FEDERAL UNIVERSITY OF SANTA CATARINA
JOINVILLE TECHNOLOGY CENTER
BACHELOR OF MECHATRONICS ENGINEERING

GABRIEL LEAL SILVA

EMBEDDED SOFTWARE DEVELOPMENT FOR A VIBRATION SENSOR NODE IN A SMART SENSOR PLATFORM.

Joinville
2022

GABRIEL LEAL SILVA

EMBEDDED SOFTWARE DEVELOPMENT FOR A VIBRATION SENSOR NODE IN A SMART SENSOR PLATFORM.

This graduation thesis is presented to fulfill the partial requirement to obtain the title of bachelor in the course of Mechatronic Engineering, at the Joinville Technology Center, from the Federal University of Santa Catarina.

Professor Advisor: PhD. Anderson Wedderhoff Spengler

Co-advisor: MSc. Praveen Mohanram

Joinville
2022

I want to dedicate this work to my two grandmothers, Haydée and Lourdes, who alone raised three children each, and to my parents and sister, Silvio, Heloisa, and Heloara; who never stopped believing in me, even when myself could not to do so.

ACKNOWLEDGMENTS

I would like, firstly, to thank God for giving me wisdom, patience, and, above all, this strong resilience that only He knows how I have it. Those attributes were essential to walk for this hole path and fulfill my goals and dreams.

To my family – especially to my dear parents Silvio and Heloisa, for being the best parents and friends that someone could have, and to my beloved sister Heloara, to whom I always tried to make the best of myself in order to be a role model and to prove that studying pays off – I want to thank them all for all the love, care, and support given through my whole life. I love you all.

I would like to say thanks to all my friends, starting with my friends of the group "Mecamigos" for all the support and friendship through all these years. A special thanks to the ones who lived with me – Juninho, Salviano and our special member Jacy – for being my second family. Another special thanks to Peteck, Chewie (Vitor Engers) and Russi, for being with me since the beginning until the end. I thank you all for making everything easier, especially during the night shifts. To Madu and Julia, who are also part of this family. To Rizzi and family, for receiving so well in their house as if I was a member of them. To all the friends that I made while tutoring, especially Carol, Dudu and Void (Gabriel), it would not have been the same without you. To Gio and M.A., who helped me to survive my first years of college. To my group "Ditadura Carnavalesca", from even before the university and now after it, thanks for all the fun that we have. And to all the friends that I could not cite here, because it is already a huge text, thank you.

Here in Germany I would like to say thanks to all my friends who lived with me, especially Domi, Stefi, Bela, Louis, Pawan, Victor, Polly, Helena and Rebeca. Those guys made the hard days lighter. A special thanks to Vitoria Sedrez, who helped and still helps a lot to make all this possible, and to João Victor, who is my party partner since Brazil.

To my supervisor Anderson, not only for accepting this complicated task of the distance orientation, but also for all the support during my whole graduation course, for the tips, advises and also the jokes that helped to keep the good mood. I would like also to thank the members of the examination board, Diego dos Santos Greff and Raphael Diego Comesanha e Silva, for accepting and taking the time to read and evaluate my last work as a graduation student.

To Fraunhofer IPT, for accepting me as working student and making this thesis possible. A special thanks to my supervisor Praveen for all the technical support provided and to Sarah for all the help and care provided.

At last, but not the least, I woud like to thank immensely the Federal University of Santa Catarina, which provided not only the technical but also the personal support so I could reach this point here today. I am eternally grateful to the Brazilian public university, especially UFSC, and I swear defend it and respect its name wherever I go.

"Love is an act of courage". (Paulo Freire)

ABSTRACT

The arrival of the 5G technology brought a new perspective to the IIoT field. Relying on even faster data rates than its forerunner, the new standard enables a wider range of sensors to be connected to the internet. In this scenario, the Fraunhofer IPT developed a concept for a multi-sensor platform, a smart sensor board which allows real-time transmission of industrial machinery data through the 5G network. This work presents the development of a software for one of its sensor nodes, using the vibration sensor as a reference. In order to supply certain a pre-processing stage right after the data sampling, an FFT and a Wavelet transformation were implemented. The response time and the accuracy of both features were obtained using different methods. At the end, the values were analyzed and some promising results were achieved. The FFT delivered answers between 750 to 950 microseconds, depending on the chosen property, and its accuracy was overall good for the application, considering a qualitative analysis. Final considerations and possible future works ere also presented and discussed.

Keywords: MSP. FFT. Wavelet. Smart sensor.

RESUMO

A chegada da tecnologia 5G trouxe uma nova perspectiva para o campo da IIoT. Apresentando uma taixa de transmissão de dados ainda maior que o seu antecessor, a nova geração ptorna possível que uma maior gama de sensores possa ser conectada a internet. Tendo em vista essse cenário, o Frunhofer IPT desenvolveu um conceito para uma plataforma multi-sensores, uma placa de sensores inteligentes que permite uma transmissão em tempo real de dados da produção mecânica através da rede 5G. Esse trabalho apresenta o desenvolvimento de um software para um dos dos nós de sesores inteligentes, usando o sensor de vibração como referência. Com o objetivo de fornecer uma etapa de pré-procesamento logo após a obtenção do dado, uma FFT e uma transformada de Wavelet foram implementadas. O tempo de resposta e a acurácia das duas foram obtidas usando diferentes métodos. No fim, os valores foram analisados e alguns resultados foram obtidos. A FFT entregou respostas entre 750 e 950 microegundos, dependendo da propriedade escolhida, e a sua acurácia foi em geral boa para a aplicação, considerando uma análise qualitativa. Considerações finais e possíveis trabalhos futuros também foram apresentados e discutidos.

Palavras-chave: MSP. FFT. Wavelet. Sensor inteligente.

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS AND ABBREVIATIONS

5G Fifth generation of the broadband cellular networks

ADC Analog-Digital Converter

API Application Programming Interface

CMSIS Common Microcontroller Software Interface Standard

CPU Central Process Unit

CSV Comma-Separated Values

DFT Discrete Fourier Transform

DMA Direct Access Memory

FPU Floating Point Unit

GNU GNU is Not Unix

GPL General Public License

GPU Graphical Processing Unit

GSL GNU Scientific Library (GSL)

IIoT Industrial Internet of Things

IPT Fraunhofer Institute for Production Technology

LwM2M Lightweight Machine to Machine

M2M Machine to Machine

MHz Megahertz

MSP Multi-Sensor Platform

MSps Megasamples

RFFT Real Fast Fourier Transform

SPI Serial Peripheral Communication

SPI Serial Peripheral Interface

UDP User Datagram Protocol

USART Universal Synchronous/Asynchronous Receiver-Transmitter

LIST OF SYMBOLS

$a_n(t)$	A signal described in the Fourier series form
j	Imaginary unit
t	Continuous time
k	Iterative variable
ω_0	Fundamental frequency
a_k	Fourier Series harmonics
$X(j\omega)$	Continuous Fourier Transform
n	Discrete time
N	Number of samples
$T_f^{wav}(a,b)$	Continuous Wavelet Transform
a	Dilation or scale parameter
b	Translation parameter
ψ	Mother wavelet
$\psi_{m,n}$	Discrete wavelet

TABLE OF CONTENTS

1 INTRODUCTION

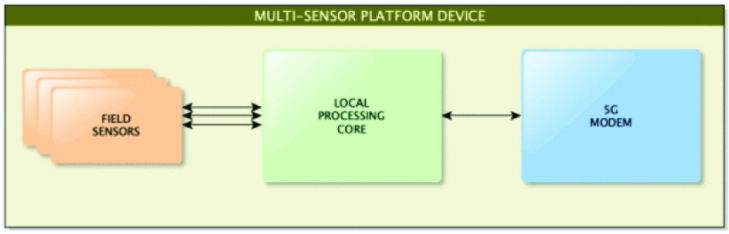
The Fraunhofer Institute for Production Technology (IPT) is a member of the Fraunhofer Society in Aachen, Germany. It develops system solutions for the networked, adaptive production of sustainable and resource-saving products. Inside its Metrology Department, the Digital Infrastructure group, through a partnership with the laboratory of the company Ericsson in Aachen, researches the possible applications of the fifth generation of the broadband cellular networks (5G) in the industrial production (FRAUNHOFER IPT, 2019).

As shown by Shafique et al. (2020), the arriving of 5G brought a series of new possible applications for the wireless technologies. Relying on a latency of less than 1 millisecond and with a throughput up to the 5 Gigabytes per second, the 5G represents a turning point in data communication for wireless sensors. Surpassing the speed of its predecessor by 10 to 15 times, the 5G brings an improvement to a well-known problem in this field: the non-feasibility of transmitting high-frequency data without wires (AL-FALAHY; ALANI, 2017).

For example, in the machinery field the 10-15 milliseconds delay attached to the 4G technology could be enough for slow-rate variables, such as temperature or pressure, but when it comes to high-frequencies, as speed or vibration, the use of wireless communication was not able to meet the time constraints required for this kind of process. However, the use of cables inside the machine were also unfeasible, due to the nature of the problem and the rough environment, making it impossible to analyze these variables while a piece is being manufactured.

In this scenario, the Fraunhofer IPT - during the international consortium 5G-SMART - started the design of a Multi-Sensor Platform (MSP). Its basic concept, which can be seen on Figure 1, is a central process unit that receives data from different types of sensor, and, after some pre-processing, sends the collected data via a 5G-modem (SCHMITT et al., 2020).

Figure 1 – Basic concept behind the Multi-Sensor Platform

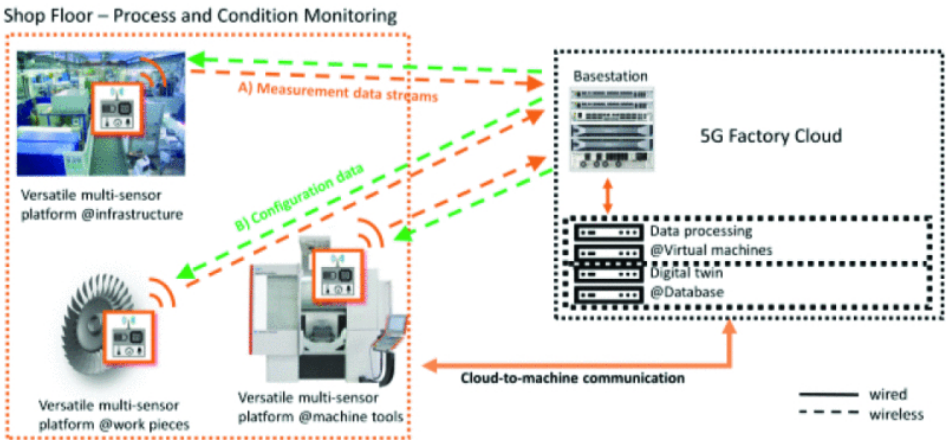


Source: Schmitt et al. (2020)

Considering a scenario like the shop floor of the Fraunhofer IPT, which is

depicted in Figure 2, the MSP is supposed to sample and send data of machinery process in real-time. Relying also on the pre-processing in hardware, the system should be able to send less data than usual, as well as decentralize its calculations.

