge of executing it was the first and most important service implemented in the device. As explained in chapter 3, there are two different type of messages defined for information exchange: one for single measurements and another one for registering large arrays of samples. Both are encoded through Protocol Buffers C language version

before being shared through a Bluetooth connection. The database, however, is accessed through JSON format, and the upload program itself was programmed in JavaScript. Then, the first step was to translate the information received to the necessary format.

Bluetooth parsing was easily adapted, because the microcomputer run a light weighted Linux distribution (Raspbian, the default option for Raspberry equipment), and therefore, communication ports are simply accessed as directories in the file system. Moreover, the Bluetooth port in the Raspberri Pi 3 is internally connected to one of the UART (serial communication) ports of the main microprocessor. Then, any information exchange in the device can be interpreted as a native serial message. The Raspberry was able to interpret the raw, encoded messages as arrays of bytes in a time persistent way, which means after one call it would stay listening to any incoming data.

Next step was to deserialize the Protobuf code. For the SBC, the methods defined by the compiler for transform any message instance to a predefined class element, with attributes and methods. JavaScript classes are exactly represented through JSON using the *toObject()*, from the Protocol Buffers standard operation. The following stage is to directly load the data points to database. MongoDB has official backed methods for connecting, authenticate credentials and upload new “documents”, but they require to manually manage some access options on each new upload. For this project, a third-party library called *Mongoose* was used instead. The connection and authentication are done only once and in one step. Mongoose needs other element to work properly, though: a data schema definition, provided with an external JSON file. Although schema-less operation is a basic principle in NoSQL, Mongoose only uses them as loose reference for indexing document collections and to implement queries in a more efficient way.

The schema definition was adapted from .proto files, with changes. First, there are no array spaces in the schema, because storing a whole array in the database was perceived as inefficient and possibly problematic. The decision was to only store single measurements in the database, and for the array-type messages, report only the most significant value obtained. The other detail is a time-reference is necessary on each point make sense about data collection. This could not be added by the microcontroller, but the SBC, through the Internet connection, can get a local date and time reference. So, the JSON schema has the additional key-value: *timestamp*. On each message, all the other attributes of the JSON document are set by the deserialization method, but the timestamp program-defined.

The nature of the CNC working cycle is to be active only on certain day periods and be idle most of the time, causing most monitored conditions to have a zero value during several hours. It is not useful to have continuous null data points during idle periods. The design decision was to not store a document if it was part of such consecutive series of “zeroes”. With that goal in mind, the following logic was implemented: data points are only saved if they are not zero-valued or if the last saved document under the same “variable” is not zero-valued. Discrete variables (light, door status) are only sent by the controller on a change, thus, they are always saved

During design stages, the intention was to keep the service as neutral as possible, looking

forward to further works involving the Mechanical Engineering Department to use the bases already developed. In the end, there was a necessity to include variable-specific options for averaging the frequency values reported. Storing the results every 5 seconds proved to include too much noise, so averaging them during a 1-minute period was the solution found. But, as they are calculated until the SBC stage, the averaging can only be done in that same step.

4.3.2 Machine status and frequency analysis

The frequency analysis was implemented in the same program that manages parsing for process order reasons. The goal was to extract the most dominant frequency trace in the reported spectrum for each data point. In this context, most dominant is the component with higher magnitude after the signal was transformed from time domain to frequency domain. The transformation applied to the sampled signals is the most common version of Fast Fourier Transform, the Cooley-Tukey algorithm, explained on chapter 2. According to the theory, the frequency traces are calculated as integer multiples of f_{sample}/N . Because both values were pre-established by microcontroller limitations and error-avoidance considerations as 62 kHz and 128, the frequency traces are statically obtained inside the program. The magnitude of the complex numbers, stated as $n = a + b * j$ is calculated as:

$$|n| = \sqrt{a^2 + b^2} \quad (4.1)$$

Frequency analysis process begins after deserializing the Protocol Buffers code. Then, the FFT algorithm is applied over the decoded sample array, which in turn generates an array with the phasors: every element of the array is in turn a complex value, with real and imaginary parts. Then, a magnitudes array is created applying formula 4.1. This array is inspected to find the biggest number in it. Finally, searching for the index of that value inside the given array and multiplying by the frequency trace base produces the wanted main frequency. The values are not stored right away, but added to a buffer that averages them in groups of 12.

A service to register machine general state was also programmed. Although such task can be delegated to the PaaS provider, the rate in which the state should be computed has to be at least near the rate any other variable is obtained. This would imply a result every minute, with the program running permanently (because there is no way to predict when information would change). The operation of the service would surpass the free computing load limits of Heroku. If the processing load is kept small, the SBC can execute it instead, without interrupting the parsing functions.

The service implementation runs on four single document queries. The program connects to the database at the start through Mongoose and every 60 seconds, it requests the latest stored value for spindle speed, and acceleration on each feed direction. Then, the states logic was included following in-situ observations. Every time the equipment is fully working, the spindle should be rotating. There are still moments when, although the

spindle is not moving, the machine is being operated. For examples, an initial reference for the positioning system is being set; or the cutting tool is being changed. In those moments, there is movement in the table and column axes, usually a strong one.. In this line of thought, there are 3 states, which were called *Running*, *Preparing* and *Resting*. The first one happens when spindle speed is over zero, the second one when spindle speed is zero, but any axis is strongly moving (1 m/s is the established limit for that strength) and the third one when the spindle and the axis are stopped. In the same way that door or light status are only updated when there is a change, the last machine state stored is kept in a variable and a new state is uploaded only if it differs. Because of this working method, there should be an initial state when the program starts. That was arbitrarily set to be *Resting*, and thus, every time the program starts, a machine status document with value 0 is sent to the database.

4.3.3 Watchdog

A watchdog monitoring safety conditions for the SBC operation was the last incorporated service. There were no safety routines in microcontroller because its internal architecture is too simple to introduce any further monitoring over them. The provider already manages web services, with alerts and logs when anything crashes, so the only section where supervision was both needed and viable was the gateway.

Most Linux distributions keep log files of RAM memory and processor usage, with details about every process running, in a directory called *proc*. In the watchdog, they were accessed through a NodeJS parsing utility for text files, to obtain the total processor load, total available memory, and free memory. Load processor is registered as the sum of fractional use by each core. Raspberry Pi 3 has 4 cores [49], hence, if each one is working full-time, the reported value would be 4. The situation is considered inside the program, just like memory, where the analyzed value is obtained as the fraction of free memory over available.

The Raspbian distro also includes internally run programs monitoring processor conditions, and they can be executed with the archives in the `/opt/vc/bin/vcgencmd` directory. Voltage, temperature, and clock frequency measurements were retrieved in that way. The programs were originally intended to be Linux terminal commands. To include them inside a Node program, a child process needs to be called and then redirected using the file reading functions. A 500 ms delay is implemented before collecting data, allowing OS to correctly link procedures.

The five variables are monitored every 5 seconds, and then compared to limits that were deemed execution-safe. The processor and memory load limits were set by user experience to 85%. For temperature, official producer guide [49] indicates internal ICs can withstand up to 70^o C. To provide some action space, the limit was put in 60^o C. For voltage and frequency, the information found was not clear, but it is known both values should be stable through all device operation. Usage experience showed values are fixed at 250 MHz

and 1.2 V, then a 10% range around was applied.

The alarms are sent as an email message, using SMTP protocol. A dedicated email address was created for the Raspberry with Gmail provider, to attach a valid sender to alarm messages. The automated email always includes which variable triggered the alarm and what value was reported for it. Multiple receivers can be set to the service, although for testing purposes only a personal email address was used. To avoid overpopulating the inbox with repeated messages about the same issue, a time restriction was set. An email is sent only if 3 hours have passed since the last email involving the same magnitude was sent. There is also a tracking of the alarm status at all time, so if the dangerous value was sustained over all the 3 hours, the message sent is different, emphasizing the situation has been occurring during long time.

4.4 Web based services

4.4.1 Web page main layout

The web site is entirely dedicated to organizing graph access, with some added extra details. *Express*, a library specially designed to implement HTTP routing in Node, was the tool in charge of setting a port in the host (automatically assigned by Heroku). The program itself would not load any visual content, that is done through HTML files, but it indicates which files should be loaded. Moreover, it uses a modifiable version of HTML called EJS, which can parse variables (strings, numbers, or arrays) send to it as arguments. The program can also get variables set by the user in the page using POST method and store the values if they are meant to be persistent (authentication credentials, for example), with a cookies session.

The web site uses in total 5 different views. The first one (the root route) is a login page. The project incorporates basic security options in a simple page that greets the user to the site and asks for its username and password. The page requires both GET and POST codes. For a GET request, it checks if there are already valid credentials registered in the browser, redirecting the user to the “home” page if positive. If not, it loads the regular login view with credentials spaces, as seen on figure 4.1. When the user submits data, the POST code is called to check if the credentials are valid. In that case it loads “home” page, but also stores information in the cookies session. If username or password were incorrect, the login page is reloaded with a warning about the mistake found. There is an extra button that gives access to the home page without login, managed as a simple hyperlink.

The home page includes title, description, one button for each graph available, one drop down menu for selecting the machine to monitor, one button for logout (erase user credentials from memory) and latest state displays for each of the discrete variables, as it can be seen on figure 4.2. With a GET request, the first thing the server does is to

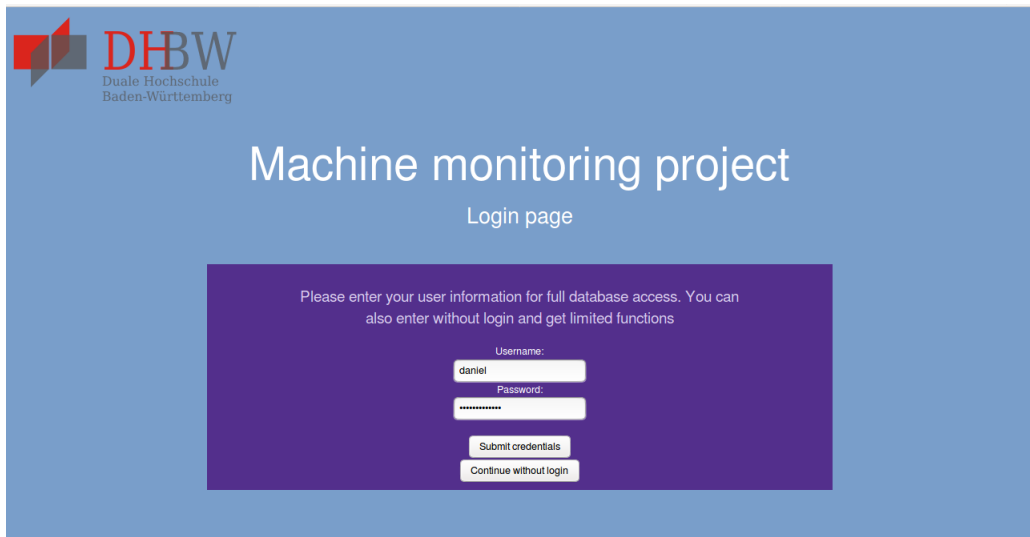


Figure 4.1: Login page for user interface. Source: Own elaboration

check the browser registers for a machine selection. This function is intended to be useful for further works. After this step, the server queries the discrete variables, looking for the last document stored. The program assigns the corresponding labels to each discrete value. The POST method covers two situations. When the logout button is pressed, it destroys the cookies session and redirects the user to the login page. When an option in the dropdown menu is selected, it changes the corresponding cookie to the new value. In the project the only option was the Hermle CNC. The graph buttons are linked inside to their respective views.

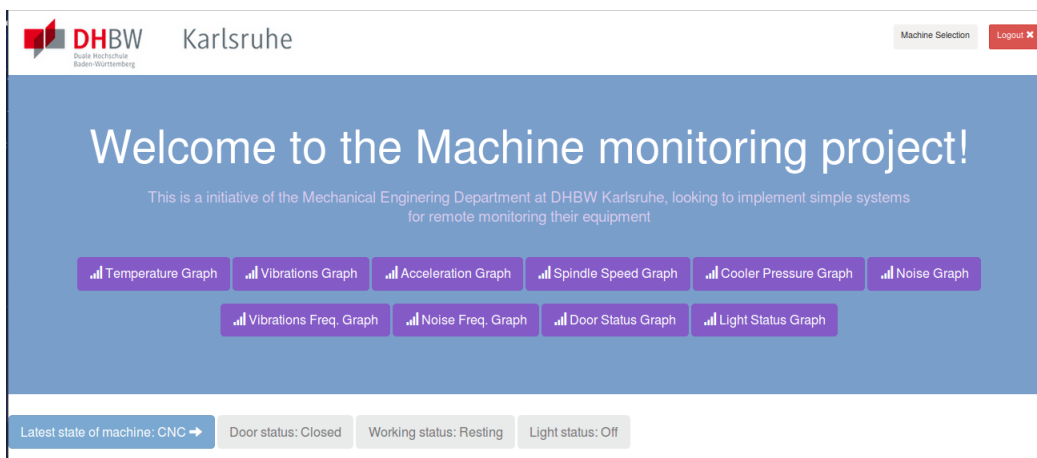


Figure 4.2: Home page for user interface. Source: Own elaboration

There is another view, which is shown when an unauthorized user tries to access protected content. It just warns about the error and have hyperlinks to the home and login pages. It can be seen on figure 4.3.

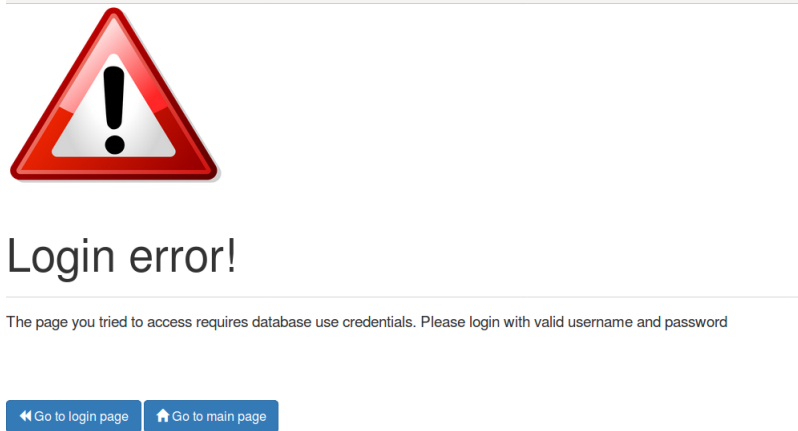


Figure 4.3: Error message for user interface. Source: Own elaboration

4.4.2 Graphics

There are two views for graph generation. One is used for continuous variables and the other one for discrete ones. There are no POST requirements in graph views, only GET processes. When a specific variable graph is requested, the program starts a query for all the documents stored under that variable name. Then, it fills out an array with all the documents timestamps. Later, it goes through each document to extract the measurement value and add it to the previous array. For the variables with multiple sensors, a pre-established order is respected when creating the arrays, using blank spaces to state the data point comes from one source only. When the array is ready, the HTML file is called and the whole collection is parsed to it, with the name of the sensors and the units of the measurements.

The Dygraphs library interprets the array as a value against time recording and creates series for each sensor if it is necessary. Dygraphs is interactive by design, so when it generates the chart, it already has zoom options, and shows series labels and specific values for points when the user hovers the mouse over them. With continuous variables, the only task done is to charge data in a Dygraphs object and set the axis labels accordingly to the units of the variables. For discrete variables, the chart is set to be a step plot, which represents values with horizontal lines, and changes with vertical ones. Then, the labels and value names shown for points are set to discrete labels, but any other intermediate value (for example, 1.2) is set to null, to ignore them during visualization.