Figure 2 – Example of applications of the multi-sensor platform

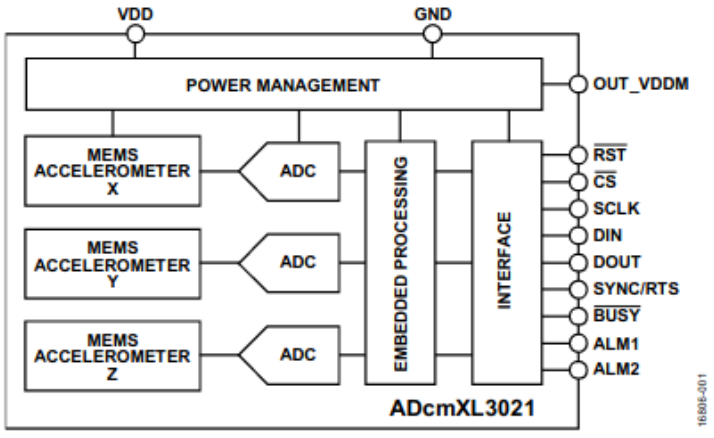


Source: Schmitt et al. (2020)

According to Schmitt et al. (2020), the MSP will work as an use-case that is cheaper and more complete that ones available in the market. After its first goal of a smarter monitoring system, the next step for the MSP will be to detect failures on the working piece before it happens, utilizing artificial intelligence on the edge.

Since this work focus in the embedded software developed for one of the nods of the MSP – focusigng in the vibration one – a quick market research was conducted trying to find a product which would presented a similar functionality. The product found was the sensor ADcmXL3021 from Analog Devices (2019), which has its block presented in the Figure 3 below.

Figure 3 – Block diagram of the sensor ADcmXL3021 from Analog Devices



Source: Analog Devices (2019)

As shown in Analog Devices (2019), this is a tri-axial accelerometer which is capable to send raw or frequency information of its three axis via Serial Peripheral

Interface . The functionalities presented on it were used as a reference in the development of some parts of this work.

1.1 GENERAL GOALS

Develop an embedded software that allows to perform digital signal processing inside a microcontroller.

1.2 SPECIFIC GOALS

The specific goals are presented below:

- Choose a microcontroller which is able to handle the proposed tasks.
- Find or develop tools which are capable to do a signal processing in the chosen microcontroller.
- Find or create tests which validate the results obtained.

The requirements needed to achieve the specific goals are better described in the methodology.

1.3 STRUCTURE

This work is divided in 6 main sections as follow: Introduction, Theoretical Background, Methodology, Development, Results and Conclusions. A brief explanation for better orientation during the text is presented below;

The first chapter, Introduction [1], presents the theme and introduces the problem to be solved. It brings the reasons of this work and also contextualizes it in a point of time in the history of the technology development, as well as the background that made it possible.

Following with Theoretical Background [2], some technical concepts behind the key points of the whole project are concisely introduced in order to guarantee a better comprehension of it.

On its third part, the Methodology [3], all the requirements of the system, as well as the main tools utilized to test and develop it are presented. In this chapter, the methods available that can be used to solve the problem are presented, along with the ones which were actually chosen and the explanation for it.

In the course of Development [4], it is discussed how all the mechanisms mentioned in the previous chapter were connected to become the solution of the purposed problem.

Then, in Results [5], all the outputs obtained from the program are evaluated and put together in different ways for comparison.

Finally in Conclusions [6], considerations over the final project are made, what could and could not be done, its main limitations, and what are the improvements and next steps for future works.

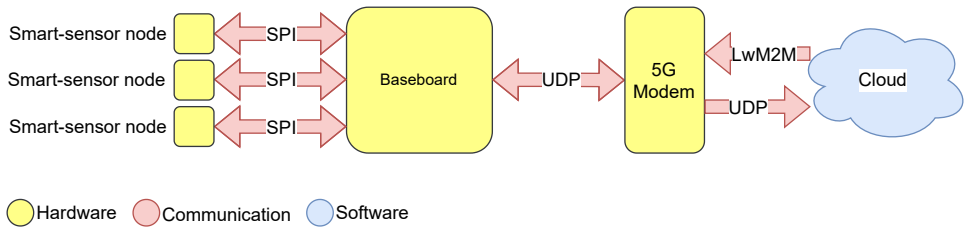
2 THEORETICAL BACKGROUND

All the technologies used, as well as the concept behind it, are explained in this chapter to provide a better comprehension of the following work and its proposal.

2.1 THE SYSTEM

The MSP is thought to be a base system where is possible to connect multiple diverse sensors in its sensor nodes, in an easy way (SCHMITT et al., 2020). A scheme is presented in the Figure 4 below for a better comprehension.

Figure 4 – Basic concept behind the Multi-Sensor Platform



Source: Autor

In the figure above it is possible to have a better look inside the system. The sensors, which are placed on the work piece inside the machine, are connected to the smart-sensor nodes – which are the focus of this works. This nodes samples and pre-process the data of the sensor, and then it send the results via SPI to the baseboard. The baseboard gathers, packs and sends the data of all the nodes through to the 5G modem, which transmits the data to the cloud. It is possible to connect to the cloud using a Machine to Machine protocol, in which its lightweight version (LwM2M) sends commands to the baseboard, to choose among the features available.

2.1.0.1 System tools

As mentioned before, one of the applications which the system will be used is the tool breakage detection. For this purpose, two tools are used. The first one is the Fourier Transform, which divides a signal in its components according to theirs corresponding frequencies, in order to tell which of this components contributes more to the amplitude of the signal. The second one is the Wavelet Transform, which solves a limitation of the fist method. Since it is localized in time – different from the Fourier Transform – it does not only separate the signal in its frequency components, but is is also able to tell when exactly each amplitude happened. In this way, the Fourier Transform can tell which frequency contributed more for the breakage, while the Wavelet

Transform is able to show when it happened (LI; DONG; YUAN, 1999).

2.2 FOURIER ANALYSIS

As it can be seen in Oppenheim e Willsky (1996), Fourier has shown that any periodic signal in continuous time can be described as a linear combination of sines and cosines, or as a sum of complex exponential, since both of them are connected via Euler's formula [1]. The former one is introduced in the Equation 1 below:

$$a_n(t) = \sum_{k=-\infty}^{\infty} a_k e^{jk\omega_0 t} \tag{1}$$

Where ω_0 is the fundamental frequency of the signal, and j is the complex variable. The coefficients a_k are called harmonics and can be calculated by the Equation 2:

$$a_k = \frac{1}{T} \int_T x(t) e^{-jk\omega_0 t} dt \tag{2}$$

Analyzing the Equations 1 and 2, it is possible to realize that a signal can be described in therms of an amplitude and a frequency that is multiple of its fundamental frequency.

2.2.1 Fourier Transform

A non-periodic signal $x(t)$ can be described as a periodic signal $\tilde{x}(t)$ in which the period tends to infinity. Since the fundamental frequency is defined as $\omega_0 = \frac{2\pi}{T}$, where T is the period of the signal, when ($T \to \infty$) – as it can be seen in Oppenheim e Willsky (1996) – the Fourier Transform is defined by the resulting Equation 3 below.

$$X(j\omega) = \int_{-T/2}^{+T/2} x(t) e^{-j\omega t} dt \tag{3}$$

The $X(j\omega)$ is a frequency domain complex evaluated function. It is the integral of the aperiodic signal $x(t)$, from its start ($\frac{-T}{2}$) to its end ($\frac{T}{2}$), multiplied by the complex exponential $e^{-j\omega t}$, where ω is defined as $\frac{2\pi}{T}$ and j is the complex unit.

The Fourier Transform transposes a time defined function to the frequency domain. It translates signals, which might have a frequency that is hard to analyze due to its many components, to a simpler function that relates amplitudes with the frequencies that they occur. Since most of signals in nature are non-periodic, the Fourier Transform becomes a particular useful tool in most of digital signal processing problems.

[1] On it's polar form: $e^{ix} = sinx + icosx$

2.2.1.1 Magnitude and Phase

As stated in Oppenheim e Willsky (1996), since the result of the Fourier Transform is a complex number, as it shown in Equation 3, in order to achieve a an easier analysis by plotting, the magnitude an phase can be obtained. The magnitude is given by the Equation 4 below:

$$|X(j\omega)| = \sqrt{X(j\omega)^2} \tag{4}$$

This equation describes which frequencies contribute more to the amplitude of the signal and it is totally real evaluated. On the other hand, the Equation 5 below, defines the phase of a signal.

$$\measuredangle X(j\omega) = -tan^{-1}(\frac{b}{a}) \tag{5}$$

Considering that $X(j\omega)$ is a complex number – and that b involves the frequency ω – the equation above describe the phase of the signal analyzed and it is also real evaluated.

2.2.2 Discrete Fourier Transform

Taking a non-periodic discrete signal $\tilde{x}(t)$, Oppenheim e Schafer (2010) show that the Fourier Transform in this case, represented in Equation 6, is analogue to the Equation 3 for discrete signals.

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-\frac{j2\pi kn}{N}}, \quad k = 0, 1, 2, 3, ..., N-1. \tag{6}$$

The above equation is called Discrete Fourier Transform(DFT), and it transposes a sequence of equally spaced samples in time to a sequence that has the same characteristic in the frequency domain. This representation is important in digital signal processing, since digital systems only work with sampled signals – because analog signs are infinite, but memory is not – while the Fourier Transform is defined in a continuous time scenario(OPPENHEIM; SCHAFER, 2010).

2.2.3 Fast Fourier Transform

The DFT can be calculated also for complex signals, since both k and n may be complex in Equation 6. Considering the input signal $x[n]$ complex, it would be needed N complex multiplications and $(N-1)$ complex addictions in order to calculate the entire DFT. This would lead to a complexity of N^2 mathematical operations, where N is the number of samples in the input signal.

The implementation of the operations of a DFT – as it is presented in the Equation 6 – would then lead to a algorithmic complexity of $O(n) = n^2$, which can be

costly, mainly for larger data sets. This number can be considerably reduced if some advantages, such as symmetry and periodicity, of this series are taken.

In order to achieve a better computational efficiency, a variety of algorithms to calculate the DFT were created through the time. This set of algorithms, which can reach a complexity of $O(n) = n * logn(n)$ are called the Fast Fourier Transform (FFT)(OPPENHEIM; WILLSKY, 1996).

2.3 WAVELET TRANSFORM

According to Daubechies (1992), the Wavelet Transform is a tool that cuts up functions into different frequency components, and then analyzes each component with a resolution matched to its scale. It can be defined by the Equation 7 below:

$$T_f^{wav}(a,b) = \frac{1}{\sqrt{|a|}} \int f(t)\psi\left(\frac{t-b}{a}\right)dt \tag{7}$$

Where a is the dilation or scale parameter, which stretches or shrinks the wavelet; and b is the translation parameter, which shifts the wavelet through the time. (DAUBECHIES, 1992)

The $\psi(\frac{t-b}{a})$ generates an entire family of functions, the wavelets, by ranging the coefficients a and b over the real numbers domain. The function $\psi(t)$, sometimes called mother wavelet, is a completely differentiable function and it must satisfy the condition:

$$\int \psi(t)dt = 0 \tag{8}$$

where $\psi(t)$ might be a complex function, and then the complex conjugate $\psi^*(\frac{t-b}{a})$ must be used in the Equation 7 instead.

2.3.1 Discrete Wavelet

The Discrete Wavelet Transformation assumes the coefficients $a = a_0^m$ and $b = nb_0a_0^m$, where m,n range over $\mathbb{Z}$, and $a_0 > 1$, $b_0 > 0$. The appropriate choices for a_0 and b_0 depends on the wavelet ψ(DAUBECHIES, 1992).
The equation that describes a wavelet becomes:

$$\begin{aligned}\psi_{m,n}(x) &= a_0^{-m/2}\psi\left(\frac{x - nb_0a_0^m}{a_0^m}\right) \\ &= a_0^{-m/2}\psi(a_0^{-m}x - nb_0)\end{aligned} \tag{9}$$

The equation 7 will then become:

$$T_{m,n} = \int f(t)\psi_{m,n}(t)dt \tag{10}$$

As shown by Daubechies (1992), with this discrete description of the Wavelet Transform, it is possible to derive equations that are the synthesis and analysis of sub-band filtering scheme with exact reconstruction. It is important not only because of the certainty that the original signal can be reconstructed, but also for the possibility to derive an algorithm which is simple to compute.

3 METHODOLOGY

In order to achieve the proposed goals, the fist step was to describe the system and all its requirements. Later, in order to satisfy the conditions listed, a market comparison was conducted to find the microcontroller that best suits the solution. All these steps about the materials used and which methods has been followed are better described in this chapter.

3.1 REQUIREMENTS

Since the smart-sensor platform is designed to be applied in the acquisition and analysis of tool machinery data, certain requirements must be fulfilled by the microcontroller which will be on its node.

- **Functional**
  - The microcontroller must have an Floating Point Unit(FPU) unit.
  - The node must be able to communicate with the base-board via SPI communication
  - The software must be able to do all its calculations and processing, as well as send its results, during the time between one Analog-Digital Converter(ADC) sample and another.
  - The software must be able to compute the magnitudes and the phases of the Fast Fourier Transform.
  - The software must provide a Peak Detection feature, of at least three peaks.
- **Non-functional**
  - The data must be sampled with at least 16 bits of precision
  - The data must be sampled in a range between 20 and 40 kHz
  - The software must have a Wavelet Transformation.

In summary, the system needs to be able to do all the pre-processing over floating data, while respecting the sampling rate constraint, and send it via SPI to a central unit which will forward it to the internet.

3.2 THE HARDWARE CONSTRAINT

The vibration sensor used int the MSP is the VS-JV10 from Kemet (2019), shown in Figure 5 below. Considering the harsh conditions of the application, the VS-JV10 suits the machinery environment due to its waterproof and oil resistant isolation

Figure 5 – Sensor VS-JV10 from Kemet

Source: Kemet (2019)

The VS-JV10 can detect vibrations in a range of 10Hz to 15kHz throughout a large sensing bandwidth, which makes it ideal for predictive maintenance and process control operations in Industrial Internet of Things (IoT). (KEMET, 2019).

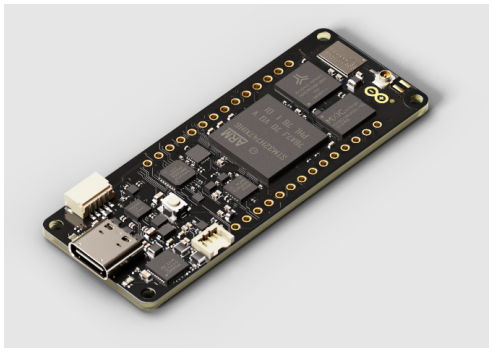
3.3 MICROCONTROLLER

The microcontrollers available in the laboratory were compared to develop the system based on the aforementioned requirements. Their main features and developing boards are presented below.

3.3.1 Arduino Portenta H7

The Arduino Portenta H7 is based on a STM32H747XI, a dual-core low-power processor with an on-chip Graphical Processing Unit (GPU) integrated on it. An image of the board can be seen in the Figure 6.

Figure 6 – Development board Arduino Portenta H7



Source: Arduino (2020)

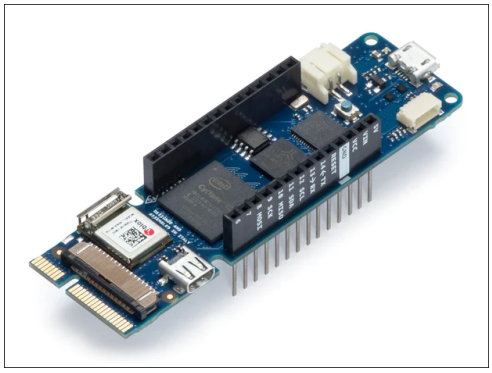
The available program interfaces are: Arduino (on top of the Mbed OS), MicroPython / JavaScript via an interpreter, TensorFlow Lite and native Mbed Applications. (ARDUINO, 2020)

3.3.2 Arduino MKR Vidor 4000

Combining Intel Cyclone 10CL016 with an SAMD21 low-power microcontroller, the Vidor brings the flexibility and process power of an FPGA, enabling the possibility to

design entire digital circuits, even for enhancing the number of serial communications. The developing board is presented in the Figure 7 below.

Figure 7 – Development board Arduino MKR Vidor 4000



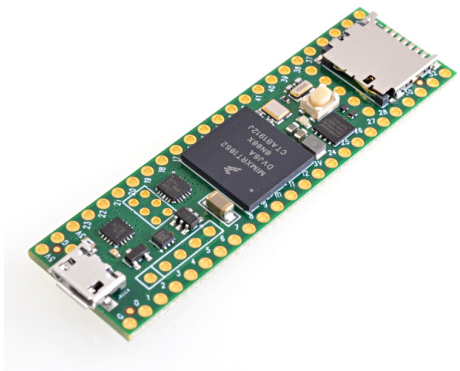
Source: Arduino (2018)

It may be programmed using the Arduino IDE for the processor and the Quartus software for the digital circuit implementation. (ARDUINO, 2018).

3.3.3 Teensy 4.1

With a small prototyping board, the Teensy 4.1 is based on a i.MX RT1060, a powerful microcontroller which is capable to support video and audio applications and provides various memory and communication interfaces. The board is presented in Figure 8 below.

Figure 8 – Development board Tensy 4.1



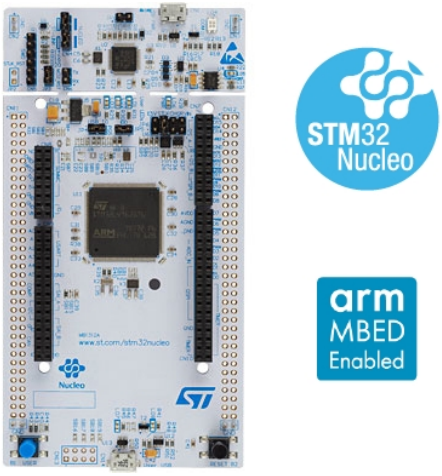
Source: PJRC (2020)

Its processor relays on some special features, such as Dual Issue Superscaler Architecture, Tightly Coupled Memory, DSP instructions and Branch Prediction. The programming interfaces available are Arduino and CircuitPython. (PJRC, 2020)

3.3.4 NUCLEO-L4R5ZI

Based on an ultra-low-power and high performance RISC core, the NUCLEO-L4R5ZI is the most standard model on this list comparing to the traditional microcontrollers. It is shown in the Figure 9 below.

Figure 9 – Development board NUCLEO-L4R5ZI



Source: STMicroelectronics (2019)

Despite of the only languages available to program being C and C++, it has at its disposal a series of useful libraries and frameworks, such as HAL, CMSIS, LWIP, FatFS and FreeRTOS.

3.3.5 Comparison and choice

A summary of the main features which were taken in considerations can be seen in the Table 1 below:

Table 1 – Table evaluating the possible microcontrollers

	Microcontroller			
Feature	Portenta	Vidor	Teensy	Nucleo
Architecture (Cortex)	M7+M4	M0+	M7	M4
Clock (MHz)	480+240	48(Up to 200)	600	120
RAM (MB)	8	8	8	0.64
FLASH (MB)	16	2	1	2
SPI	6	1(Up to 7)	3	3
ADC	3	n/a	2	1
ADC resolution (bits)	16	n/a	10	12

Source: Author.

Although some of these microcontrollers have high speed chips or even two cores, the Arduino interface or other high level languages could become a issue with the sampling or the memory used by it. The lacking of a base material and a community of developers also influences the choice. Taking all the aforementioned conditions into

consideration, the NUCLEO-L4R5 was chosen to be the microcontroller where the system would be built on, whic despite of its memory being much lower than the others, it still suits for the application.