When the data arrays are generated, the program also saves them to a .csv file using a specialized library. The .csv files are always stored in the temporary directory inside the server, an access-free space for the program inside the host memory. A button in the graph pages has always a hyperlink to the corresponding .csv file, linked to the browser as a download order. Finally, every graph page has another hyperlink to the home page. An example of the graph view is offered in figure 4.4

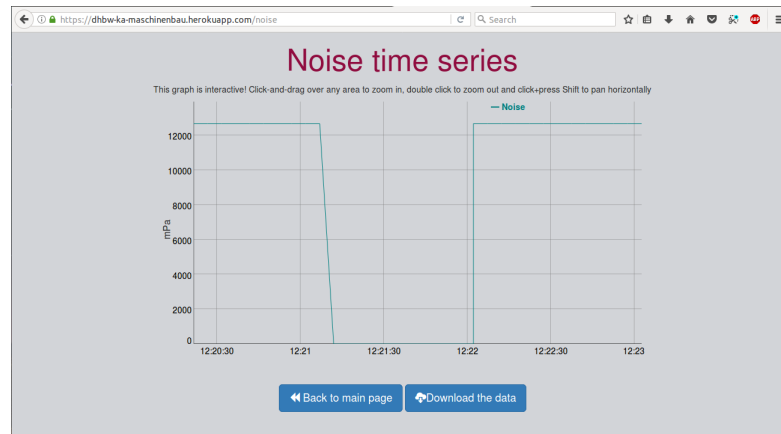


Figure 4.4: Example of graph view for user interface. Source: Own elaboration

4.4.3 Security options

User authentication protects some GUI functions, to test the existent abilities in vulnerable data protection. It was decided all continuous variable charts should be protected, letting the discrete variable graphs free to access. Hence, latest state information in home page is also left without protection. To apply this function, the database characteristics were exploited. MongoDB only lets a program to load a database if it provides a valid username and password. In the case of mLab, the registered client can create users' credentials through the provider website but needs to verify own identity first. Moreover, users can be set as read-only, avoiding possible information manipulation. Furthermore, every time a database is recalled, it is requested through secure HTTP protocol, an encrypted version of the regular protocol that adds an extra layer of safety.

Only one user with write privileges was created and its usage is restricted to parsing and machine status services on the SBC. Another public access user, with read-only status, was created as a public key for unprotected data sets. The username and password were random generated and included inside the server program. Whenever unprotected data is requested, the browser loads this public key. Finally, users with read-only privileges were created for the designer and the professors of the DHBW. When the page checks credentials validity, be it on login page or graph view, the program tries database access with the username and password stored in cookies. If the database throws an error, the combination parsed by the user is not valid.

Chapter 5

Results and analysis

5.1 Sensor calibrations

First sensor to pass calibrations was the phototransistor. The tests tried different environment illumination conditions on the component, to prove it could distinguish between room light and a directly focused lamp light. The output was registered as analog signal (in millivolts), although the sensor is used in digital configuration. The intention was to contrast the output voltage in each case with known recognition values. In Atmega 328P, “high” logical level starts on 3 V and “low” level goes up to 0.8 V. Information collected in table 5.1, evidences the output stays half volt over the “high” limit when exposed to any intensity of room illumination. When exposed to the lamp, in a dark or lit room, the response is respectively set half volt under the “low” limit. Such data is considered a strong proof the sensor selected can work without any problem as a change detecting switch on any lamp. Other tests were conducted to register the triggered interrupts amount with one light change. Between 11 and 43 interrupts were detected with only one click on the lamp, indicating a clear necessity for an anti-bounce routine.

For the door switches, the tests conducted were focused to determine maximum distance to change state. This value differs when bringing closer the two pieces and when separating them, because the sensor experiments hysteresis effects. In the first case, it is around 17 mm, in the second one it is 21 mm. The existent door gap 20 mm wide, therefore placing the movable piece in it should bring it closer enough to activate the switch. Similar procedures were executed with the distance sensor, to establish critical recognition distance. Bringing an object closer to the sensor activated the output at 14.6 cm, while taking it farther triggered a change at 15.3 cm. The 7 mm hysteresis is very important to consider, as the difference between the spindle inner and outer diameter is only 15 mm. Hence, the sensor needs careful positioning to avoid false positives during measurement. Considering all involved distances, the perpendicular extension required for the sensor should be 10.3 cm long.

The linear accelerometers were tested with gravity force applied on each side, in positive

Condition	Highest voltage (mV)	Lowest voltage (mV)	Average voltage (mV)
Inner office, low natural light, no artificial light	4897	4863	4883
Near window, low natural light, no artificial light	4819	4761	4805
Near window, low natural light, room lights on	4770	4712	4740
2 m under room lights, low natural light, room lights on	4741	4502	4603
20 cm under room lights, low natural light, room lights on	4926	4775	4828
Directly under lamp on, room lights off	210	190	196
Directly under lamp on, room lights on	244	225	234
Near window, high natural light, room lights on	3975	3540	3691

Table 5.1: Calibration tests for light sensor

and negative directions. Table 5.2 shows the initial data collected for these devices, indicating each direction has a slightly different response. The best option was to treat them separately in the control program, to keep a better precision. The signal processing includes a zero-level offset calculation (the sensors operate in this way to distinguish directions), and then a sensitivity factor. For the 3 axes, such sensitivity is around 0.14 m/s^2 per ADC unit. In chapter 3, it was recalled the sensitivity is not enough to cover machine full range. Nonetheless, most works are executed with a feed over 200 mm/min, a value that seems to be detected by the lower boundaries of the sensor.

Variable	Direction	Average ADC value	Standard Deviation (ADC units)
x	Positive	399,191	0,315
	Negative	265,610	0,359
y	Positive	396,457	0,588
	Negative	260,326	0,054
z	Positive	404,101	0,394
	Negative	269,651	0,573

Table 5.2: Measurements for linear accelerometer calibration

About noise conditions, standard deviations seem to be relatively low for all 3 directions. However, CNC acceleration range is so large, even the sensor is not capturing it completely.

Given that there is already an important information loss occurring, noise distortion should be kept at minimum. On table 5.3, results after filtering (with 6 samples) are presented. Considering sensitivity factor, standard deviation was diminished in most cases to the half, and in some of them, even the fourth part. The noise seems to be contained in less than a half ADC unit and it is considered small enough to do not affect the output. Regarding accuracy error, it only seems to affect zero level accelerations. To tackle the issue, a 0.5 bias limit was added to the control program.

Variable	Reference (m/s^2)	Standard deviation (m/s^2)	Average error (m/s^2)
Z feed	9,81	0,02	0,04
	-9,81	0,11	-0,03
	0	0,04	0,16
Y feed	9,81	0,05	0,09
	0	0,09	-0,22
	-9,81	0,01	0,03
X feed	9,81	0,06	-0,01
	-9,81	0,01	0,06
	0	0,10	0,44

Table 5.3: Results for lineal accelerometer after processing

In the case of vibration sensors, a similar approach was used. The selected model does not recognize static accelerations; consequently, it had to be tested on free fall, where the reference value is still the gravity acceleration, $9.81 m/s^2$. Ten consecutive tests with 30 measurements each were executed, and their results can be seen on table 5.4. It is evident collected data has a great variability. Even when comparing obtained averages, there seems to be no definitive agreement between tests. Standard deviation covers almost a third part of ADC capacity, which proves the sensor has too much noise to conserve precision in the output. Nevertheless, a trend was observed on the measurement series. Most of the values collected were on the ADC maximum (1023), but every test had around 5 other low values registered. Then, for calibration purposes, the 1024 output was associated to reference value, establishing a sensitivity of $0.0096 m/s^2$ per ADC unit.

The real operation of vibrations sensors is not over stable accelerations, but quickly changing signals. For that reason, blocking heavy leaps in measurement is not the best strategy, although it provides a much better approximation in stable environment. In the end, a large-sample averaging method was selected, as explained in chapter 3. The results of 1000 sample filter can be appreciated on 5.5. In general terms, zero level performance is on par with the rest of sensors, having a bias that can be easily blocked on processing, and a standard deviation around 2 ADC units. For the upper extreme, the performance is still faulty. Although the noise has a slightly smaller range, standard deviation is one fourth of the whole possible values range. Anyways, 1000-sample averaging provided the best lab results for the sensor, and the test was done without adding any analog filter, thus it was the final solution programmed in the MCU.

Number of test	Average ADC value	Standard deviation (ADC units)
1	800	276
2	920	236
3	930	180
4	831	386
5	877	340
6	778	380
7	728	391
8	824	370
9	962	171
10	870	281

Table 5.4: Measurements for vibrations sensor calibration

Reference m/s^2	Standard Deviation m/s^2	Average error m/s^2
0	0,03	-0,01
9,81	2,33	1,01

Table 5.5: Test results for vibration sensor after processing

The pressure sensor was tested with the help of a compressor and an adjustable pressure control valve. Although operative value for the CNC is 7 bar, tests were done up to 3 bar to not force the valve excessively. On figure 5.1, there is a graph produced with the calibration data. Correlation parameter R shows a very linear response from this sensor, as expected from datasheet details. The sensitivity is also very similar to producer suggested values. Noise conditions are relatively low for small pressures, but they show a tendency to grow for larger reference values. Real measurements are expected to be at least twice the test ones, hence, some problems could arise if no further processing is added. With a simple 6 measurement averaging, the standard deviation diminished between 2 and 5 times, according to tables 5.6 and 5.7. Compared to the expected 0.1 bar resolution, the variance is already located 2 orders of magnitude lower, enough to assure no visible effects on the output. Accuracy error is in the worst case the same as required resolution, and it was ignored.

Microphone output is supposed to be lineal when measuring sound pressure on the air. However, in acoustic environments the decibel-registered SPL is the most common way to report magnitudes. For testing purposes, environment noise was measured through a PC application using SPL conventions and then converted to pascal units through the formula:

$$SPL_{dB} = 20 \log \frac{P_1}{20 \mu Pa} \quad (5.1)$$

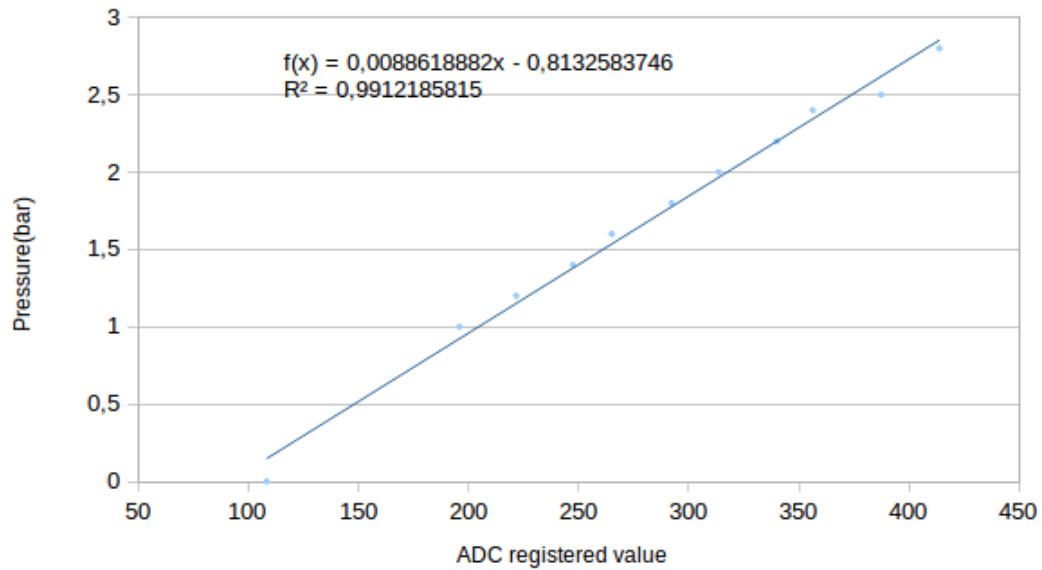


Figure 5.1: Calibration curve for pressure sensor. Source: Own elaboration

Reference (bar)	Standard deviation (ADC units)
0	1,54
2,4	1,70
2,8	2,85
2,5	1,48
2,2	1,66
2	1,53
1,8	1,44
1,6	1,75
1,4	1,39
1,2	1,51
1	1,61

Table 5.6: Measurements in pressure sensor calibration

Figure 5.2 shows the microphone has a linearity similar to the pressure sensor, which in turn proves it is good enough for project goals. The test values used for calibration went from 20 dB (the average background noise in a quiet office) to 96 dB (expected noise

Reference value (bar)	Standard Deviation(bar)	Average error(bar)
0	0,0043	-0,0955
3	0,0044	-0,1146
2	0,0041	-0,0097

Table 5.7: Test results for pressure sensor after processing

for industrial environment). Considering the logarithmic nature of sound perception, the sensitivity shown on the graph can describe the interest range (the area between 60 and 90 dB, typical number in heavy industrial activity) with a relatively good resolution. Also, the test showed the microphone output gets saturated over 100 dB, confirmed upper limit for its operation.

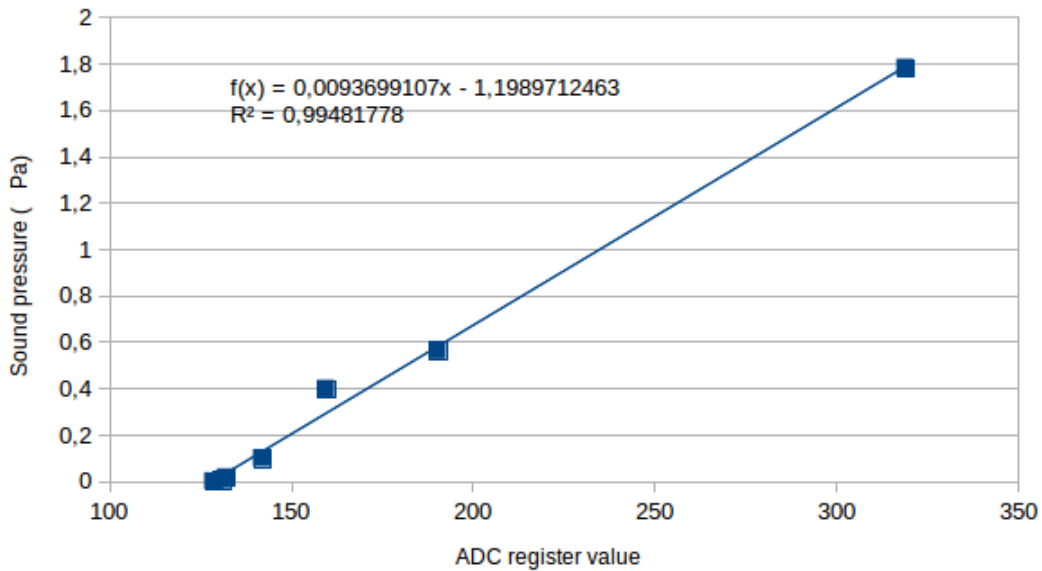


Figure 5.2: Calibration curve for microphone. Source: Own elaboration

Noise incidence in audio records is shown on table 5.8. The first set of data was used for the response curve. Although remarked as “before processing”, the measurements reported were in fact RMS numbers of the signals, because the nature of the audio waves require this step before comparing measurements. The table shows the noise variance was rather ample. With sensitivity conversion, the noise addition would represent a precision loss of 0.252 Pa, around half of the expected lower limit. After applying 6-measurement averaging, the noise deviation was reduced in one order of magnitude. For practical reasons, the same SPL values could not be repeated in the second set of data. Still, variance, even for smaller values, is small enough to not affect the output more than 1 dB. The impact is barely noticeable in real situations, so the proposed filter was accepted.

Finally, tests were executed to prove FFT algorithm performance. To do so, a web-based application for audio tone generation was used with the microphone to catch sinusoidal sound waves on different frequencies. The app was verified in first place connecting the microphone to an oscilloscope, where perfect correspondence between reported and measured frequencies was confirmed. Then, the microphone signal was sampled and analyzed through the program mentioned in chapter 4. Different combinations of sample rate and total sample numbers were examined. The table 5.9 shows the two with better experimental fits. Standard deviation for a specific measurement is either null or a very large number. In this case, the wrong measurements consisted in frequency analysis reports where the main component was 0 Hz. Because the reference levels were in the order of kilo-

Condition	Reference value(Pa)	Standard deviation(ADC units)
Before processing	0,0014	21,42
	0,0178	15,11
	0,0063	18,99
	0,1002	16,67
	0,3990	27,50
	0,5637	28,61
After processing	0,0002	1,11
	0,0008	0,93
	0,0016	1,22
	0,0056	0,78
	0,0032	3,73
	0,0050	3,38
	0,0080	3,32
	0,0112	2,67
	0,02	3,25

Table 5.8: Comparison of microphone performance

hertz, such errors have a deep performance impact. The situation motivated the addition of averaging in the frequency analysis. Although not shown in this inform, moving-average with around 5 samples was enough to avoid the zero-valued measurements, because they tended to occur rather isolated.