3.4 CMSIS LIBRARY

The Common Microcontroller Software Interface Standard(CMSIS) is an open-source abstraction layer for microcontrollers. It provides a series of middlewares, peripheral interfaces and real-time operating systems API to Arm Cortex Processors(ARM, 2019)

3.4.1 CMSIS-DSP

. The CMSIS-DSP is the branch of the CMSIS library for digital signal processing. It supplies functions of many mathematical functions for microcontrollers - such as linear algebra, numerical analysis and complex numbers operations - present for integer, fixed and floating point units.

3.4.1.1 Real Fast Fourier Transform

As shown previously on the Chapter 2, the FFT is an algorithm to calculate the Fourier Transform over a sequence of complex numbers. Therefore, since the ADC output is a series of unsigned integers, the input data in most digital signal processing applications using embedded systems is real. Hence, to compute an FFT, it would be necessary either a method to convert the sample data to the complex domain or to calculate a the Fourier Transform in a data set that is real.

According to Arm (2019), the Real Fast Fourier Transform(RFFT) is an algorithm that takes advantage of the symmetry properties of the FFT. The real sequence is initially handled as if it were complex and then later a processing stage reshapes the data to obtain half of its frequency spectrum in complex format. The resulting data is a real array that represents a complex number, where the real and the imaginary part are interleaved, starting with the fundamental frequency on the third position of the array. The first position represents the DC offset, and the second is always 0(ARM, 2019).

3.5 GSL LIBRARY

The collection of free software GNU is Not Unix (GNU) has an entire branch for of C routines for numeric computing. The GNU Scientific Library (GSL) is available as an Application Programming Interface (API) format and its source code is distributed under the GNU General Public License, which guarantees the four freedoms to run, study, share, and modify the software. (GNU, 2021)

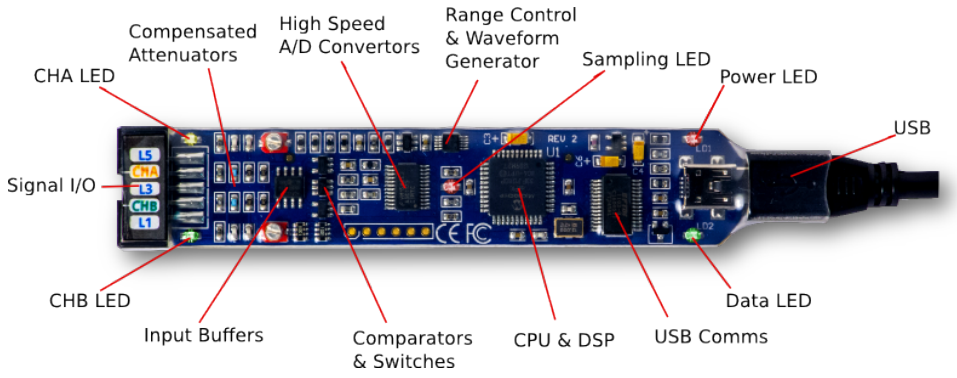
3.5.0.1 Wavelet Transform

The library provides an entire group of functions to calculate 1D and 2D digital waveletes. It abstracts the digital wavelet transformation into simple multiplications of coefficients, which are stored as constants. This makes very easy to port another families of wavelets, besides the ones already available.

3.6 BITSCOPE

During the testing, it is necessary an instrumentation which is capable of supply input data, as well as analyze the corresponding output. For generating the input signal and to execute all the hardware validations, a BitScope BS05 as shown in Figure 10 was used.

Figure 10 – Digital oscilloscope BitScope BS05U



Source: BitScope (2022)

According to BitScope (2022), the BitScope is a fully featured mixed signal test and measurement system, which can be used as a digital oscilloscope, logic analyzer and function generato at the same time. The device is connected via USB to the computer, as it can be seen in Figure 10, which controls the mode being used and shows the data being sampled. The device relies on a bandwidth of 20 MHz and its logic capture can reaches up to 40 MSps .

3.7 TESTING

In order to check the feasibility of the proposed solution, it is necessary to test the correctness of features calculation, as well as if the results can be delivered within the stipulated time. The structure of this procedures and how they have been executed are described below.

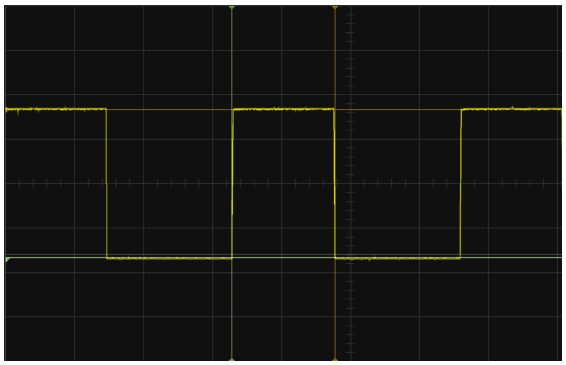
3.7.1 Time testing

Two types of tests were planned to evaluate the execution time for the feature calculation – via software and via hardware – which are better explained below. All the tests were done using 1024 samples, which were converted to 32 floating-point unit values. For the timing tests, a 10 kHz noisy sine wave array provided by the CMSIS-DSP library was used (ARM, 2019).

3.7.1.1 Time testing through hardware

As shown by Stewart (2006), a logic analyzer – in this case a digital oscilloscope – is one of the best tools for an accurately execution time with microsecond resolution. One of the approaches listed by him is to send strategic signals to an output port, which are read by the logic analyzer as events. Based on this procedure it was possible to derive the method presented in Figure 11, which was applied to measure features that had no parameter variations – namely magnitude and phase of the FFT.

Figure 11 – Oscilloscope image of a GPIO toggle every ADC conversion.



```
//GPIO High
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_SET);

//FFT calculation
arm_rfft_fast_f32(&fft_inst,fft_input,fft_output,0);

//Condition of the RFFT
fft_output[0] = 0;
fft_output[1] = 0;

//Magnitude calculation
arm_cmplx_mag_f32(fft_output,fft_feat, N_SAMPLES);
//GPIO low
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_RESET);
```

Source: Author

The method depicted above consists in setting a GPIO to high (A4 in the figure), executing the procedure to be evaluated – in this case, the FFT – and then setting the GPIO to low again. The pin is analyzed during the whole action by an oscilloscope, in order to measure its toggle time, which will the equal the time elapsed during the whole function execution.

3.7.1.2 Time testing through software

The testing through software is based on a timer method described by Stewart (2006). It consists in reading a timer counter value at the beginning and at the end of the code. The elapsed time is then given by the difference of the two samples. The code shown in Listing 3.7.1.2 below uses a 10MHz timer with a 16 bit values, which can count up to 65.535 microseconds.

```
1 //All Daubechies implemented coefficients
2 const uint16_t coeff[N_COEFFS] = { 4, 6, 8, 10, 12, 14, 16, 18, 20};
```

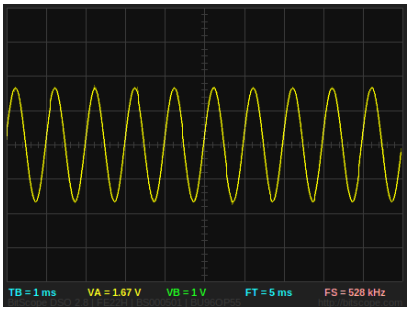
```
3 //daubechies wavelet
4 const gsl_wavelet_type *wavelet = gsl_wavelet_daubechies;
5
6 volatile uint32_t delta_t = 0;//variable for time elapsed calculation
7
8 //for loop though Daubechies coefficients
9 for( uint8_t i = 0; i < N_COEFFS; i++)
10 {
11     //copies the data from another file to input array
12     arm_copy_f32(testInput_f32_10khz, input, N_SAMPLES);
13
14     delta_t = __HAL_TIM_GETCOUNTER(&htim1); //reads the time counter
15
16     //initialize the wavelet with the right coefficients
17     gsl_wavelet_init(&w,wavelet, coeff[i]);
18
19     //executes the wavelet transformation
20     gsl_wavelet_transform_forward(&w, input, 1, N_SAMPLES, &work);
21
22     //calculate the time elapsed
23     delta_t = __HAL_TIM_GETCOUNTER(&htim1) - delta_t;
24 }
```

In the snippet above, all the Daubechies wavelets family implemented is tested at once. The value of the time elapsed can be easily obtained via debugging, what makes it easy to plot.

3.7.2 Precision testing

After the time constraints were tested, the author developed a method aiming to prove the accuracy of the features implemented. In order to make it closer to a real application, it was decided to use the ADC instead of a fixed input. The signal generated to test the precision of all the features is demonstrated in Figure 12.

Figure 12 – Oscilloscope image of the 1 kHz sine wave generated as input.



Source: Author

The planned test consists in using the Universal Synchronous/Asynchronous Receiver-Transmitter (USART) to send data, first the floating-point converted signal and

the later the result of the evaluated feature, to the computer. Both of the outputs were stored in different CSV files, which could be loaded by a python program. The input results were them read by a script and analyzed either through the FFT module of the library NumPy from Harris et al. (2020) or using the library PyWavelets from Lee et al. (2019), and also saved in a file. Hence, the both methods to calculate the required feature could have their results easily analyzed point by point.

In order to standardize the tests, the ADC was configured as one shot and the frequency used was around 16,7 kHz. The input was a 1 kHz sine wave generated by the digital oscilloscope/function generator. The USART baudrate was 115200 and the STM debugger was used to control its output (STMICROELECTRONICS, 2019).

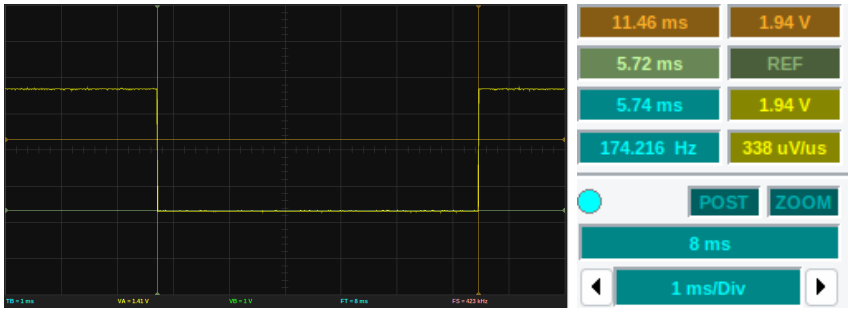
4 DEVELOPMENT

After having listed all the techniques and tools needed to design the prototype of the code, this chapter approaches how everything comes together to design the software. Here are presented the ADC configurations and all the algorithms used in the project of the software.

4.1 ADC

In order to obtain an optimal resolution of the reconstructed signal, the sampling rate was chosen to have around 10 times the upper frequency boundary of the sensor introduced in Chapter 3. As shown in Figure 13, the closest frequency above this limit for a sampling of 1024 points was around 178kHz.

Figure 13 – Oscilloscope image of a GPIO toggle every ADC conversion.



Source: Author

The signal generated on the Figure 13 above was made toggling a GPIO inside the interruption of a complete conversion using the Direct Memory Access of the ADC. Hence, the signal only emits an edge after 1024 were read, which means that the actual sampling rate is calculated by multiplying the frequency showed on the Figure 13 by the number of points acquired, resulting in around 178kHz.

4.2 FFT

As mentioned in Chapter 3, for the FFT the CMSIS library was used. Taking in consideration what was mentioned before, there were no much alterations needed in this part. The snippet of code in the Listing 4.3 below gives an example of how to properly use the library.

```
arm_rfft_fast_instance_f32 fft_inst;

float32_t input[N_SAMPLES], output[N_SAMPLES], magnitudes[N_SAMPLES/2];

arm_rfft_fast_f32(&fft_inst, input, output, 0);
output[0] = 0;
output[1] = 0;
arm_cmplx_mag_f32(output, magnitudes, N_SAMPLES);
```

First, an instance of the structure of the transform and the algorithm being used must be declared - in this case a fast RFFT of a 32 bits floating-point unit. Then, 3 arrays are declared: the one that stores the input, the one which will store the output and the one to be used to store the magnitude. After that, the FTT algorithm is applied using the struct instance, the input and the output. Since it is an in-place algorithm - which means that it also changes its input - as the output is done, the input is also changed and can not be used anymore. In the end, the so called ”output” is used to calculate a complex magnitude, which expects an interleaved (real[0], imag[1], real[2],...) fashion array as an input, that returns its results in its second parameter(magnitudes). It is important to notice that, due to the nature of this last procedure, the second array must have half of the size of the first one, which must have the same size that is passed as the third parameter.

4.3 THE PHASE ALGORITHM

The phase algorithm is based on a code, which can be seen in 6.2, that first appeared on Apple (2005), a mail exchange among Apple developers, where some approaches for making an *atan2* function in C which was faster than the one already present in the language can be seen. The adaptation from one of these codes is exposed on the Listing 4.2 below

```
#define MY_PI     3.141592f
#define MY_PI_2   1.570796f
#define MY_PI_3_4 2.356194f
#define MY_PI_4   0.785398f

static inline float32_t
fast_atan2(float32_t y, float32_t x)
{
    float32_t r, angle;
    float32_t abs_y;
    arm_abs_f32(&y, &abs_y,1);
    abs_y += 1e-10f;

```

```
14      if ( x < 0.0f )
15      {
16          r = (x + abs_y) / (abs_y - x);
17          angle = MY_PI_3_4;
18      }
19      else
20      {
21          r = (x - abs_y) / (x + abs_y);
22          angle = MY_PI_4;
23      }
24      angle += (0.1963f * r * r - 0.9817f) * r;
25      if ( y < 0.0f )     return ( -angle );
26      else return ( angle );
27  }
```

The algorithm is an improvement based on Iowegian International Corporation (1994), and the maximum error is mentioned to be around 0.0102 radians. In order to be feasible for a microntroller and the application itself, all the floating point variables and constants were changed to be accordingly to the Arm Math library.

4.4 PEAK DETECTION

Although it has its minor modifications to best fulfill the requirements while being more generic, the basis for this algorithm was the classical and simple maximum value finding inside an array. The implementation is shown in Listing 4.4 below.

```
1  void peak_detection( float32_t *magnitudes, float32_t *peaks, uint16_t *
       index, uint16_t low, uint16_t high, uint16_t n_peaks, uint16_t size )
2  {
3      if( !magnitudes || !peaks ) return;
4      if(high > size || low > high ) return;
5
6      uint16_t n_max = 0;
7
8      for(int i = low; i < high; i++)
9      {
10         peaks[i-low] = magnitudes[i];
11         index[i-low] = i;
12         if(n_max++ < n_peaks)
13         {
14             for(int j = low; j < high; j++)
15             {
16                 if(magnitudes[j] > peaks[i-low])
17                 {
18                     peaks[i-low] = magnitudes[j];
19                     index[i-low] = j;
20                 }
```

```
21                 }
22                 magnitudes[index[i-low]] = -4;
23             }
24             else break;
25         }
26 }
```

Despite the resemblance with the maximum value finding algorithm, the code above relays on some features specially thought for the application. Among its parameters, it is possible to check inside a certain frequency band – between *low* and *high* – instead of the whole set of samples. There is also an option, when the array *index* is not invalid, to get the indexes of the peaks, which allows to reconstruct which frequency had those magnitudes. In summary, the code runs through an array *magnitudes* of size *size*, from *low* to *high*, and looks for *n_peaks* inside of it, which will be returned inside *peaks*. It will also return the indexes of those peaks inside *index* if it is valid. It is important to notice that, after a peak is found, the value in the original array is changed to -4, changing its original data. The reason for this is to make it possible for the algorithm to find other peaks than the first one. The choice of the value was done considering that the algorithm, although is not common, might be used to find peaks in the phase too, most likely in another applications. Since the *atan2* returns a value between $-\pi$ and π, the chose value to represent an error was the closest integer value which was bigger than the maximum result possible.

4.5 WAVELET ALGORITHM

Since the GSL was originally written to be a numerical library for C programs, some considerations had to be made when adapting it to a microcontroller. For example, due do the changing of nature of the application, the encapsulation was broken on behalf of the memory, so all the allocations could be sparred. The precision of all constants were changed from double to single floating-point unit, while the unused ones were removed. As the microcontroller is sampling an one-dimensional side and only doing the forward transformation, the 2D Wavelets as well as the inverse transformations were also deleted.

4.6 THE MAIN CODE

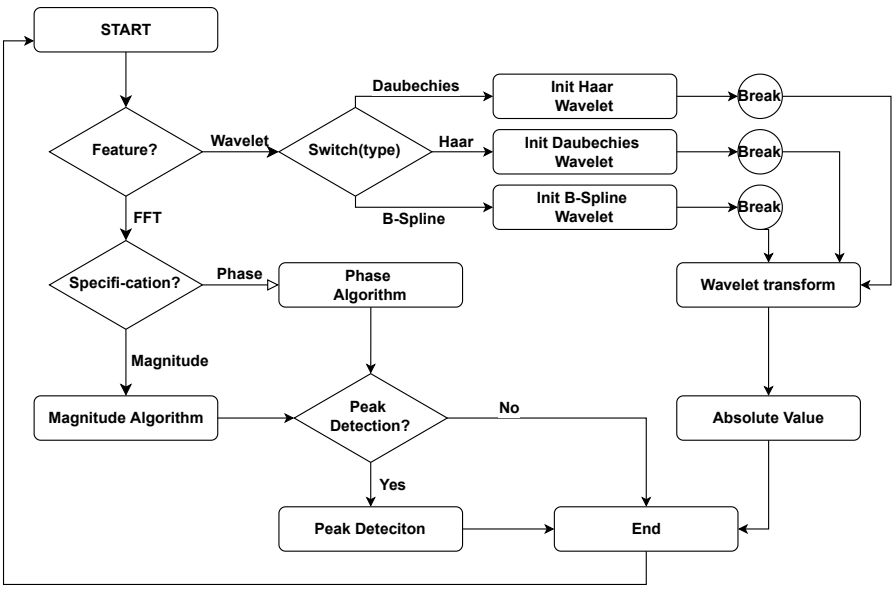
The algorithm, which is partially present in the Appendix 6.2, starts initializing all the peripherals and all the structures needed to the Fourier and Wavelet transform. The ADC uses the Direct Access Memory technology, which means that the data collected goes directly to memory, without going through the CPU (TANEMBAUM; BOS, 2014).

The STM32 Nucleo-L4R5 provides two interruptions to deal with the ADC buffer in the DMA mode, for when its half or completely full. The interrupts call a callback that were used to convert the sample from bytes to voltage level, as it can be seem in the Listing 4.6 below.

```
1 void HAL_ADC_ConvHalfCpltCallback(ADC_HandleTypeDef *hadc)
2 {
3     for(int i = 0; i < 512; i++)
4     {
5         fft_input[i] = 3.3f *  ((float32_t)adc_buffer[i])/0x0FFF;
6     }
7 }
8 void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef *hadc)
9 {
10    for(int i = 512; i < 1024; i++)
11    {
12        fft_input[i] = 3.3f *  ((float32_t)adc_buffer[i])/0x0FFF;
13    }
14    cpt_flag = 1;
15 }
```

After the ADC completes its cycle, is sets a flag to warn the main loop that the array is full and the processing operations can be applied. The ADC restarts again (circular mode) and the main sets the complete flag to 0 after it is done with the processing. The processing part done by main after the complete acquisition and converting of the data can be better visualized in the flowchart shown in the Figure 14 below.

Figure 14 – Flowchart of the processing part of the algorithm



Source: Author

The code starts checking which transform was chosen, the Wavelet or the

Fourier. If the former has been chosen, then it is checked if the magnitude or the peak feature should be extracted from the data, and then if a peak detection must be done. On the other the hand, if the latter has been chosen, it checks which wavelet it must be used, then it initializes it properly, then executes a wavelet transform and an absolute value trough the whole data array. It is important to notice, that despite of it is not in the Figure 14, the correct family of the wavelet must be chosen t for the software to function accordingly. It is not show in the flowchart above, because it is not checked by the main code, but by the wavelet, not executing the wavelet when it has a wrong value.