About accuracy error, it is necessary to consider that FFT only generates discrete frequency spectrum and obtaining the exact value of a main frequency is highly improbable. In the table, it is possible to see when no strong deviation is involved, the error is two or three orders of magnitude smaller than the reference. Under those conditions, the error is justified by the discretization possibilities and can be ignored. The zero-valued measures obviously add a heavy amount of error, because for them, the deviation is exactly the equal to reference. Just like standard deviation, filtering solved this problem. Finally, there is no clear difference in performance between the two sample settings used, which means the algorithm is barely affected by operative parameter alteration.

5.2 Bluetooth and Protocol Buffers

To test serialization protocol working state, set-up data was sent with it through the pretended communication channel. The data was manually loaded in the microcontroller using the message structures previously designed, sent to the Bluetooth module through UART, received in the SBC and decoded to review the integrity of the message. Errors were evaluated on the numeric elements of the messages.

As it can be seen on table 5.10, the trials were executed with different baud rates on the

Conditions	Reference value (Hz)	Standard deviation(Hz)	Average error(Hz)
35 kHz sample rate, 512 samples	200	30,57	56,45
	300	0	26,56
	400	0	-10,16
	500	0	89,84
	1000	0	42,97
	2000	0	85,94
	5000	1099,43	-467,10
	10000	1099,43	-466,60
	20000	66,90	401,37
60 kHz sample rate, 128 samples	500	0	46,88
	900	0	-6,25
	1200	673,70	341,45
	1500	509,27	355,26
	2000	898,27	855,26
	4000	935,59	136,51
	8000	3817,63	3134,87
	10000	5056,72	4228,62
	15000	0	46,88
20000	0	515,62	

Table 5.9: Results for frequency analysis tests

UART port, to asses which one was more appropriate for the system. Communication speed is vital to avoid interrupt collisions in the microcontroller program. Therefore, the goal was to use the fastest rate that did not present data losses. For each test, 20 messages with random assigned numerical values were sent. The Protobufs implementation on NodeJS throws run-time errors were a message sent is impossible to decode. Hence, the “received” set of data only counts messages that were both successfully received on the Raspberry Bluetooth port and decoded back in the program dedicated to it.

Type of message	Baud rate	Received	Average error
One-value, integer	921600	0	N.A
	460800	20	0
	230400	20	0
One-value, floating point	921600	0	N.A.
	460800	17	0,000002
	230400	20	0,000002
Array of values	460800	0	N.A
	230400	11	0
	115200	20	0

Table 5.10: Test results for communication with Bluetooth and Protobufs

Empirical data proves raising bytes quantity increases communication error. To include all operative conditions existing for the project, the 115200-baud rate is the fastest option, and correspondingly it was selected for the MCU program. In all tests, integer values did not present any deviation when the message was received. Distortion on binary information, when present, causes changes not only on the numerical values themselves, but in the whole serialized code, causing the whole message to be unreadable. Therefore, by guaranteeing that messages are getting decoded, the integrity of the data is mostly secured. In the case of floating point numbers, the small errors perceived are due to conversions in a different environment during decoding. The errors found are 5 orders of magnitude smaller than the desired resolution, so is safe to ignore them. The examination was repeated with the selected baud rate but positioning the SBC about 10 meters from the MCU, inside the CNC workshop, to detect noise susceptibility in the channel. The same 100% efficiency was found, evidencing no trouble with environmental noise.

Measurements were then made to measure the time communication protocols would take to be executed. The initial design goal was to keep it under a 10% of the timer period. This should prevent interrupt unawareness by control program execution, especially relevant with the spindle speed routine. With timers set every second, the execution for the longest sections should not exceed 100 ms. The figure 5.3 displays the serialization of a full 128 sample message and its transmission. The process can take almost 70 ms, which is still under the desired limits. The only other process that would take more time would be the temperature sensor conversions (90 ms according to datasheet [40]) but this operation runs on the sensors themselves, not the MCU.

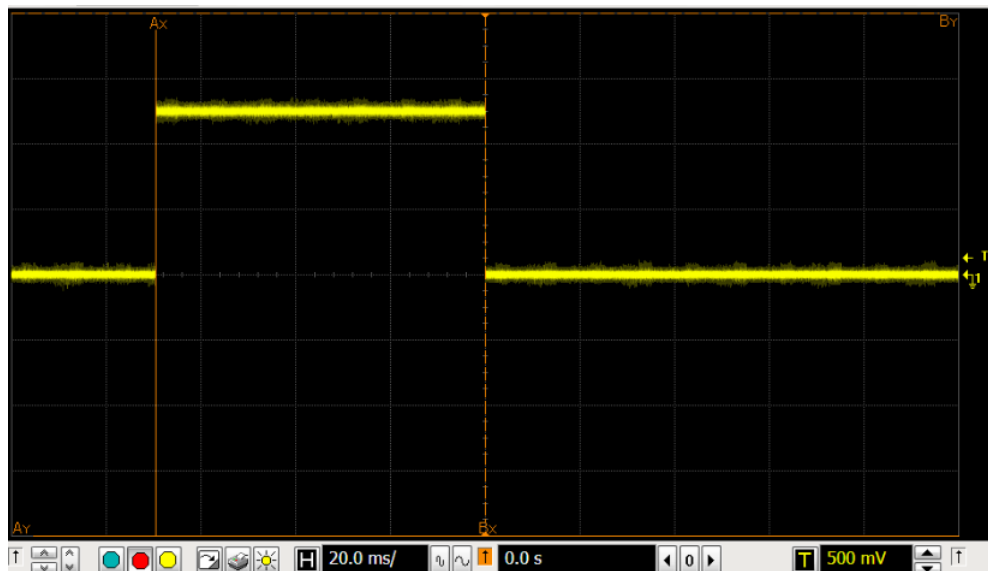


Figure 5.3: Time required for sending a full array-type message. Source: Own elaboration

The final step for MCU settings examination, the ADC real sample rate was determined. Following datasheet instructions, a 16 pre-scaler was selected for the unit, with the intention to operate on a 70 kHz rate. Figure 5.4 exposes that in real conditions, the frequency is more close to 60 kHz. The magnitude is still good enough to cover the desired ranges

following the Nyquist theorem, but the frequency analysis program must be adapted to include this detail.

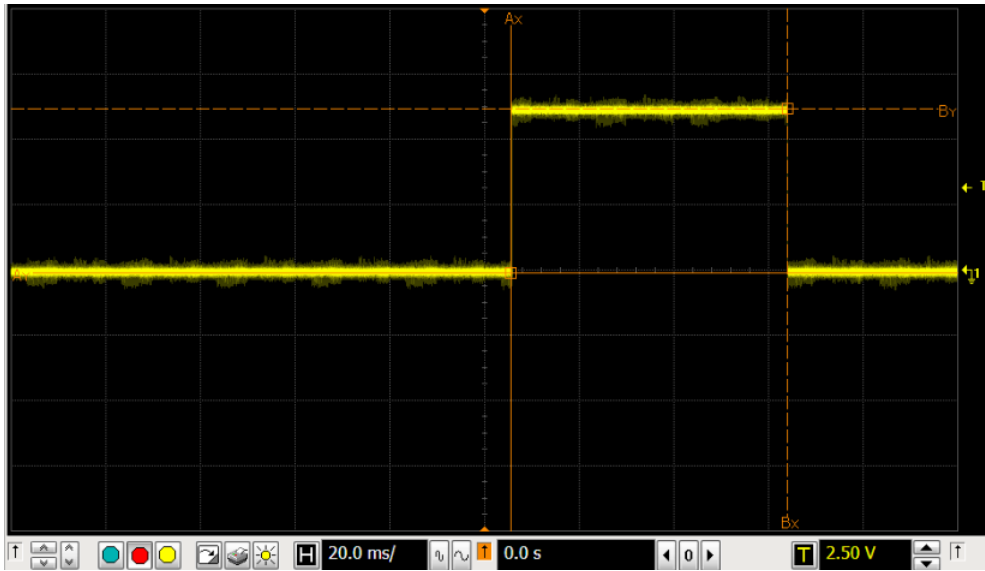


Figure 5.4: Time required for a single ADC conversion. Source: Own elaboration

5.3 Web page with made-up data

To analyze Internet communications side functional abilities without sensors circuit intervention, pre-defined data was generated from an independent MCU (for easiness, an Arduino card was used) and transmitted to the cloud using the designed information bus (UART-Bluetooth-SBC-Mongo database-web server). The data was set up with the one-value message type, including all the “required” spaces. All the safety and display options, according to chapter 4, were included in the web site operation.

The first data set uploaded was labeled as “acceleration”. It consisted of a random series of values between 0 and 100, sent in regular intervals of time (5 seconds). To prove the uploading algorithm in presence of null-valued messages, the magnitudes sent during a 50 seconds period, in the middle of the test, were purposely set to zero. The image 5.5 was obtained directly from the corresponding web page.

The data points keep a 5 seconds distance between them, correctly stating no data loss in the process. Also, the values on the graph stay under 100, so there was no heavy distortion in them. The zero-valued period, although not clearly showed in the figure, was saved as only two points (at the beginning and the end), so the test proved no extra data is uploaded when the system is “inactive”.

To clearly asses the results distortion, a new test was executed with data points strictly fit to a function. In this case, data was labeled “Vibrations”, and the function chosen was a quadratic curve, specifically $f(t) = 0.8t - 0.0128t^2 + 4$, with t being time in seconds. The

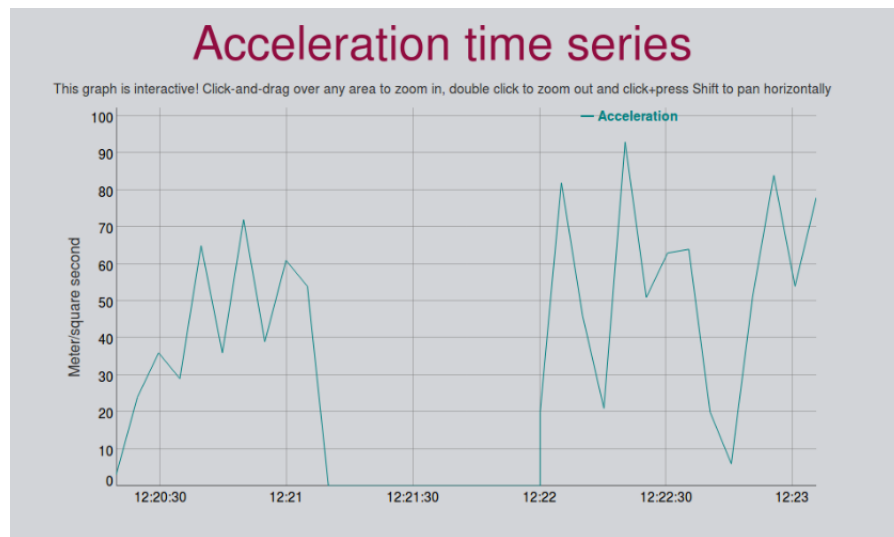


Figure 5.5: Results obtained for made-up data: “Acceleration”. Source: Own elaboration

sending periods were still set to 5 seconds and the 50-second null space was preserved. Results can be appreciated on figure 5.6. The null period was also saved as two points, as expected. The general shape of the graph follows smoothly the shape of a quadratic function, and the individual points are shown in the positions predicted by the given function. In that sense, no important distortion was detected.

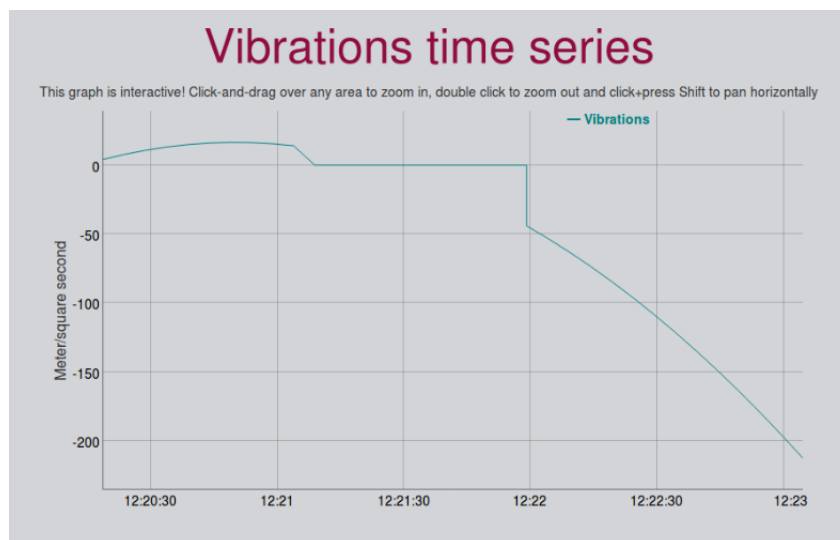


Figure 5.6: Results obtained for made-up data: “Vibrations”. Source: Own elaboration

Finally, to prove graphic library ability to handle multiple series in one chart, messages with 3 different *sensor* spaces were sent, all under a “Temperature” *variable* label. All data sets were created with a sine function but including a 90° phase shift between them. Figure 5.7 shows the graph created with these parameters. The 3 series can be easily distinguished, and they conserve the sinusoidal shape expected from them. There is small shape distortion in the initial portion of one graph, caused by an initial value equal to zero, which was ignored by the upload algorithm. Still, it is considered a successful attempt.

With these tests, the login, logout, machine selection and error page functions of the web site were also tested, reporting a positive performance in all those areas.

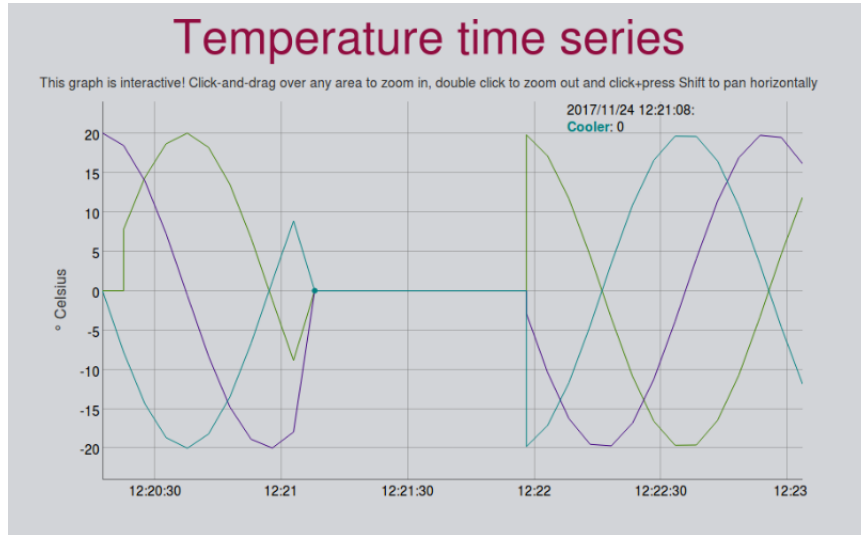


Figure 5.7: Results obtained for made-up data: “Temperature”. Source: Own elaboration

5.4 Web page with real-time data

During sensor installation, a first run with the whole system working was executed. Preliminary versions of the MCU and SBC programs were used. For that reason, some of the filters were still not optimized and the frequency averaging was not even implemented. Also, no “Machine status” was included, as the corresponding service was still incomplete. The data was collected over the space of about 2 hours. During this time, the real-time status function for light and door state was checked, resulting in an operation without errors. Temperature and pressure measurements worked with almost no noise. Vibrations were perceived slightly noisier, but still useful. The main issues happened with the acoustics, acceleration, and spindle speed measurements.

Acoustics monitoring, seen on figure 5.8, has too much variance to be considered representative or useful. Although is the workshop environment is relatively loud and noisy, the microphone measurements should at least point out some difference related to the sensor placement, something not clearly present in the graph. Consequently, too much external interference is still reaching the MCU. To attack the situation microphone filter were strengthened, both in the analog side, changing the capacitor (it was first set as 50 nF, and then raised to 500 nF); and the digital side, raising the sample number from 6 to 12.