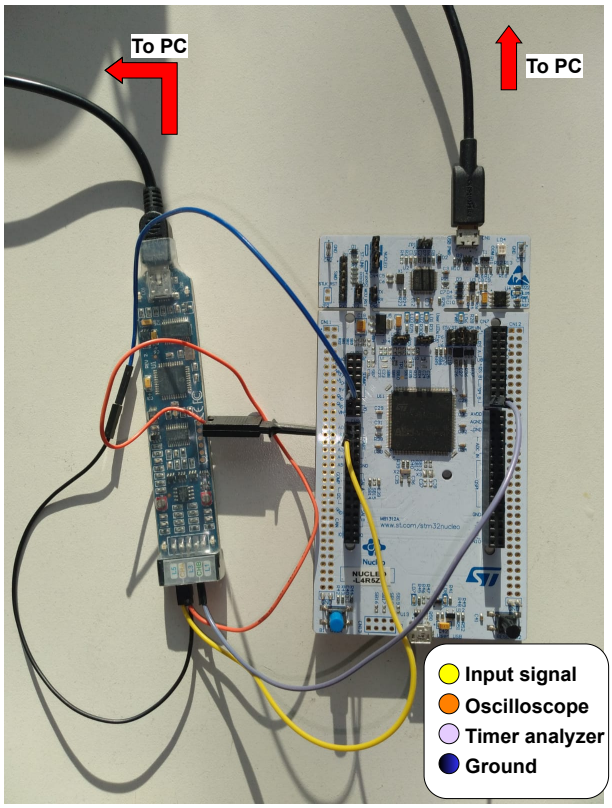
5 RESULTS

The results obtained by experimentation are exposed here in this chapter. At the end of this section, the requirements presented in the Chapter 1 are scrutinized to evaluate in which extension they have been fulfilled.

5.1 SETUP

For a fairness purpose all the timing presented here were done using a constant array with the data of a noisy sine wave of 10kHz as input. The clock frequency was 120MHz and the ADC had its sample rate around 178kHz as mentioned in Chapter 4 before. An image of the equipment set up to generate this results is shown in the Figure below.

Figure 15 – Setup used to obtain the results.



Source: Author

As can be seen in the Figure, both USB cables, from the oscilloscope and from the microcontroller, are plugged to the computer in order to send and receive data. According to the captions, the yellow wire is the output of the function generator (input data of the microcontroller), the orange one is the oscilloscope probe, the grayish purple is the logic analyzer for the timing test, while the blue an black one is the ground wire of

the system.

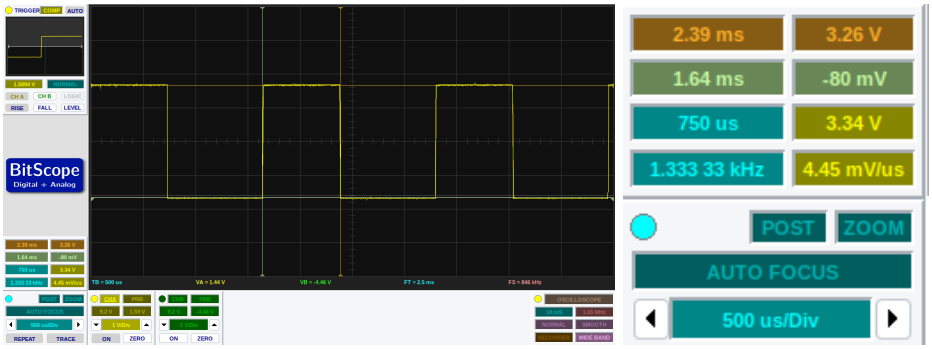
5.2 TIME CONSTRAINTS

In order to be considered feasible, all the features available on the system must fit inside the time window between two ADC samples. So, as shown in Chapter 4, every computation must be inside the 5,74 milliseconds window shown in Figure 13.

5.2.1 Fast Fourier Transform

Since the Fourier Transform alone does not give much information about the system, the RFFT timing analysis was done either with the magnitude or the phase calculation. The time results for an RFFT and the complex magnitude algorithm are shown in the Figure 16 below.

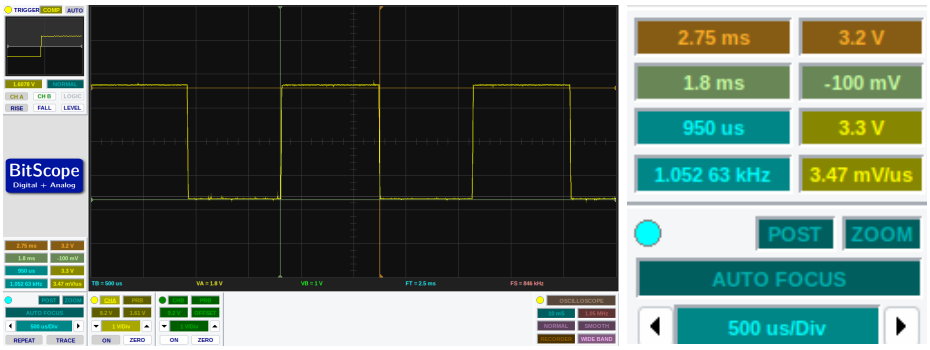
Figure 16 – Oscilloscope image of a GPIO toggle with the FFT and peak calculation in between.



Source: Author

It is possible to see above that the FFT with the magnitude calculation takes about 750 microseconds to be done, which fits inside the time window between the samples. The timing results for the phase algorithm can be seen in the Figure 17 below.

Figure 17 – Oscilloscope image of a GPIO toggle with the FFT and phase calculation in between.



Source: Author

As it shown by Figure 17, the peak algorithm takes 950 microseconds to compute, which is also inside the ADC samples time window. Although both of them

meet the requirement wide gap, there is still a significant difference of more than 25% between them. It could be due to two things: the difference between the nature of the operations performed by the two features; and also because the phase algorithm is an Atan2 algorithm adapted and applied two by two in the input array, while the magnitude algorithm belongs to the ARM library.

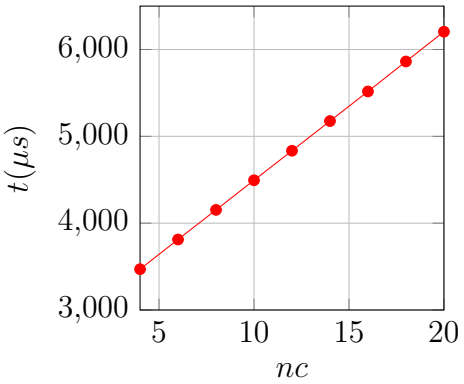
5.2.2 Wavelets

Due to the considerable number of wavelets present in this work and the different nature among them, some results are presented as images of an oscilloscope - in the cases where the wavelets do not have a variety of vanishing moments - while others are shown in a plot.

5.2.2.1 Daubechies

In the Figure 18 below, it is possible to see the correlation between the number of coefficient used to calculate the Daubechies Wavelet and the time to compute them.

Figure 18 – Relation between the value of N and the time spent in microseconds to compute the wavelet.



Source: Author

As it can be seem in the plot above, the time spent in the wavelet transform varies linearly with the number of coefficients used. Using 16 coefficients, the microcontroller takes 5,5 ms, and 5,8 ms in case of 18 coefficients are being used. Despite of the calculation with 16 values are still inside the time window, it is too close to its deadline. So, in the case where another function would be needed for any other reason, maybe this tight space could not be enough for it.

5.2.2.2 B-Spline

Since the B-Spline wavelets varies with i and j, besides the number of coefficients too, its values are better shown in a table as in the Table 2 below.

Table 2 – Table relating the B-Spline families, the number of coefficients and the time in microseconds to compute each one of them

i,j	1,3	1,5	2,2	2,4	2,6	2,8	3,1	3,3	3,5	3,7	3,9
nc	6	10	6	10	14	18	4	8	12	16	20
μS	3812	4494	3812	4494	5177	5859	3471	4153	4835	5518	6204

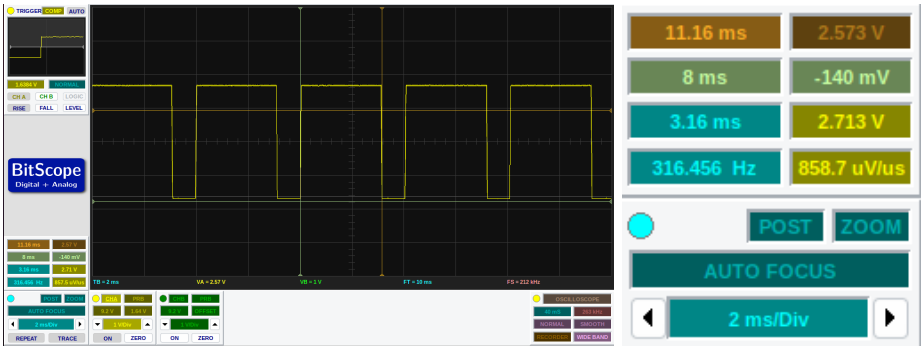
Source: Author

It is possible to see above that, every pair of i, j uses a specific number of coefficients, and that the time variation is pretty similar to the Daubechies Wavelets results.

5.2.2.3 Haar

Since the Haar wavelet family only has one wavelet, the visualization through the oscilloscope was the most appropriate one. The timing results for the Haar Wavelet are presented in the Figure 19.

Figure 19 – Oscilloscope image of a GPIO toggle with the Haar wavelet calculation in between.



Source: Author

As it can be seen above, it takes 3,6 ms to compute a Haar Wavelet transform, which meets the time constraints previously imposed.

5.2.3 Precision

For the system to work, not only the time constraints must be satisfied, but also the results of the operation need to reach a minimum precision that satisfies the requirements of the application. The results of the accuracy tests mentioned in Chapter 3 are presented below.

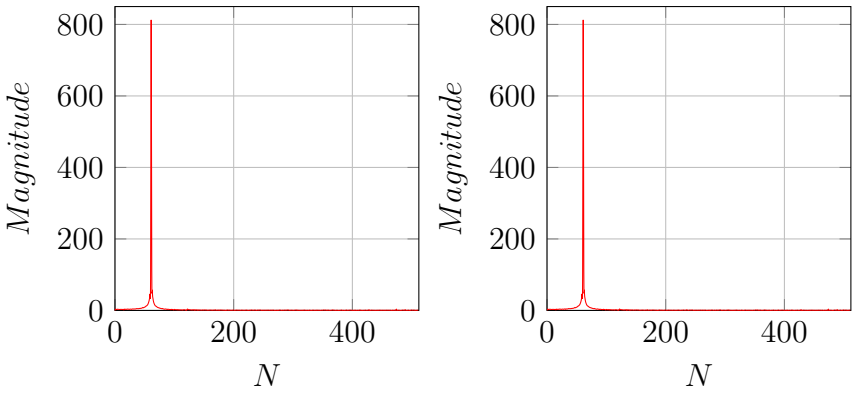
5.2.4 FFT

As mentioned before, the FFT alone does not show much information about the signal. In order to check the precision of the microcontroller features, only the magnitude and the phase were compared, and not the FFT itself.

5.2.4.1 *Magnitude*

Since it is a normal sine wave, the expected output for the magnitude is a lonely peak in a specific frequency. The results obtained are shown in the Figure 20 below.

Figure 20 – Comparison between the FFT implemented in the microntroller and the one in python for the same input. Left: Microcontroller. Right: Python.



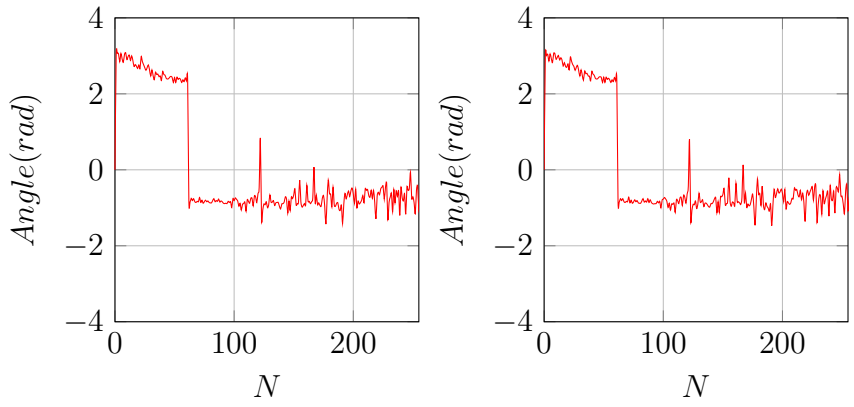
Source: Author

As mentioned above, the result is a peak between the samples 61 and 62. Considering that the sample rate was around 16,6 kHz, it corresponds to a frequency band between 993,5 and 1009,77 Hz both in Python and in the microcontroller. The obtained results are consistent with the 1 kHz input.

5.2.4.2 *Phase*

Since the input signal is only a normal sine, there must be only one point as a phase – in the corresponding frequency of the sine wave. However, for means of comparison, all the other points were kept in order to check the precision of the algorithm. The result is shown in the Figure 21 below.

Figure 21 – Comparison between the phase calculation algorithm implemented in the microntroller and in python. Left: Microcontroller. Right: Python.



Source: Author

Checking the point corresponding the sine wave frequency – the chosen one

was 61, since it has a magnitude 4 times greater than the 62 from the previous analysis – it was around 2,42 rad using both methods, which corresponds to almost 135 degrees. This results in a sine wave slightly shifted, which means that something is adding a small phase to the system.

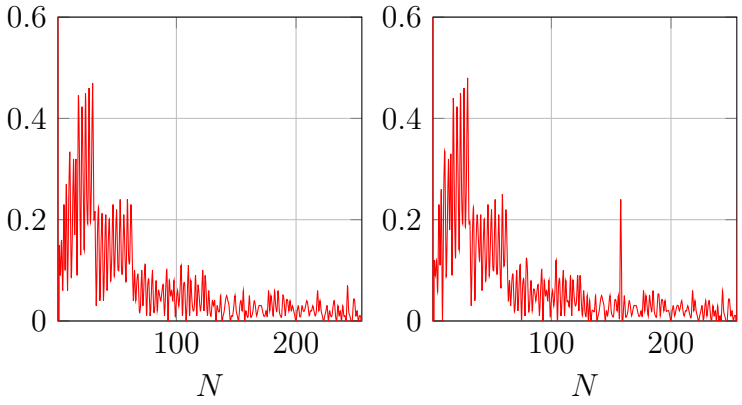
5.2.5 Wavelet

The results regarding the precision of the wavelet families are presented below. For the Daubechies and B-Spline families, only one type of each is shown.

5.2.5.1 Haar

The Haar wavelet famly only contains one wavelet regarding its coefficients, which are only two. The precision results of it are shown in Figure 22 below.

Figure 22 – Comparison between the Haar wavelet implemented in the microntroller and the one in python for the same input. Left: Microcontroller. Right: Python.



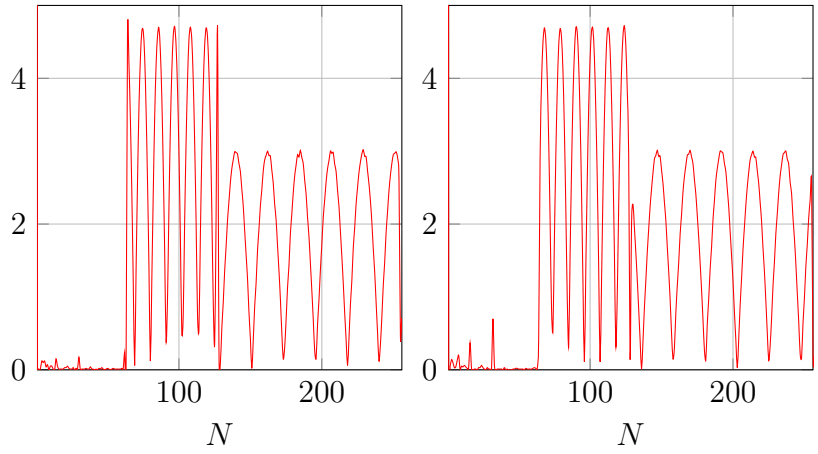
Source: Author

The resulting waves are very similar, but the peak between the samples 150 and 200 can be critical, since the wavelet algorithm – mainly the Haar – is capable to evaluate rapid changes in the frequency.

5.2.5.2 Daubechies

The Daubechies with 4 coefficients was chosen to perform the accuracy test, as it is the second fastest, since a Daubechies with 2 coefficients would be equal in output to a Haar Wavelet. Its precision results are depicted in Figure 23 below.

Figure 23 – Comparison between the FFT implemented in the microntroller and the one in python for the same input. Left: Microcontroller. Right: Python.



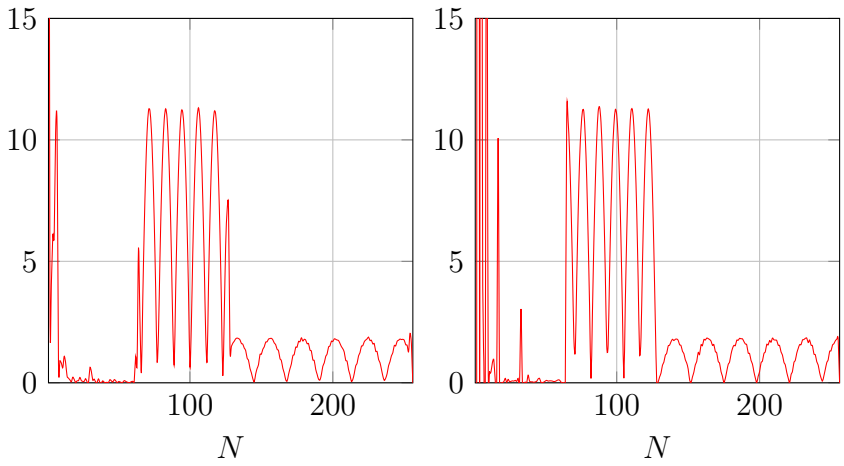
Source: Author

As in the Haar wavelet, is also possible to notice a small distortion is some parts of the transformed wave. Despite of it is looking overall similar, the wavelet transform – as mentioned before – is meant to detect rapid changes in frequency and this difference, although it may seem small, can result in significant wrong answers.

5.2.5.3 B-Spline

The B-Spline 3.1 was chosen due to its time calculation time being similar to the Daubechies 4. The results for this wavelet – which is referred as biorthogonal in python – are shown in the Figure 24 below.

Figure 24 – Comparison between the B-Spline Wavelet implemented in the microntroller and the one in python for the same input. Left: Microcontroller. Right: Python.



Source: Author

Again, the wavelet seems to present some distortions, which seem even bigger in this one. While other wavelets presented differences in only specific parts, the B-Spline errors appeared in almost the whole output.

5.3 REQUIREMENTS

In order to achieve a more precise approach of the goals reached, every requirement previously presented is listed down and analyzed.

5.3.1 Functional Requirements

The functional requirements are, in simple words, the things that the system must do, otherwise it will not work. The fundamental requirements for the software were:

"The microcontroller must have a Floating Point Unit"

This requirement was fully met. Since it is become more common nowadays, all the boards listed in the Chapter 4 contain a chip with an FPU.

"The node must be able to communicate with the base-board via SPI communication"

Although the SPI communication was not implemented here, the chosen microcotroller do have SPI ports enough to communicate with the baseboard, as well as at least one external ADC.

"The software must be able to do all its calculations and processing, as well as send its results, during the time between one ADC sample and another."

As shown in the Chapter 5, the software was able to solve almost all the implemented features inside the time window of an ADC sample. The few exceptions were some wavelets with a considerably number of coefficients, but in those cases there were wavelets in the same family that respected the time constraint.

"The software must be able to compute the magnitudes and the phases of the Fast Fourier Transform"

It can be seen in the Chapter 4 that the algorithm implements both parts of the requirement, and in the Chapter 5 it is shown that both approaches meet the time constraints previously mentioned.

"The software must provide a Peak Detection feature, of at least three peaks"

This requirement, as the way it is purposed, is completely fulfilled. However, if more peaks would be required, it may fail, depending of the amount of peaks desired.

5.3.2 Non-functional requirements

The non-functional requirements describe how the system does the thing that it must do. If they are not met, the system may still work, but probably in a wrong way or with a reduced performance. The non-functional requirements are checked below.

"The data must be sampled with at least 16 bits of precision."

This requirement was not fulfilled. The ADC of the chosen microcontroller has only 12 bits of resolution and no external ADCs were used.

"The data must be sampled in a range between 20 and 40 kHz."

Although it is a bit broad, this requirement is still fulfilled. The first idea was thought for all kinds of sensors, and using a minimum frequency which obeys the Nyquist criteria. However, the software developed here was tested with a sampling rate around 150 kHz, to keep a resolution of 10 times the frequency that the chosen vibration sensor uses.

"The software must have a Wavelet Transform."

When it comes to time constraint, it is possible to affirm that the microcontroller was able of doing a Wavelet Transform, only failing with a high number of coefficients. Regarding the precision, the results were not satisfactory for the application an a deeper research should be conducted in order to why the values presented this difference and what can be done to change it. Since only one tool was used to compare the results, a third one – like Matlab – should be used to improve reliability of the tests.