The acceleration measurements, shown in figure 5.9, also presented a heavy output variance. The situation is even worse considering most of the time, the real accelerations present in the CNC machine should be null. Moreover, the average reported value is near

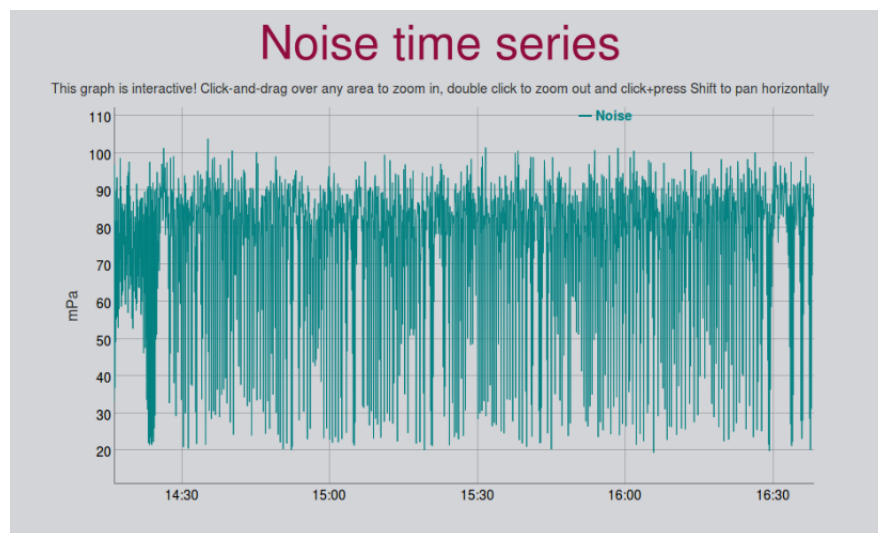


Figure 5.8: Results obtained for real data on 1st try: Noise. Source: Own elaboration

1 m/s^2 , which is the identified limit for tool change fast movements. A closer inspection, confirms all Y direction measurements are over the zero line, while the X direction ones are under it. This characteristic suggested the root problem was faulty calibration. The specific sensor used for calibration process was not any of the installed ones, resulting in differences on the reference zero-level offset used. A quick recalibration was done in-situ to define the real zero outputs. After it, the averaging filter was also changed from the original number of samples to 60, to avoid future noise induced errors.

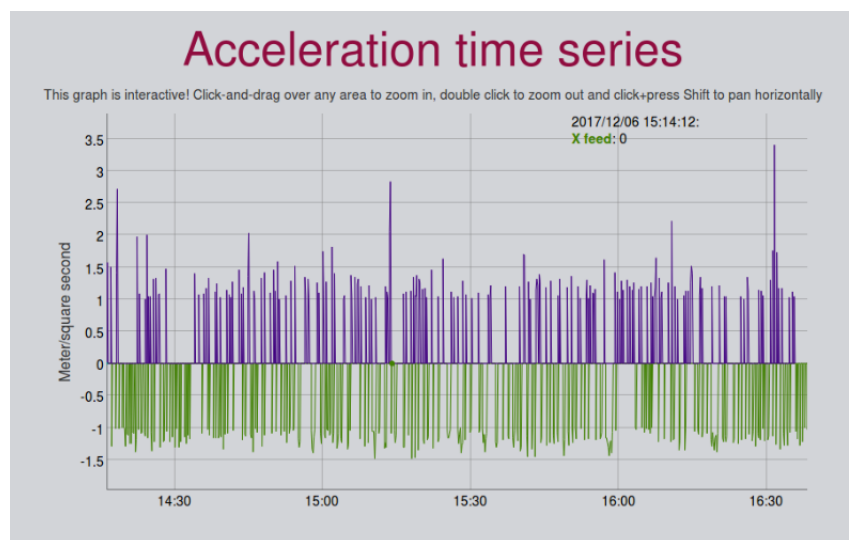


Figure 5.9: Results obtained for real data on 1st try: Acceleration. Source: Own elaboration

The main concern for spindle speed monitoring, although there were precision errors issues, was accuracy. As it can be noted in figure 5.10, the values obtained from the sensor and the counting algorithm in the MCU never surpassed 35 RPM. But the real spindle speed was set during the whole process to 400 RPM. The bias was too big to be ignored, and the results were simply not representative of machine state. During consecutive new

tests exclusively done with the spindle speed variable, using progressively higher RPM values, it was discovered the measurements were perfectly accurate for magnitudes under 100 RPM. Nonetheless, over this value, the algorithm produced results proportionally smaller. For speeds around 600 RPM and over, the system reported only zeros. This behavior suggested a problem with the interrupt handling. Although in theory no issue should be expected in the MCU, the evidence was enough to try another approach. The solution implemented was to move the counting algorithm to the SBC as another service, considering this component has a far superior computing power. The program was written in NodeJS, and directly loads the measurements to the database.

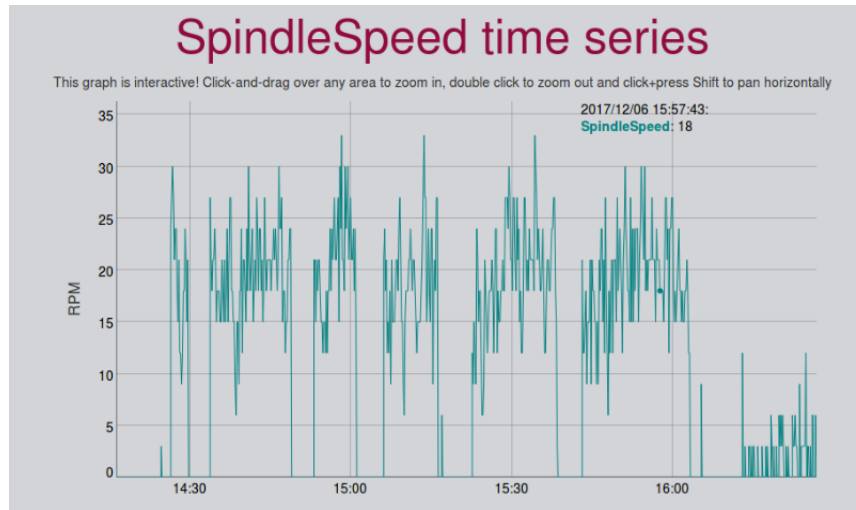


Figure 5.10: Results obtained for real data on 1st try: Spindle speed. Source: Own elaboration

Following the modifications, the database was cleared and the whole system was put to work again, to record data over a larger period. The intention was to use it for at least two weeks. The data shown in the following figures was collected between December 9th and December 21th, 2017, and the machine was actively used on December 11th and December 12th.

Figure 5.11 refers spindle speed results. The chart was zoomed in to show only areas where non-zero values were present. Operative periods in two different days can be distinguished, rendering the information useful. The measurements seem less noisy than first tests. However, the variation range is equal in relative terms. The accuracy improved in significant terms. For a reported value of 600 RPM, the experimental results oscillated between 400 and 600 RPM, with a maximum error of 33.12% and an average of 16.10%, while in the first test the average error was around 92%. Still, present variability is too high for what is supposed to be a constant value. Reminding this variable is measured without any filtering, the usage of post processing could be useful.

Figure 5.12 show collected data for cooler subsystem pressure.

A comparison between this graph and the previous one indicates working peaks occurrence at similar times. The type of piece machined during tests generated large amounts of chip and produced localized heat, thus, air was switched on almost the whole time the

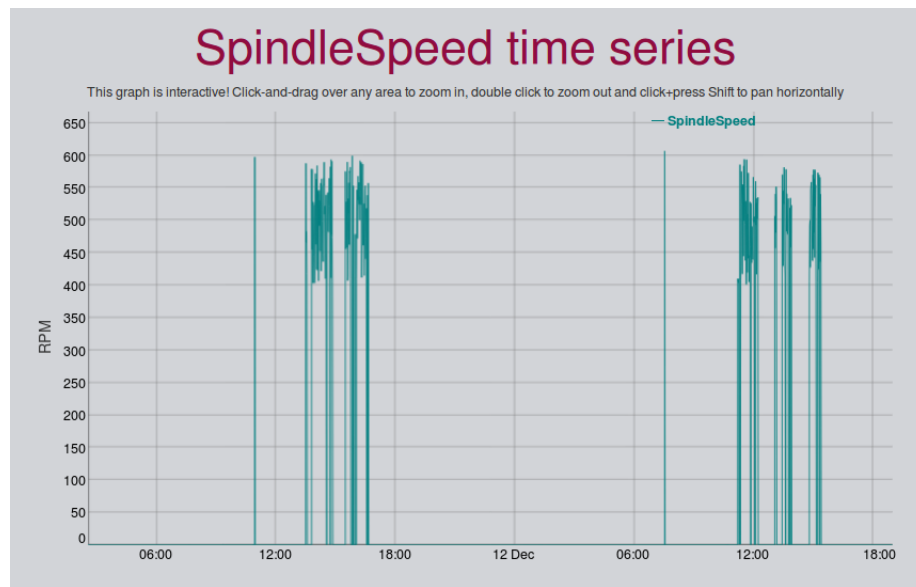


Figure 5.11: Operational data: Spindle-speed. Source: Own elaboration

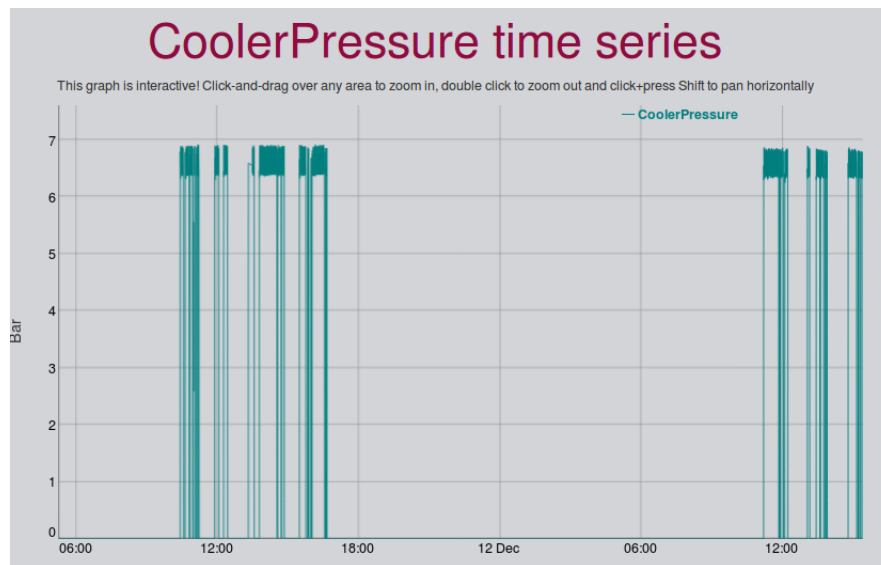


Figure 5.12: Operational data: Pressure. Source: Own elaboration

CNC was under operation. For that reason, no analysis on air cooling effects over other variables can be made, as there are no operative points without air pressure presence. The measurements collected have actually one of the best performances in the whole project, with a 0.8 bar standard deviation and an average value (6.46 bar) valid in comparison to real conditions, considering maximum bomb pressure is 7 bar, but losses in control valves and hoses normally exist. In general terms, the pressure measurements comply with system expectations.

Figure 5.13 includes the results for registered temperatures. The sensor accuracy seems right, because most values in the graphic are around 22^o Celsius, an accepted value for an inner-building spaces as the workshop. Moreover, the 3 series follow the same pattern

in general line, with a difference between them present in the form of an offset. Such offset seems to correlate with the distance of the given sensor to the working area. In that sense, the sensor placed in the cooler system has higher temperatures than the table one, given the compressed air effects do not affect the hose surface.

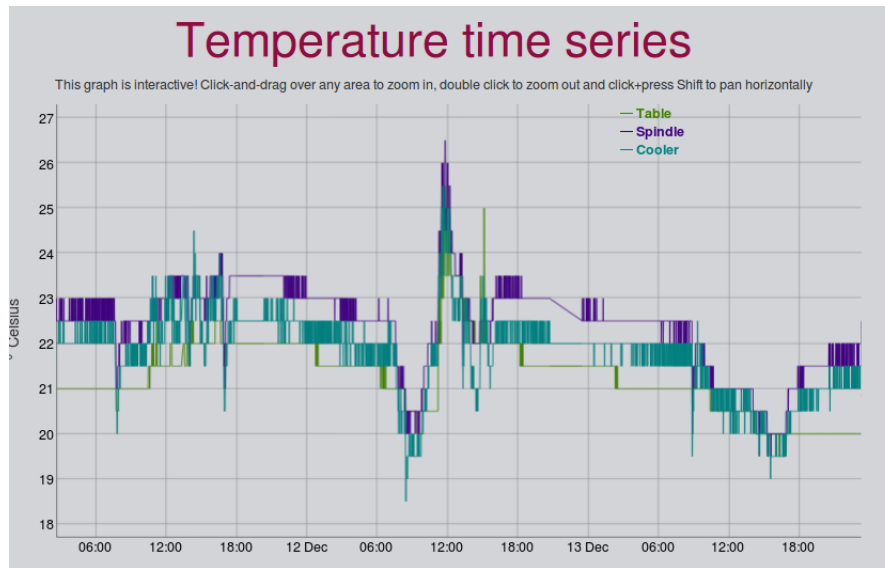


Figure 5.13: Operational data: Temperature. Source: Own elaboration

The general behavior of the data series has a slight rise on December 11th and noticeable peak on December 12th, coinciding with the operative times. There are also minimum points on the morning of December 12th and afternoon of December 13th. Although the origin of these other variations remains unknown, one possible explanation is they correspond to maintenance labors to clean the machine after heavy work or prepare the area for it. About noise, the standard deviation for the three sensors is around 0.7° C, which is near the minimal sensor resolution (0.5° C). Table sensor has large periods with constant output and reacted slower to changes.; cooler sensor has the most unstable output. Such differences could indicate measurements variances are caused in great part by real environment circumstances.

The vibrations magnitude chart on figure 5.14 specifies big peaks occurred on December 12th during the active usage of the CNC. The data values are noticeable higher for the sensor located on the table, on almost 1:2 ratio when compared to the column sensor. Also, there is a clear difference between both sensors in noise conditions. While table sensor kept its output in zero for most of the time, only showing results during operative time (the mentioned peak and smaller results on December 11th), the spindle column sensor had avariable output around $0.02 m/s^2$, with very small variance, during rest time. This noise value is two orders of magnitude smaller than the peak performance registered for the same sensor, hence it can be discarded as zero-level bias, like it was done with another variables processing. The main effect of this noisy background, though, can be appreciated on figure 5.15, which shows the respective frequencies reported for both sensors. While the table sensor only shows values during the commented operative periods, the column

sensor generated frequency analysis results all the time, mainly reporting noise frequency. For that reason, no clear pattern is found in the series. Anyways, the data collected suggests a correlation between spindle speed, vibrations, and temperature. Chapter 2 explains this could cause an impact over tool health and pieces quality. Also, it appears December 12th working load was more demanding for the machine than other days.

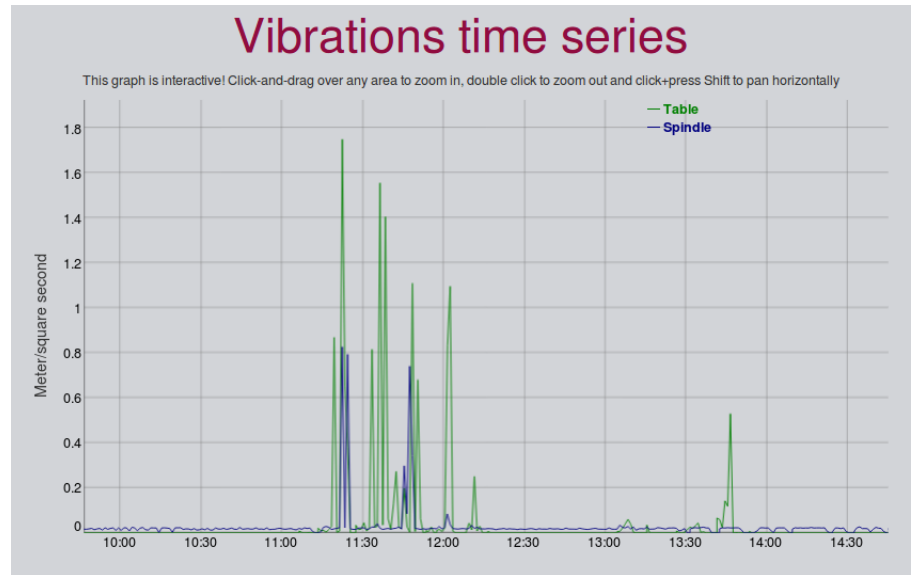


Figure 5.14: Operational data: Vibrations. Source: Own elaboration

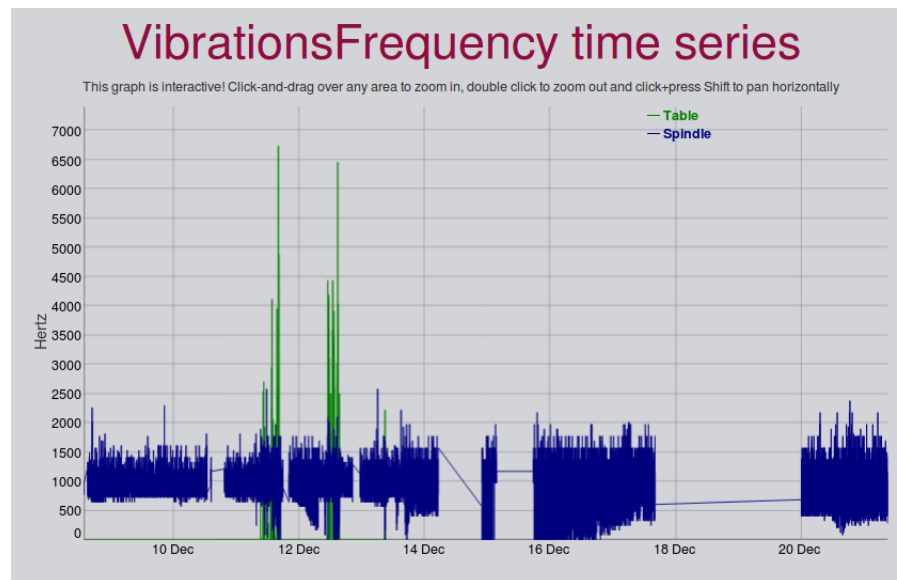


Figure 5.15: Operational data: Vibrations frequency. Source: Own elaboration

In the linear acceleration case, the figure 5.16 describes the behavior found. At first glance, the signals recorded are still very noisy and with a significant variance. The noise variability extends over the expected range for measurements (from -1.5 to 1.5 m/s^2), and the standard deviations in each series represent a 30% of that same quantity, so no clear distinction can be done between valuable results and noise. Compared to the first

test, however, the measurements are distributed around the zero level, proving the re-calibration was effective. A closer look to the 3 different series in the graph also remarks the differences between them. The Y direction feed actually has the expected output for the sensor, with a few momentary peaks corresponding to sudden movements and a zero level most of the time. This direction was being measured with a different sensor, because the column controls that feed. Z direction accelerations, by other side, seem to lose their calibration after December 12th, and all values reported after that date are negative. As the initial values did not show that tendency, it was clear that some misalignment happened with the sensor position. Therefore, a trouble during the installation the table sensor was considered. An inspection showed the sensor lost its adhesion to table surface, which partially explains the errors registered.

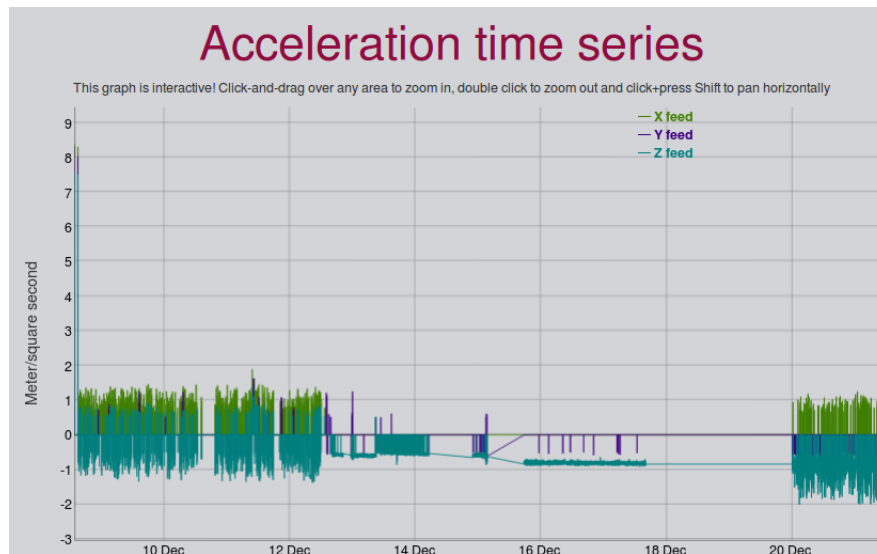


Figure 5.16: Operational data: Acceleration. Source: Own elaboration

Noise strongly diminished between December 13th and 20th, when the machine was barely used. Reminding sensor sensitivity was high, this event could mean much of the noise caught was generated by vibrations and external actions over the machine. In any way, the acceleration data would require better filters to generate useful information. The side effect to heavy noise in acceleration monitoring is that “MachineStatus” variable had a large number of erroneous results. For that reason, it was not even considered into analysis.

About acoustics measurement, figures 5.17 and 5.18 show noise captured by the microphone is still too ample to be limited by proposed processing. In consequence, the collected data has no recognizable patterns and no relation can be easily established with the rest of variables. There were no specific values expected for the sensors, but noise variance can be compared with the mean measurement reported. The average sound pressure detected is 76.64 mPa, while the biggest instant changes between adjacent points can be as high as 71.41 mPa. Having both quantities so close to each other proves the low reliability these results have. In the frequency domain, the standard deviation found is 500 Hz, and the average frequency reported is 1484.4. Although slightly better, the reported

amounts represent a 30% variability. The only relatable information can be seen on figure 5.18, where the peak frequency happens during recognized operation time. Bearing in mind averaging with higher sample numbers, using other statistical operators (median, for example) and different analog filters were proven without any visible advantage in performance, one possible solution would be to use low-pass higher order digital filters (Chebysev, Butterworth, etc). The amount of variance, also suggests using a different sensor may be necessary, as the electret seems to have a very open field of measure. A microphone with a more focused range could be useful for the intended function.

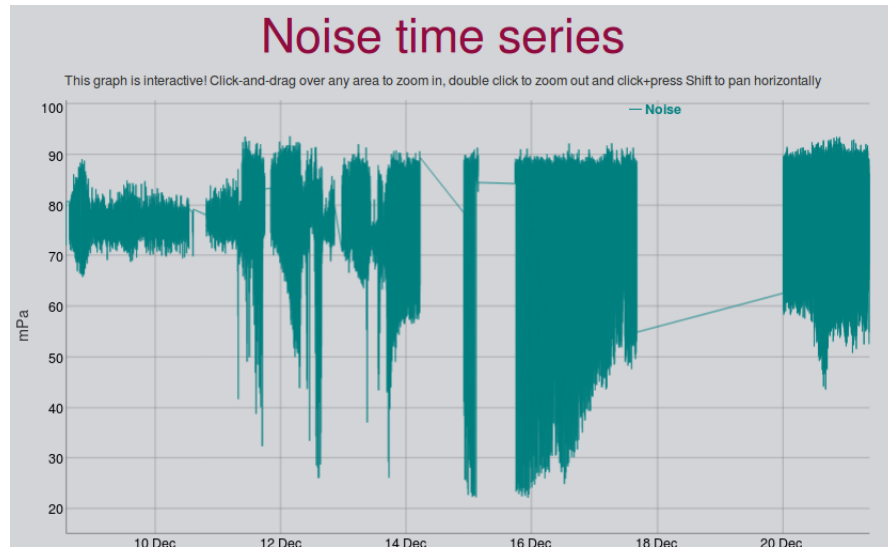


Figure 5.17: Operational data: Acoustics. Source: Own elaboration

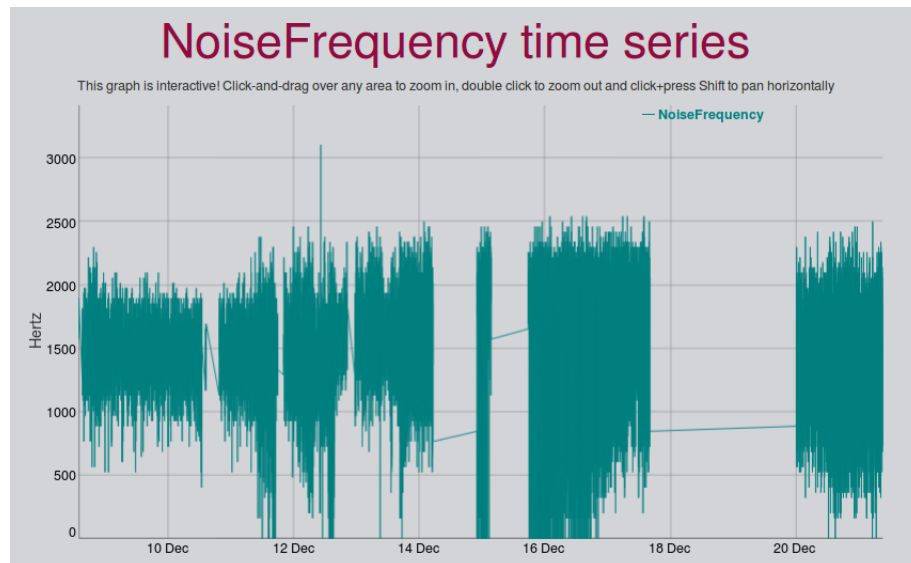


Figure 5.18: Operational data: Acoustics frequency. Source: Own elaboration

The other collected variables, data and light status, are not shown here as they do not require further analysis. Door status presents periods with high change amounts before and after operating the CNC, as would be expected of regular work preparations and

posterior cleansing. About light status, its behavior is the same as the door, consistent with its aiding function. The light also has a period of multiple changes on December 21, when intense maintenance labors were executed.

Looking to the graphs with constantly varying values (specially 5.17, 5.18 and 5.13), “jumps” between data points can be identified. The missing information in those periods was caused by a faulty Internet connection on the SBC, which effectively stopped data uploads. Although the original plan was to use wired LAN access, the building limitations made necessary the application of a Wi-Fi network. The wireless link proved to be unstable, though. Whenever the signal was lost, the Raspberry required a reboot to get back to work, but no precautions were taken to alert about it. In the end, some of the reboots were executed even 2 days after signal was lost.

5.5 Analysis of operative data

Data for spindle speed, temperature and vibrations presented hints to a mutual correlation. To assess if this possibility is statistically significant, tests with collected measurements full sets for each variable were run. Statistical analysis software Minitab was chosen as testing environment. Two different parameters were studied. The first, Pearson correlation coefficient, indicates the degree of strictly lineal dependency between magnitudes in a scale from -1 to 1, where zero is no absolute presence of correlation and 1 is perfect correlation. The second one, the Spearman correlation coefficient, is used for monotonic relationships, where the change rate can be different for variables involved, as long as both follow the same direction. Therefore, Spearman coefficient also includes non-linear relations [50].

The first test studied the correlation between spindle speed and vibrations. Figures 5.19 and 5.20 show the analysis coefficients found for table and column sensors. Even when they seem to follow a very similar pattern, the noise presence in the spindle sensor affected in a great manner the statistical characteristics of the measurements, so its coefficient related to table sensor register is lower than 0.5, indicating a weak correlation. The noise also affected the correlation perception of with the spindle speed, and the coefficient in this case is even smaller. Although correlation seems to be poor in all cases, the p-value does not allow to categorically reject the possibility of some influence between variables. For the table sensor, the coefficient obtained with Spearman test is over 0.7, a high quantity if noise and environmental conditions are considered. Pearson coefficient is lower than Spearman one, so the idea of a strictly linear relationship is discarded. In summary, there is strong statistical evidence a monotonic correlation between the speed and the vibrations exists, at least on the table.

For temperatures, the results can be reviewed on figures 5.21 and 5.22. In this case, as suggested by the graphs, the relation between the different temperature sensors outputs is the highest one obtained for any analysis in the project. Such situation indicates the

Correlation: SpindleSpeed, Vibrations Spindle, Vibrations Table

	SpindleSpeed	Vibrations Spind
Vibrations Spind	0.113 0.000	
Vibrations Table	0.221 0.000	0.316 0.000

Cell Contents: Pearson correlation
P-Value

Figure 5.19: Correlation analysis for vibrations: Pearson. Source: Own elaboration**Spearman Rho: SpindleSpeed, Vibrations Spindle, Vibrations Table**

	SpindleSpeed	Vibrations Spind
Vibrations Spind	0.091 0.000	
Vibrations Table	0.708 0.000	0.110 0.000

Cell Contents: Spearman rho
P-Value

Figure 5.20: Correlation analysis for vibrations: Spearman. Source: Own elaboration

three sensors were affected by the same environmental effects, and the information collected by them is representative of CNC conditions. Then, comparing their respective coefficients related to spindle speed, the response is much less related, with all the coefficients staying under 0.3. Unlike vibrations, that are closely associated to the rotations by the mechanical structures, many factors easily influence temperature. The evidence shows their correlation is better than spindle vibration sensor, though. In that sense, it is possible to confirm spindle action has a little, but noticeable influence in temperature state. As every Pearson coefficient found was higher than its Spearman counterpart, the influence must be considered linear.

Finally, another test was done with the vibration frequency reported values. The output of such analysis is on figures 5.23 and 5.24. The great impact noise had on frequency in the spindle sensor caused loss of valuable information to establish relationships with other variables; under the present condition, the information kept lost its usefulness. Hence, in all comparisons made with this sensor data, the p-value was big enough to discard any correlation. In the opposite side, the table vibration sensor presents a considerably strong relation to spindle speed. Furthermore, the correlation between frequency and spindle speed appears to be monotonic too. The results reinforce the proposition that both variables are statistically related.

Looking at figure 5.15, the peak frequencies for vibrations have values around 6.5 kHz. The author of [9] found peak vibrations for both regular and worn out tools in levels under

Correlation: Table Temperature, Spindle Temperature, Cooler Temperature, Spindle Speed

	Table Temperatur	Spindle Temperat	Cooler Temperatu
Spindle Temperat	0.844 0.000		
Cooler Temperatu	0.805 0.000	0.898 0.000	
Spindle Speed	0.381 0.000	0.256 0.000	0.341 0.000

Cell Contents: Pearson correlation
| P-Value

Figure 5.21: Correlation analysis for temperatures: Pearson. Source: Own elaboration**Spearman Rho: Table Temperature, Spindle Temperature, Cooler Temperature, Spindle Speed**

	Table Temperatur	Spindle Temperat	Cooler Temperatu
Spindle Temperat	0.873 0.000		
Cooler Temperatu	0.835 0.000	0.880 0.000	
Spindle Speed	0.258 0.000	0.199 0.000	0.247 0.000

Cell Contents: Spearman rho
| P-Value

Figure 5.22: Correlation analysis for temperatures: Spearman. Source: Own elaboration

100 Hz, but as [12] explains, structure-based emissions can be found also in frequencies over the 4 kHz. There are strong possibilities the reported high frequency vibrations are related to tool quality degradation, according to [6]. More measurements including other work conditions would be necessary to clarify the panorama.

Correlation: Vib. Freq. Table, Vib.Freq. Spindle, Spindle Speed

	Vib. Freq. Table	Vib.Freq. Spindl
Vib.Freq. Spindl	-0.021 0.018	
Spindle Speed	0.518 0.000	-0.013 0.130

Cell Contents: Pearson correlation
| P-Value

Figure 5.23: Correlation analysis for vibrations frequency: Pearson. Source: Own elaboration

Spearman Rho: Vib. Freq. Table, Vib.Freq. Spindle, Spindle Speed

	Vib. Freq. Table	Vib.Freq. Spindl
Vib.Freq. Spindl	0.012	0.155
Spindle Speed	0.715	0.003
	0.000	0.773

Cell Contents: Spearman rho
P-Value

Figure 5.24: Correlation analysis for vibrations: Spearman. Source: Own elaboration

5.6 Watchdog tests

Before implementing watchdog final version on the system, a quick test to prove the algorithm performance was run. To make sure that the alarms were responding, it was necessary to trigger them on purpose, a task that could be easily achieved changing the set limits to regular operative levels. Core temperature was the selected condition, because under regular operation, it maintains a stable level (around 40^o C) compared to memory and processor loads, that can easily fluctuate; but it has small variance amounts, unlike core voltage and frequency, that are basically constant. The limitation for temperature was then set to 30^o C to make sure alarm state was always on, and later the SBC was turned with only the watchdog service running in it, for a 4 hour period.

The figure 5.25 shows the contents of an e-mail message received in the personal account set as receiver, just seconds after the service was put on operation. The algorithm was able to identify at first try the status of the component as dangerous, given the false rules programmed for the test. The message includes the value measured and the correct units, thus it can be regarded as a successful evidence of right watchdog operation.

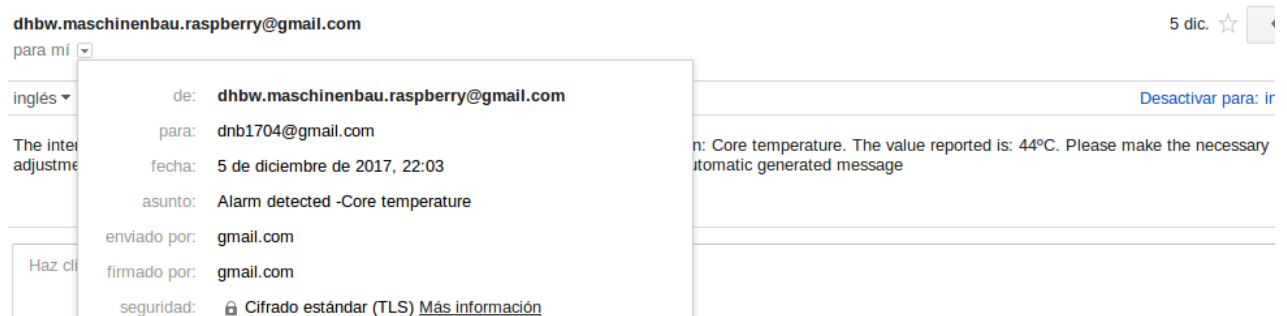


Figure 5.25: Message for setup alarm. Source: Own elaboration

The image 5.26 corresponds to a new alarm e-mail received on the same address exactly 3 hours after the first one, fact proved by the details on each message. The heading and the written contents of this second e-mail match the ones corresponding to a repeated alarm situation. There were no other messages received from the watchdog service during the trial period; therefore, the timing control (intended to avoid excessive inbox flow) and

longtime scenario identification were proven to work.

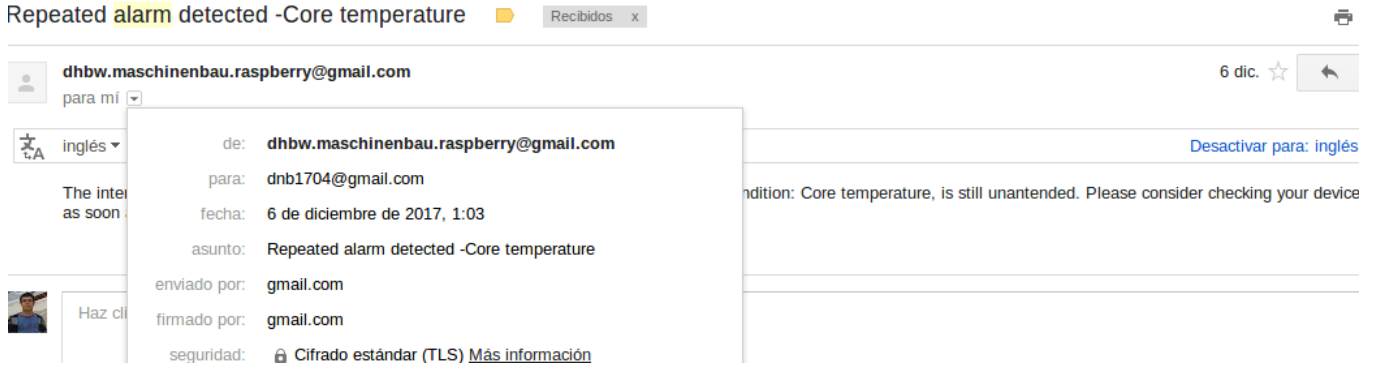


Figure 5.26: Message for repeated alarm. Source: Own elaboration

Considering the results obtained during testing stages, the watchdog was implemented with the real limits. Nevertheless, while executing trials in other system sections, several messages, identified as false positives were detected on the receiving account. The email on figure 5.27 was one of them. A quick inspection suggested a reading confusion inside watchdog program, caused by the use of extremely short delays. In this case, the delays were adjusted from 50 to 500 ms. After correcting the mistake, no more errors were perceived from the watchdog.

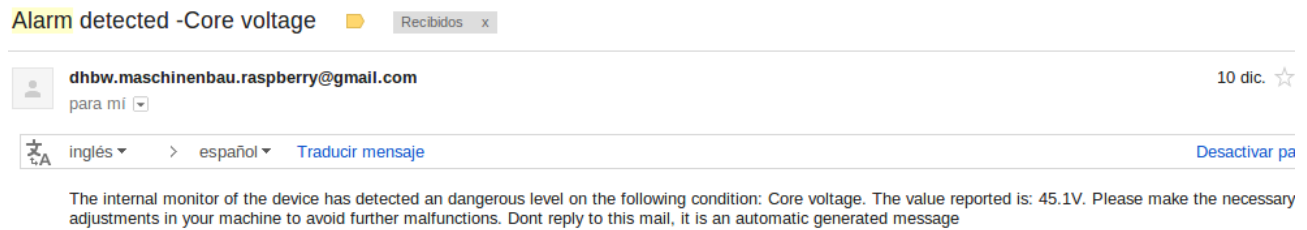


Figure 5.27: Example of wrongly activated alarm. Source: Own elaboration

5.7 Costs involved in the project

One of the goals proposed for the system designed was to keep it low-cost, using the proposed €250 limit as reference. A summary of all costs involved in final solution development was made after system implementation, to evaluate if the target was really achieved. The results of the budget review are presented on table 5.11. The methods for estimating each listed element vary between categories. For all the hardware components except the PCB, prices were easily registered in order lists or receipts. PCB case was a bit different, because the DHBW provides free fabrication service to students and researchers working with the institution. In real terms, no costs were added to the project's budget by this element. Nonetheless, looking the monitoring system as a product with a monetary value, it is necessary to include at least a PCB price estimation. With that goal in mind, web-based producers were searched. The price shown in the table was obtained through

[51]. This producer was chosen because they distribute to all Germany, and they also calculate prices for single delivery, small-sized, simple layered boards, just like the one required for the project.

About software, using free and open-source options was a design decision, explained in chapter 4, and no budget charge was added by the category. The case of human resource costs was a situation similar to the PCB price, because all work involved in design and installation was done without remuneration. No simple references exist about DHBW salaries, and the simpler alternative was to use any other known source. For the design work, considered an engineering labor, the reference consulted was [52], which states graduation projects general guidelines. The minimal salary established in the document is C\$300000 per month. The conversion to euros was done using the exchange rates C\$569.80 = \$1 and \$1.2338 = 1 €, consulted from [53]. For installation job, costs were computed as CNC lost work hours, since it was necessary to stop and keep the machine off during such labors. The minimum payment for a metal machining technician was found in [54] to be C\$12829.63 per every 8 hours. The aforementioned exchange rates were applied. Finally, for all other resources, the monthly price was approximated to 200 €, based on personal renting experience with similar accommodations.

The total value assigned to the project is €2197 or \$2711 using the consulted exchange rates. Such amount of money is way over the established low-cost limit. Nevertheless, the costs are including many factors that were not actually added in the real implementation. Considering only hardware elements in the budget, the value assigned is €245 (\$302), which is still under the limit proposed. Hence, the system developed is not expensive, at least speaking about materials. A big part of the monetary value the system acquires comes from the original work it requires, being a tailored solution.

Category	Element	Quantity	Total Price (€)
Hardware	Sensors	See table 3.1	102.84
	Other Ics	See table 3.3	68.92
	Other electronics	8 resistors, 6 capacitors, 1 crystal, 1 micro USB port	3.43
	CAT5 cable	15 meters	12.15
	PCB	1 piece (87 x 71 mm)	32.13
	Terminal blocks and sockets for PCB	7 pieces	2.47
	Cable binders	50 pieces	15
	Other installation materials	2 tubes of liquid adhesive, 2 rolls of adhesive tapes, several screws and nuts	8
Software	Database storage	500 MB	Free (free tier cloud service used)
	Web server renting	1000 h per month	Free (free tier cloud service used)
	Libraries and development environments	Several different software packages	Free (open-source software used)
Human resource	Enginnering labors	3 months as half-time job	1280
	Installing labors	32 hours of technician job	73
Other resources	Electricity, Internet, working space and access to lab equipment	3 months	600

Table 5.11: Costs of the project by category

Chapter 6

Conclusions and future work

After studying the results obtained from the different tests performed with the whole IoT system, the selected sensors and the respective processing, although having the required performance in lab placed tests, seem deficient for the real machine installation. Under controlled environment, the highest deviation and average errors found were around 20% and 10% of the real measured value (both found in the vibration sensor). However, the noise added by different elements (wiring, electrical motors, and controls), and the existence of uncontrolled external conditions affecting the machine and the sensor set itself introduced unexpected distortions that caused loss of useful data in some of them. Feed acceleration and sound measurements can be labeled as faulty, as the noise captured in both completely blocked the possibility of extracting useful information. In the acceleration measurements, the noise variation is in the exact same range as the expected measurements (1.5 m/s^2), and standard deviation reach the 30%. Meanwhile, for the microphone, instant variability between adjacent measurements is almost equal to the average result registered (71 to 76 mPa).

Still, considering quality terms, the measurements for temperature and cooler pressure, that kept standard deviations about 3% and 12% of the values registered, and the status monitoring over door and light (which reported zero errors during tests, mainly because of their discrete nature) generated perfectly valid information about real time operation with the filters proposed, and can be considered successfully implemented. Vibrations measurements, even with a remaining noise of 0.002 m/s , were distinct enough to allow extraction of properties from the data set, considering the reported values are two orders of magnitude over the noise. About the frequency analysis algorithms, the noise effects were heavily dependent on the sensor. In that way, the frequency results for table vibrations, which had no appreciable noise in magnitude measurements, were in turn free from heavy variability. In the other hand, acoustic frequency has the same issues as acoustic magnitude measurements (standard deviation of 33% the average measure). Apart from this irregular performance, the circuit achieved the proposed goal to be low-cost, keeping the total manufacture and installation materials expense under €250

In terms of data collection and communication, the microcontroller programming did not

have performance issues in any aspect but one. For counting the spindle revolutions, the processing speed and interrupt handling features of the Atmega 328P lead to erroneous results in the output. Moving the signal processing to the SBC was a temporary solution. In all other areas, the proposed design fulfilled data integrity, safety, visualization and access requirements. The Internet based communication chain to be highly reliable once an initial connection is established and would continue to operate uninterruptedly while Internet access is still available. This was proved by the 100% transmission rate and null average error reported for baud rates under 115200 and integer values. Also, the protocols used for internal data exchange could operate on the fast speeds require by the controllers, namely 70 ms, even under the noisy conditions introduced in the test environment.

About the graphical interface, the tests done with made up data proved database and server connection reliability, because all the datapoints intended to be stored were shown without any loss or distortion in value. Also, the graphics library functions allowed to distinguish very clear the different types of generated curves, in a way that general shape and details could be easily visualized, although the decision to not store repeated null points caused some slight changes in the way the graphs are perceived. Because all other functions in the webpage (zoom in the graphs, login and blocked access, downloadable data) were tested during this same trial, the general concept and functionalities of the GUI comply with the intended goals for the section.

For performance evaluation, the correlation test run suggests the existence of a heavy interaction between spindle activity and table vibrations, considering the coefficients for monotonic correlation of 0.7 both in magnitude and frequencies. Also, there is a mild influence over temperature raise, where the linear correlation coefficients are around 0.3. By theoretical references, these situations are known to directly affect the quality of operations. Although the quantity and variability of the measurements reported is not enough to generate a categorically analysis that links the variables, it was possible to use system data to extract deeper relationships, what proves the efficiency of a monitoring system for maintenance purposes. In general terms, although some of the measurements were not representative of the real conditions of the machine, the system achieves the proposed goals of monitoring simple individual conditions and store them in a way that could be easily and safely accessed to execute deeper analysis.

For future works in the same area, there are several recommendations to consider. About sensor processing, great performance improvement can be done with the inclusion of bias rejection for the vibrations sensors. The measurements for both accelerations and acoustics could be enhanced with the application of different digital filters, such as Chebysev or Butterworth. In that sense, the design strategy to choose the adequate parameters for the processing algorithms could also be based on analytic solutions and theoretical reference, instead of relying on empirical work, which was proven during the project to be a slow approach. The criteria used to establish frequency limits could rely on the presented results, making the design process even faster. In the case of acceleration, apart from using more advanced digital filtering and include the possibility of previous analog stage to prevent aliasing, it is highly recommended to firmly secure the sensors to its position

on the machine (as the sensitivity can be seriously affected by installation details) and to calibrate each sensor used individually. For the acoustic recordings, even more than recommending new filters, the conclusion is that electret microphones are not appropriate for the CNC environmental conditions, and a different approach should be tried.

In terms of communication, the most important improvement that could be done is changing the gateway main LAN connection from Wi-Fi to wire based, as the first method presented repeated issues to keep a long-time access. In the same way, an additional feature in the already working watchdog that could reset the SBC in absence of Internet could be useful in future implementations, as important periods of time remained without data because there was no warning about the lost connection. Regarding the microcontroller and the processing issue found in it, other models and providers should be considered for further implementations. Furthermore, there are some COTS options that integrate MCUs with microprocessors and Internet modules, collecting in one component the functions of controller and gateway. Such alternatives could be useful to avoid extra communication steps, but they would require to have one individual gateway on every monitored machine, which could cause data collisions if wrongly managed.

MCU demands can also be lowered from different sides. All data filtering methods were located in the controller for this project. Nevertheless, to expand the scope to more complex monitoring conditions, moving information processing to the cloud services is the best option. The sample rate used for frequency measurements, on the other side, was intentionally higher than the minimum required, but empirical results show no need for that prevision. Finally, with lower sample rates, the use of crystal oscillators could be avoided.

One last recommendation, regarding performance analysis, is that more work should be done with the spindle speed, temperature, and vibrations parameters, using incoming data from more diverse operation conditions. Tool state should also be compared to the measurements registered.

Bibliography

- [1] Baden-Württemberg State Government. Our State: Business Location-Home to commerce and industry. [Online]. Available: <https://www.baden-wuerttemberg.de/en/our-state/business-location/>
- [2] Duale Hochschule Baden-Württemberg. About us. [Online]. Available: <http://www.dhbw.de/english/dhbw/about-us.html>
- [3] Duale Hochschule Baden-Württemberg Karlsruhe campus. About Us-The dual study concept. [Online]. Available: <https://www.karlsruhe.dhbw.de/en/general/about-us.html>
- [4] Duale Hochschule Baden Württemberg Karlsruhe campus. Maschinenbau: Aus Lehre & Forschung-Modelfabrik. [Online]. Available: <https://www.karlsruhe.dhbw.de/mb/aus-lehre-forschung.html>
- [5] D. E. Kandray, *Programmable Automation Technologies - An Introduction to CNC, Robotics and PLCs*. Industrial Press, 2010. [Online]. Available: <http://app.knovel.com/hotlink/toc/id:kpPATAICN2/programmable-automation/programmable-automation>
- [6] S. Kalpakjian and S. Schmid, *Manufacturing, Engineering and Technology*, 7th ed. Pearson Education, 2014, vol. 2.
- [7] H. Kief and H. Roschiwal, *CNC Handbook*. Mc-Graw Hill Professional, 2012.
- [8] A. S. Agredo, “Elaboración de un plan de mantenimiento para fresadora MFG de 3 ejes del Laboratorio de Mecatrónica de la Universidad EAFIT,” Bachelor Thesis, Universidad EAFIT, Medellín, Colombia, 2012.
- [9] P. Wright, D. Dornfeld, and N. Ota, “Condition monitoring in end-milling using wireless sensor networks (WSNs),” *Transactions of NAMRI/SME*, vol. 36, 2008.
- [10] N. Ahmed, A. J. Day, J. L. Victory, L. Zeall, and B. Young, “Condition monitoring in the management of maintenance in a large scale precision CNC machining manufacturing facility,” in *2012 IEEE International Conference on Condition Monitoring and Diagnosis*, Sept 2012, pp. 842–845.

- [11] J. Z. Zhang and J. C. Chen, "Tool condition monitoring in an end-milling operation based on the vibration signal collected through a microcontroller-based data acquisition system," *The International Journal of Advanced Manufacturing Technology*, vol. 39, no. 1, pp. 118–128, Oct 2008. [Online]. Available: <https://doi.org/10.1007/s00170-007-1186-6>
- [12] T. Zafar, K. Kamal, R. Kumar, Z. Sheikh, S. Mathavan, and U. Ali, "Tool health monitoring using airborne acoustic emission and a PSO-optimized neural network," in *2015 IEEE 2nd International Conference on Cybernetics (CYBCONF)*, June 2015, pp. 271–276.
- [13] DR. JOHANNES HEIDENHAIN GmbH, *Technical Manual TNC 407, TNC 415, TNC 425*, Heidenhain.
- [14] R. Toulson and T. Wilmshurst, *Fast and Effective Embedded Systems Design - Applying the ARM mbed (2nd Edition)*. Elsevier, 2017. [Online]. Available: <http://app.knovel.com/hotlink/toc/id:kpFEESDAA1/fast-effective-embedded/fast-effective-embedded>
- [15] J. Huang and K. Hua, *Managing the Internet of Things - Architectures, Theories and Applications*. Institution of Engineering and Technology, 2017. [Online]. Available: <http://app.knovel.com/hotlink/toc/id:kpMITATA02/managing-internet-things/managing-internet-things>
- [16] R. Stackowiak, A. Licht, V. Mantha, and L. Nagode, *Big Data and the Internet of Things: Enterprise Information Architecture for a New Age*. Apress, 2015.
- [17] H. Geng, *Manufacturing Engineering Handbook*, 2nd ed. Mc-Graw Education, 2016.
- [18] K. Munir, M. S. Al-Mutairi, and L. A. Mohammed, *Handbook of Research on Security Considerations in Cloud Computing*. IGI Global, 2015. [Online]. Available: <http://app.knovel.com/hotlink/toc/id:kpHRSCCC03/handbook-research-security/handbook-research-security>
- [19] L. Newcombe, *Securing Cloud Services - A Pragmatic Approach to Security Architecture in the Cloud*. IT Governance Publishing, 2012. [Online]. Available: <http://app.knovel.com/hotlink/toc/id:kpSCSAPAS4/securing-cloud-services/securing-cloud-services>
- [20] J. M. Pintos, C. N. Castillo, and F. López-Pires, "Evaluation and comparison framework for platform as a service providers," in *2016 XLII Latin American Computing Conference (CLEI)*, Oct 2016, pp. 1–11.
- [21] S. Mohanty, M. Jagadeesh, and H. Srivatsa, *Big data imperatives: Enterprise 'Big Data'warehouse, 'BI'implementations and analytics*. Apress, 2013.
- [22] D. Gourley and B. Totty, *HTTP: the definitive guide*. O'Reilly Media, Inc., 2002.

- [23] C. Musciano, B. Kennedy *et al.*, *HTML, the definitive Guide*. O'Reilly & Associates, 1996.
- [24] K. R. Fall and W. R. Stevens, *TCP/IP illustrated, volume 1: The protocols*. Addison-Wesley, 2011.
- [25] S. Tilkov and S. Vinoski, "Node.js: Using JavaScript to Build High-Performance Network Programs," *IEEE Internet Computing*, vol. 14, no. 6, pp. 80–83, Nov 2010.
- [26] S. Popić, D. Pezer, B. Mrazovac, and N. Teslić, "Performance evaluation of using Protocol Buffers in the Internet of Things communication," in *2016 International Conference on Smart Systems and Technologies (SST)*, Oct 2016, pp. 261–265.
- [27] A. Nagy and B. Kovari, "Analyzing .NET serialization components," in *2016 IEEE 11th International Symposium on Applied Computational Intelligence and Informatics (SACI)*, May 2016, pp. 425–430.
- [28] K. Maeda, "Performance evaluation of object serialization libraries in XML, JSON and binary formats," in *Digital Information and Communication Technology and its Applications (DICTAP), 2012 Second International Conference on*, May 2012, pp. 177–182.
- [29] T. Dawborn and J. R. Curran, "docrep: A lightweight and efficient document representation framework," in *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, 2014, pp. 762–771.
- [30] BSON organization. BSON (Binary JSON): FAQ. [Online]. Available: <http://bsonspec.org/faq.htm>
- [31] Google Developer. Protocol Buffers Developer Guide. [Online]. Available: <https://developers.google.com/protocol-buffers/docs/overview>
- [32] Google Developers. Protocol Buffers API Reference. [Online]. Available: <https://developers.google.com/protocol-buffers/docs/reference/overview>
- [33] S. W. Smith, *The scientist and engineer's guide to digital signal processing*. California Technical Pub., 1999.
- [34] R. Oshana, *DSP Software Development Techniques for Embedded and Real-Time Systems*. Elsevier, 2006. [Online]. Available: <https://app.knovel.com/hotlink/toc/id:kpDPSDTE1/dsp-software-development/dsp-software-development>
- [35] T. T. Tran, *High-speed DSP and Analog System Design*. Springer Science & Business Media, 2010.
- [36] Y. He, *Technical Datasheet-5mm Phototransistor T1*, Everlight Electronics Co., Ltd.
- [37] Conrad Electronics, *Datasheet-File No.: H21205C10*, Conrad Electronics SE.

- [38] Analog Devices, *Preliminary Technical Data-ADXL335*, Analog Devices Inc.
- [39] Honeywell, *Honeywell Sense and Control Pressure Sensors*, Honeywell International Inc.
- [40] Maxim Integrated, *DS18S20 High-Precision 1-Wire Digital Thermometer*, Maxim Integrated Products, Inc.
- [41] SHARP, *GP2D150A-General Purpose Distance Measuring Sensor*, SHARP.
- [42] Microchip Technology Inc. New/Popular 8-bit Microcontrollers Products. [Online]. Available: <http://www.microchip.com/ParamChartSearch/Chart.aspx?branchID=1012>
- [43] Analog Devices, *LCMOS 4-/8-Channel High Performance Analog Multiplexers-ADG408/ADG409*, Analog Devices Inc.
- [44] ROHM Semiconductor, *Non-Isolated Step-Down 3-Terminal DC/DC Converters-Datasheet*, ROHM Co., Ltd.
- [45] National Semiconductor, *LM386-Low Voltage Audio Power Amplifier*, National Semiconductor Corporation.
- [46] Atmel, *ATmega328/P-DATASHEET COMPLETE*, Atmel Corporation.
- [47] MongoDB Inc. MongoDB for GIANT ideas. [Online]. Available: <https://www.mongodb.com/>
- [48] NPM, Inc. npm Documentation; 01-What is npm? [Online]. Available: <https://docs.npmjs.com/getting-started/what-is-npm>
- [49] The Raspberry Pi Foundation. Raspberry Pi FAQs. [Online]. Available: <https://www.raspberrypi.org/help/faqs/>
- [50] Minitab Express Support. A comparison of the Pearson and Spearman correlation methods. [Online]. Available: <http://support.minitab.com/en-us/minitab-express/1/help-and-how-to/modeling-statistics/regression/supporting-topics/basics/a-comparison-of-the-pearson-and-spearman-correlation-methods/>
- [51] Eurocircuits GmbH. Naked proto pcb service (calculator). [Online]. Available: <http://be.eurocircuits.com/shop/orders/configurator.aspx?loadfrom=web&service=nakedproto&lang=en>
- [52] Área Académica de Ingeniería Mecatrónica ITCR, “Normas para proyecto de graduación,” Published for internal use of ITCR professors and students, 3 2017.
- [53] Banco Central de Costa Rica. Tipos de cambio. [Online]. Available: www.bccr.fi.cr/indicadores_economicos_/Tipos_cambio.html

-
- [54] Ministerio de Trabajo y Seguridad Social de Costa Rica. Lista de salarios. [Online]. Available: <http://www.mtss.go.cr/temas-laborales/salarios/lista-salarios.html>
- [55] Arduino. From Arduino to a Microcontroller on a Breadboard. [Online]. Available: <https://www.arduino.cc/en/Tutorial/ArduinoToBreadboard>

Appendix A

Example of the Cooley-Tukey FFT algorithm

For the purpose of proving how the FFT is calculated, the following example will use a square signal centered around zero level, with peak-to-peak amplitude of 2 and a given frequency f . If this signal is digitalized with a sample rate of $2f$, the result is an array of values in the following way:

$$[1, -1, 1, -1, 1, \dots]$$

To keep the example short, the amount of samples considered would be 4. The algorithm starts by sorting out the elements on the array in groups of odd and even indexed values. Considering the initial array is $[1, -1, 1, -1]$, executing the sort process by placing the even elements to the right generates the following new array $[1, 1, -1, -1]$. Now, the basic formula of FFT, as stated in equations 2.15 and 2.16, can be recalled recursively to create frequency spectrum with powers of 2 number of elements. For the first case, $N=2$. Taking the first two elements of the sorted array:

$$X_0[0] = E + e^{-\frac{2\pi j}{2}0}O = 1 + e^0 * 1 = 2$$

$$X_0[1] = E - e^{-\frac{2\pi j}{2}0}O = 1 - e^0 * 1 = 0$$

The same process, applied to the following two elements, would be:

$$X_1[0] = E + e^{-\frac{2\pi j}{2}0}O = -1 + e^0 * -1 = -2$$

$$X_1[1] = E - e^{-\frac{2\pi j}{2}0}O = -1 - e^0 * -1 = 0$$

So, after the first step, the results of the FFT algorithm are two arrays of 2 elements. Reminding that the values in the frequency spectrum are complex numbers, the respective arrays can be expressed as:

$$X_0 = [2 + 0j; 0 + 0j]$$
$$X_1 = [-2 + 0j; 0 + 0j]$$

The following step would take $N=4$. The calculations for this step are:

$$\begin{aligned} X[0] &= E[0] + e^{-\frac{2\pi j}{4} \cdot 0} O[0] = X_0[0] + e^0 X_1[0] = 2 + -2 = 0 \\ X[1] &= E[1] + e^{-\frac{2\pi j}{4} \cdot 1} O[1] = X_0[1] + e^{-\frac{\pi j}{2}} X_1[1] = 0 + (\cos(\frac{\pi j}{2}) - \sin(\frac{\pi j}{2})j) * 0 = 0 \\ X[2] &= E[0] - e^{-\frac{2\pi j}{4} \cdot 0} O[0] = X_0[0] - e^0 X_1[0] = 2 - (-2) = 4 \\ X[3] &= E[1] - e^{-\frac{2\pi j}{4} \cdot 1} O[1] = X_0[1] - e^{-\frac{\pi j}{2}} X_1[1] = 0 - (\cos(\frac{\pi j}{2}) - \sin(\frac{\pi j}{2})j) * 0 = 0 \end{aligned}$$

Because the value of N is equal to the total amount of samples, this is the final step. The result of the FFT, expressed in the way of phasors, would be:

$$X = [0 + 0j; 0 + 0j; 4 + 0j; 0 + 0j]$$

Although in this case all the values obtained are purely real, that is not always like that. To know the total magnitude of a single phasor, equation 4.1 should be applied. For the sake of keeping the example complete, the calculations of magnitude are the following:

$$\begin{aligned} |X[0]| &= \sqrt{0^2 + 0^2} = 0 \\ |X[1]| &= \sqrt{0^2 + 0^2} = 0 \\ |X[2]| &= \sqrt{4^2 + 0^2} = 4 \\ |X[3]| &= \sqrt{0^2 + 0^2} = 0 \\ |X| &= [0; 0; 4; 0] \end{aligned}$$

To relate the magnitudes of the spectrum to real frequency values, one have to consider the base frequency for the FFT analysis, given by f_{sample}/N . In this case, the base frequency would be:

$$f_{base} = \frac{f_{sample}}{N} = \frac{2f}{4} = \frac{f}{2}$$

Then, each index in the frequency spectrum array represents a integer multiple of the base frequency. A frequency array can be generated parallel to the magnitudes, as follows:

$$F = [0f_{base}; 1f_{base}; 2f_{base}; 3f_{base}] = [0\frac{f}{2}; 1\frac{f}{2}; 2\frac{f}{2}; 3\frac{f}{2}] = [0; \frac{f}{2}; f; \frac{3f}{2}]$$

By inspection, one can conclude the only frequency band with a value different than zero, and therefore the main component of the signal, is f . This was the original value set to the signal, so the Cooley-Tukey algorithm is proven to work

Appendix B

Installation and user guide

This section explains the steps required to install and get to work the different software components involved in the project, and also provides instructions on how to make changes to the provided programs. A very important detail to consider is that the whole system was designed in a Linux-based environment and it is intended to be used in that way. Also, most of the software specifically created for the project was developed using the NodeJS language. As the components requiring programming have it already installed for the testing done, they can be used without worrying for any software installation. However, if the user intends to make changes to any of the programs, it should consider these conditions.

B.1 Quick Setup

1-The physical installation requires the set of sensors (already attached with cables in groups), the controller circuit soldered onto the PCB, the Raspberry Pi, and two cell-phone chargers of 1 A. The circuit should have the ADG409 multiplexer and the Atmega 328p mounted on it, but they are deattachable, so you should check them before starting. The same should be done with the Raspberry and the micro SD

2-To place the sensors, make sure the machine is off and without any tool in the spindle. As explained in chapter 3, the sensors are grouped by their location in the machine: table, column, and the 3 independent sensors: light, door status and pressure. For the column sensors, a wide cable binder can be used around the structure to fasten to it.

3-Table accelerometer can be fastened to the table grooves using a screw, nuts and the hole the sensor itself has. For the other table sensors, there is no proffered method of attachment; adhesive tape should be enough.

4-To place pressure sensor, find the relief valve of the compressor, in the backside of the machine. Locate the tee on the output hose. Introduce the hose fixed to the sensor in the free hole of the tee and move the black switch in it to the “pass” position.

5-Each sensor cable needs to be connected to the controller circuit using the respective block. The blocks input can be opened using the screw on top of them (flat screwdriver required), and are labeled with their function in the following way: 5V: 5V power input 0: Ground 3V: 3.3V power input X: X feed accelerometer Y: Y feed accelerometer Z: Z feed accelerometer Vt: Table vibrations sensor Vs: Column vibrations sensor T: Temperature sensors M: Microphone L: Light sensor D: Door status sensor S: Spindle speed sensor

6-If the MCU and multiplexer are not in position, insert them in the corresponding platforms. The direction of the multiplexer should make coincide the chip indent with the socket one, but for the microcontroller should be the opposite one. In case of doubt, review figure 3.1.

7-Plug one of the chargers in the micro-USB port in the backside of the PCB, and power it on. The red LED of the Bluetooth module on the circuit should get on and start to flicker.

8-Insert the SD and the other charger in the corresponding slots of the Raspberry, and power it on. The green and red LEDs should light on, and after a couple of seconds the LED on the PCB should change the frequency of flickering.

9-The programs in the Raspberry should run automatically on reboot. You can check their state opening the terminal inside the Raspberry and running the following command: **sudo systemctl status *name-of-service.service*** Also, for the data upload services (named “Comm” and “Status”), a work log can be retrieved executing this command: **sudo journalctl -f -u *name-of-service.service***

8-In case the logs for the services show the following message: “MongoError: connect ECONNREFUSED”, you should contact the personnel in charge of internal network and ask them to raise the firewall restrictions for host “ds125335.mlab.com” at port “25335”.

9-The data can be checked at the webpage: <http://dhw-ka-maschinenbau.herokuapp.com/>. To have full access to all the graphs, you would need a valid username and password. If you do not have an user assigned, use the guest user credentials.

B.2 Modifying existing software

For the following sections, it is assumed you have access to the full collection of programs and related documents involved in the project. The whole group of files was left to the personnel of Mechanic Engineering at the DHBW Karlsruhe, so you can contact them in case of missing archives.

B.2.1 Installing NodeJS and libraries

For Debian Linux distributions and related ones (like Ubuntu) the installation of Node can be done through the terminal, via package management. Just run the following

commands:

```
sudo apt-get update
```

```
sudo apt-get install nodejs
```

It is also recommended to install NPM, which is an utility that allows the management of Node libraries and packages to be done easily. To install it, run the command: **sudo apt-get install npm**

All the libraries used for both the Raspberry services and the webpage server programs are included in the folder named “node-modules”. You should place it inside the same folder that would contain all other programs you would use, to avoid mistakes. In case the folder is missing, you can always get them from file named “package.json”. In that case, you need to get that file inside the common folder for programs, access that folder from terminal and run the command: **npm install**

In case you are going to use a new Raspberry Pi for the project, you should also execute the previous instructions in it. Consider that the Raspberry requires to burn an adequate operative system image in the SD before putting it to work. For this project, the operative system chosen was Raspbian (available for downloads in

<https://www.raspberrypi.org/downloads/raspbian/>) but any Linux distribution would work fine. Also, remember that Raspberry is a minimal computer unit, so in order to work in it you would need a screen with HDMI port and a keyboard, or remote access services like SSH or VNC servers.

B.2.2 Modifying Raspberry Pi services

The services already included in the project’s Raspberry Pi are fully enabled to be modified and adapted without any problem. To change any characteristic in their operation, locate the corresponding script for the service desired (both share the same name) and open it. If the system warns you about the file, ignore all options and select “Open File”. After making the desired changes, you need to restart the service. To do so, execute the following commands:

```
sudo systemctl stop name-of-service.service
```

```
sudo systemctl start name-of-service.service
```

In case you want to include to include a whole new service in the Raspberry, follow these instructions:

- 1-Write the source script for the service, in the programming language you prefer. Consider that you need to have installed the compiler or interpreter for that language in your Raspberry.

- 2-Save the script in a known folder in your Raspberry

3-Go to the folder “/lib/systemd/system/” and create a new text file named “*name-of-service.service*”. The file should contain the following lines:

[Unit]

Description=*add a description here*

After=multi-user.target

[Service]

Type=simple

ExecStart=*folder where compiler or interpreter is installed and direction where you saved your script*

Restart=on-abort

[Install]

WantedBy=multi-user.target

As a tip, compilers are usually installed in the “usr/bin/” folder.

4-Open the terminal and execute this commands, one by one:

```
sudo chmod 644 /lib/systemd/system/name-of-service.service
```

```
chmod +x direction where you saved your script
```

```
sudo systemctl daemon-reload
```

```
sudo systemctl enable name-of-service.service
```

```
sudo systemctl start name-of-service.service
```

After this, your script would be automatically executed every time the Raspberry powers on.

B.2.3 Linking a new Bluetooth module

If you want to connect any Bluetooth device to a Raspberry Pi 3 (that has Bluetooth reception included in it), you need to establish first a secure link. You can do this directly from the terminal, following this instructions:

1-Open the terminal and run the command **sudo bluetoothctl**. You would see a list of all devices already recognized by the system, and the terminal should indicate you are working on the “bluetooth” mode.

2-Run the command **agent on**, and then **default-agent**. You should see success messages for both

3-Run the command **scan on**. This will show a list with all the available Bluetooth devices, with their corresponding addresses (series of alphanumeric combinations) and, if

you allow from the device itself, the nickname of it.

4-Select the address of the device you want to connect, and then execute **pair *address***. If your device has a password, you would be asked to type it in.

5-After the device is rightly paired, you should execute the command **trust *address***, to be sure the system will always recognize it.

6-Come out of the “bluetooth” running **exit** and in the regular terminal, run this command: **sudo rfcmm watch hci0**. This should connect the device and create a port in the Raspberry, in the direction “/dev/rfcomm0”, that you can use as any serial port.

You should run the **rfcomm** command everytime the Raspberry or the Bluetooth device restarts. To make easier this task, you can create a service that takes care of it. Simply create a script with the “#!/bin/bash” first line and the command after it, save it with the “.sh” extension and follow the regular instructions for services. Because it is a shell script, it will not need a compiler specification.

B.2.4 Creating new server repository

The webpage for the project was created using Heroku as PaaS. By default, Heroku stores the page info using Git repositories, so this is the easiest way to keep an archive for future work. To create a new Git repository with the webpage files:

1-Verify that you have Git installed by running the command **git**. If installed, you will see a list of common commands. If not, you can download it running **sudo apt-get install git-all**

2-Copy all the webpage files, including the programs, HTML scripts and images, in a known folder on your PC. Keep an eye on the folder order provided, as the webpage needs to keep track of links between different directions.

3-From the terminal, access the page files folder.

4-Run the command **git init .** and then **git add ..** You should see confirmation messages.

5-Run the command **git commit**. You will be asked to add a comment about the changes done to the repository. Write something you consider important about it, then click CTRL+X and then the key “Y”.

B.2.5 Deploying changes to the webpage

The changes you do on the webpage files will not be reflected on the final product until you update the repository in the Heroku server. To do it, you can use the dedicated command line or CLI, which is the fastest way to link the repository to the cloud. The full instructions to update the page are:

1-To install Heroku CLI, run in order the following terminal commands:

```
sudo add-apt-repository "deb https://cli-assets.heroku.com/branches/stable/apt
./"
```

```
curl -L https://cli-assets.heroku.com/apt/release.key — sudo apt-key add -
```

```
sudo apt-get update
```

```
sudo apt-get install heroku
```

2-Access to the Heroku account running **heroku login** in the terminal. You will be asked about the email and password for the account; write the project’s credentials as provided in the webpage files.

3-Access the repository folder from the terminal. If it is the first time you try to upload the repository to Heroku, run **heroku git:remote -a dhw-ka-maschinenbau**. If not, continue to the next step

4-If you created a new file inside the folder, run **git add .**, and then **git commit**. Write a comment and exit with CTRL+X. If you only made changes to already existing files, you only need to do the commit part.

5-Run **git push heroku master**. After this, changes should be visible in the webpage

The CLI also allows you to check the logs to search for errors (run **heroku logs** from inside the folder) and to rename the webpage (run **heroku apps:rename newname**. About webpage files, remember that the one named “Procfile” always define the main program to run inside server.

B.2.6 Administering database users

You can change the users defined for the database and webpage access using the Heroku credentials. To do it, go to the following website: <https://dashboard.heroku.com/login>. Enter the credentials, and you will see a list of pages associated with the account. Select the one called “dhw-ka-maschinenbau” (or the current name you have for it). Then, under the “Installed add-ons” title, search for “mLab MongoDB”. After clicking on it, you will be redirected to the database presentation page. You can see there tabs to visualize data stats, search for specific datapoints and even erase the collection from there. The tab named “Users” is the one indicated for roles and permissions. From there, you can add a new user or remove existing ones. When adding a new user, you can select if you want it to be read-only or not.

B.2.7 Programming the MCU

Any microcontroller needs an external programming device to upload the desired instructions in it. For the Atmega 328p, the programmer can be replaced for an Arduino UNO, which is a cheap and widely available solution. To use this method, the Arduino IDE is

required. The steps to program the Atmega chip are:

1-Go to the following website: <https://www.arduino.cc/en/Main/Software> and download the version of the IDE that suits your PC. After downloading the package, open the files and follow instructions for installation.

2-Open the IDE, go to Files/Examples and select the one named “ArduinoISP”. Compile it and load it to the Arduino board, using the corresponding icons in the upper side of the screen. Remember to select, in the “Tools” tab, “Arduino/Genuino Uno” as the board, “AVR ISP” as programmer, and the right port were you connected the USB cable for the Arduino.

4-With the Arduino board and the MCU, prepare the circuit shown on figure B.1. Notice that you will also need a 1 k Ω resistor and the crystal clock that you will use for the MCU common operation

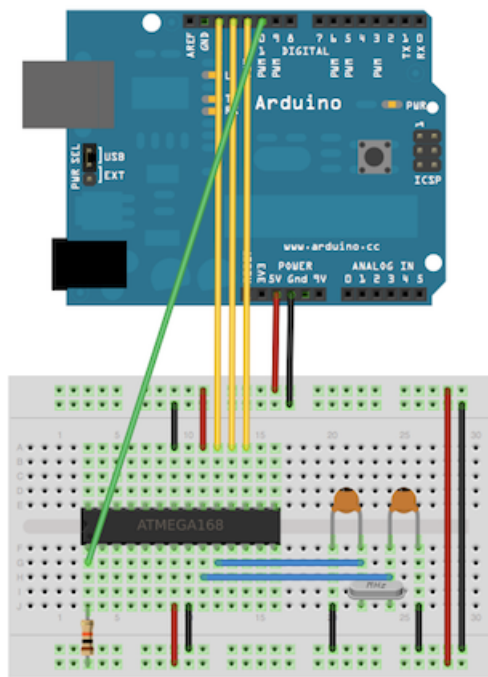


Figure B.1: Circuit for Atmega 328 programming. Source: [55]

3-In the “Tools” tab, change the board to “Arduino Duemilanove or Nano w/ ATmega328” and the programmer to “Arduino as ISP”.

4-If it is the first time you program that specific MCU, run Tools/Burn bootloader

5-Search and open in the IDE the program you want to load to the controller. Compile it to check for errors, and then run Program/Upload using programmer. This flashes the MCU with the chosen script.

Keep in mind that for the control program used in this project, is necessary to have the following files inside the “libraries” folder of the Arduino installation directory, before

compiling it: the “OneWire” library for the temperature sensors; “TimerOne” library for the timing structure; and the “nano” files, including “Model.pb.h” and “Model.pb.c”, for the Protocol Bufers encoding

B.2.8 Changing the Protobuf message

To allow the upload service on the Raspberry and the control program in the MCU to run the Protobuf utility, different files compiled from the .proto archive to the specific language are needed. If you make a change to “Model.proto” or need to create a new type of message, you need to recompile the files. For the Raspberry ones, the JavaScript version is the one required. The steps to use it are:

1-From terminal, get into the folder containing the .proto file. It is recommended that this folder should be the same containing the rest of programs.

2-If you have not installed the compiler before, run the command **npm install google-protobuf**. Also, download from <https://github.com/google/protobuf/releases> the package corresponding to JavaScript, and un-compress it. This is the compiler, “protoc”

3-Every time you need to compile a new version, run the command:

protoc -js_out=import_style=commonjs,binary:. *name-of-file*.proto.

This will create a file named *name-of-file*_pb.js. You need to place that file in the same folder as “Comm4.js”

For the microcontroller, the version is known as “Nanopb”. To use it:

1-If you have not downloaded the “protoc” compiler, do it following previous instructions

2-Download the compressed file from <https://jpa.kapsi.fi/nanopb/download/>. Un-compress it inside the folder containing the .proto file

3-From the uncompressed files, you need to copy the ones named “pb”, “pb-common” and “pb-encode”, with both .c and .h extensions to the Arduino libraries folder. If you already copy there the “nano” folder included in the program files of the project, ignore this step.

4-From the terminal, get into the folder containing the .proto file and run the command:

./nanopb/generator-bin/protoc -nanopb_out=.*name-of-file*.proto -D PB_FIELD_16BIT=1

5-The output should be 4 files. Copy the ones named *name-of-file*_pb.c and *name-of-file*_pb.h to the Arduino libraries folder. Compile and load the controller program to the MCU again

Appendix C

Electronic circuit in detail

In the next page, an schematic drawing of the main circuit for the monitoring system is presented. It is included as a separate section to allow more level of detail for a better visualization.

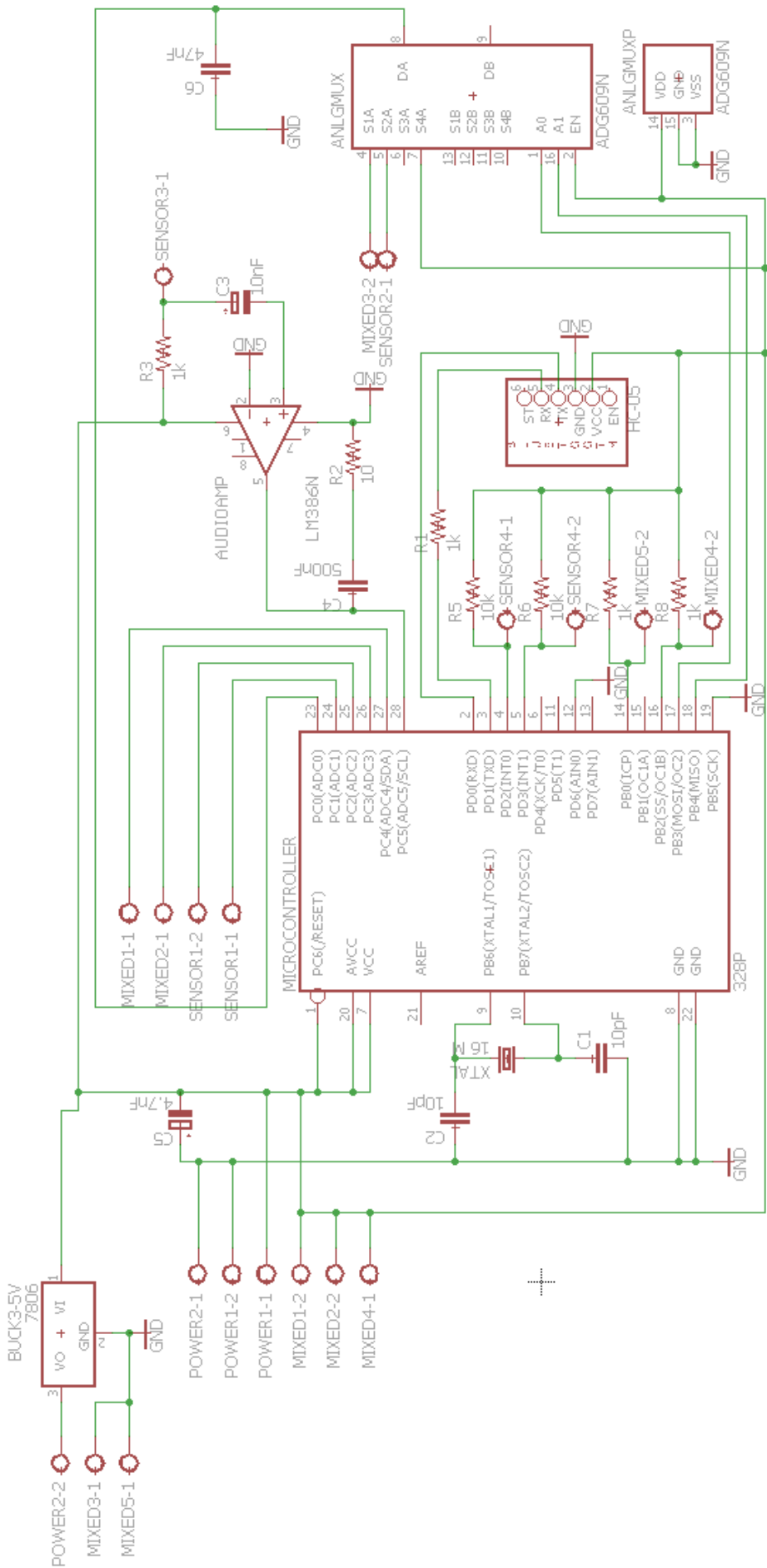


Figure C.1: Main electronic circuit

Index

CNC machine, [7](#)

Database selection, [36](#)

Fast Fourier Transform, [19](#)

FFT implementation, [41](#)

Filtering, [31](#)

Hermle UWF900E specs, [10](#)

HTTP and HTML, [13](#)

Internet of Things, [10](#)

Microcontroller selection, [26](#)

Monitoring results, [58](#)

PaaS providers, [37](#)

PCB, [93](#)

Platform as a Service, [12](#)

Protobuf implementation, [30](#)

Protocol Buffers, [15](#)

Sensor calibration, [47](#)

Sensor selection, [24](#)

Watchdog, [42](#)

Web site design, [43](#)