6 CONCLUSION

This chapter is divided in two sections which together constructs the conclusion of this work. The former summarizes the results obtained and all that have been done until now. The latter brings a glance of what can be done next, to complete or improve what has been done in this project.

6.1 FINAL CONSIDERATIONS

Considering everything that has been analyzed and shown, it is possible to check that the software fulfill a considerable part of its purpose. Comparing to the previously mentioned product – the sensor in Chapter 4 which calculates FFT – the software alone does the same basic functionalities that it does and some other pre-processing part, in a simpler way. In the actual stage, the software is already enough to the applications of Fraunhofer's IPT MSP sensor node. However, after adjustments in the Wavelet algorithm and with some other features added, the software would bring a tool that is not seen in this kind of product, making the prototype more powerful and unique.

6.2 FUTURE WORKS

Despite the chosen microcontroller be considered low power, an analysis of consumption was not made. A deeper study about the microcontrollers themselves need to be done, where the power consumption should also be taken in consideration. In addition to that, one real analysis of speed about the Arduino IDE or the viability of using the other mentioned microcontrollers without it could be beneficial too. Furthermore, a wider variety of models and manufactures should be considered as well, mainly to check about smaller chips, since the chosen one is still too big for the application.

When it comes to features, the most classical ones in the digital process field were included. Additionally, the wavelet transform brings to hardware a tool that is relatively new in the literature, and not so easy to find in this kind of systems. A more specific analysis of the problem could be made to consider other types of features that could me even more efficient in the solution.

Specifically about the peak detection algorithm, in order to make it more generic, there is the possibility of sorting the magnitude or the phase array, and then send only the number of elements requested. An analysis of the trade-off must be done to check the availability of this solution, mainly if the time required pays it off and until how many peaks it would be feasible. The problem of this solution would be how to send the real

positions of the array – because they would be changed with the sorting algorithm – since it is also important. In the version used in this work – the one available at time – CMSIS did not have any kind of sorting algorithms, while the newest versions do have it, which would be nice to give a try. The algorithm implemented here was the simplest as possible, but it does the job. There are more efficient ways to execute a peak detection using statistical models, and this would be the natural evolution of this work in this specific branch.

Besides the aforementioned research in order to fix the wavelet mistakes, there are many things to be done after that. The families used were the ones already present on GSL and they are all discrete. Although having a wider range of wavelets available would give the software a more general purpose, it would be better a deeper analysis of the problem to match which kind of wavelet would suits better for the application. Depending of the result of this analysis, a further research for a C algorithm capable of calculate a continuous wavelet – if it is needed – and if a microcontroller would be able to reproduce it inside the given time constraints is also something that might be worth to search for.

About the phase calculation algorithm, although it solves the problem inside the given requirements, CMSIS also released their own algorithm of ArcTan2. Since it was developed by an engineers team from ARM, it would be a good strategy to migrate to it, not only because it is probably faster, but it should be also better integrated to the library.

The communication part was not added, since it was not ready yet. The SPI communication between the microcontrollers should be the next step of it, together with a self-made protocol that is able to tell which features are wanted and the additional informations of it. After that, the central microcontroller should be able to send those commands, receive the data and send then via UDP protocol. The main commands would be sent and received using the Lightweight Machine to Machine protocol, which connects the microcontroller to the cloud.

As the future idea of Fraunhofer IPT is to migrate the application to the Artificial Intelligence on the Edge field, a research can be done to analyze the viability of it. Since the worries about data protection is growing everyday, the solution could really fit in the Federated Learning field for example, with a classification algorithm running locally, while it sends the results to a local-deployed Artificial Intelligence.

REFERENCES

AL-FALAHY, N.; ALANI, O. Y. Technologies for 5g networks: Challenges and opportunities. **IT Professional**, v. 19, n. 1, p. 12–20, 2017.

ANALOG DEVICES. **ADcmXL3021:** Wide Bandwidth, Low Noise, Triaxial Vibration Sensor. 2019. Disponível em: https://www.analog.com/en/products/adcmxl3021.html#product-samplebuy.

APPLE. **RE: Fast atan2**. 2005. Disponível em: https://web.archive.org/web/20140115122512/http://lists.apple.com/archives/perfoptimization-dev/2005/Jan/msg00051.html.

ARDUINO. **MS Windows NT Kernel Description**. 2018. Disponível em: https://store.arduino.cc/products/arduino-mkr-vidor-4000.

ARDUINO. **Portenta H7**: Meet Portenta H7, designed for high performance. 2020. Disponível em: https://www.arduino.cc/pro/hardware/product/portenta-h7.

ARM. **CMSIS-DSP** CMSIS DSP Software Library. 2019. Disponível em: https://arm-software.github.io/CMSIS_5/DSP/html/index.html.

BITSCOPE. **BitScope Micro | BS05**. 2022. Disponível em: https://www.bitscope.com/product/BS05/.

DAUBECHIES, I. **Ten Lectures on Wavelets**. 2. ed. Philadelphia: Springer-Verlag, 1992.

FRAUNHOFER IPT. **Fraunhofer IPT and Ericsson launch 5G-Industry Campus Europe, Europe's largest Industrial 5G Research Network**. 2019. Disponível em: https://www.ipt.fraunhofer.de/en/Press/Pressreleases/20191213-fraunhofer-ipt-and-ericsson-launch-5g-industry-campus-europe_europes-largest-industrial-5g-research-network.html.

GNU. **GNU Scientific Library**: Release 2.7. 2021. Disponível em: https://www.gnu.org/software/gsl.

HARRIS, C. R. et al. Array programming with NumPy. **Nature**, Springer Science and Business Media LLC, v. 585, n. 7825, p. 357–362, set. 2020. Disponível em: https://doi.org/10.1038/s41586-020-2649-2.

IOWEGIAN INTERNATIONAL CORPORATION. **DSP Trick:** Fixed-Point Atan2 With Self Normalization. 1994. Disponível em: http://dspguru.com/dsp/tricks/fixed-point-atan2-with-self-normalization/.

KEMET. **VS-JV10A:** KEMET, VS-J, Vibration, Metal Sealed, Piezoelectric, Propietary Piezo Ceramic Material, Built-in Amplifier, Detachable Cable. 2019. Disponível em: https://www.kemet.com/en/us/sensors/product/VS-JV10A.html.

LEE, G. R. et al. Pywavelets: A python package for wavelet analysis. **Journal of Open Source Software**, The Open Journal, v. 4, n. 36, p. 1237, 2019. Disponível em: https://doi.org/10.21105/joss.01237.

LI, X.; DONG, S.; YUAN, Z. Discrete wavelet transform for tool breakage monitoring. In: **International Journal of Machine Tools and Manufacture**. [S.l.: s.n.], 1999. p. 1935–1944.

OPPENHEIM, A. V.; SCHAFER, R. W. **Discrete-Time Signal Processing**. 2. ed. New Jersey: Prentice Hall, 2010.

OPPENHEIM, A. V.; WILLSKY, A. S. **Signals & Systems**. 2. ed. New Jersey: Prentice Hall, 1996.

PJRC. **Teensy® 4.1 Development Board**. 2020. Disponível em: https://www.pjrc.com/store/teensy41.html.

SCHMITT, S. S. et al. Meeting the requirements of industrial production with a versatile multi-sensor platform based on 5g communication. In: **2020 IEEE 31st Annual International Symposium on Personal, Indoor and Mobile Radio Communications**. [S.l.: s.n.], 2020. p. 1–5.

SHAFIQUE, K. et al. Internet of things (iot) for next-generation smart systems: A review of current challenges, future trends and prospects for emerging 5g-iot scenarios. **IEEE Access**, v. 8, p. 23022–23040, 2020.

STEWART, D. B. Measuring execution time and real-time performance. In: **In: Proceedings of the Embedded Systems Conference (ESC SF**. [S.l.: s.n.], 2006. p. 1–15.

STMICROELECTRONICS. **NUCLEO-L4R5ZI**: STM32 Nucleo-144 development board with STM32L4R5ZI MCU, supports Arduino, ST Zio and morpho connectivity. 2019. Disponível em: https://www.st.com/en/evaluation-tools/nucleo-l4r5zi.html.

TANEMBAUM, A. S.; BOS, H. **Modern Operating Systems**. 4. ed. Amsterdam: Pearson, 2014.

APPENDIX A - MAIN CODE

```
1 #include "main.h"
2
3 #include <stdio.h>
4 #include <string.h>
5
6 #include "arm_const_structs.h"
7 #include "arm_math.h"
8 #include "features.h"
9 #include "gsl_wavelet.h"
10
11 #define N_SAMPLES 1024
12 #define N_PEAKS 3
13
14 typedef enum feature {FFT, WAVELET} feature_t;
15 typedef enum feat_spec{MAGNITUDE, PHASE} feat_spec_t;
16 typedef enum wavelet_type{DAUBECHIES,HAAR,BSPLINE}wavelet_type_t;
17
18 ADC_HandleTypeDef hadc1;
19 DMA_HandleTypeDef hdma_adc1;
20
21 volatile uint8_t cpt_flag = 0;
22
23 arm_rfft_fast_instance_f32 fft_inst;
24
25 uint16_t adc_buffer[N_SAMPLES];
26 float32_t fft_input [N_SAMPLES];
27 float32_t fft_output[N_SAMPLES];
28
29 float32_t fft_feat[N_SAMPLES];
30 uint16_t indexes[N_SAMPLES];
31 float32_t fft_peaks[N_SAMPLES];
32
33 gsl_wavelet w;
34 gsl_wavelet_workspace work;
35 float32_t scratch[N_SAMPLES];
36
37 feature_t feat = WAVELET;
38 feat_spec_t spec = PHASE;
39 uint8_t pk_dtct = 0;
40
41 wavelet_type_t wav = DAUBECHIES;
42 uint8_t wavelet_centered = 0;
```

```
43 uint8_t n_coeff = 4;
44
45 int main(void)
46 {
47     HAL_Init();
48
49     SystemClock_Config();
50
51     arm_rfft_fast_instance_f32 fft_inst;
52
53     arm_rfft_fast_init_f32(&fft_inst, N_SAMPLES);
54
55     gsl_wavelet_workspace_init(&work, scratch, N_SAMPLES);
56
57     HAL_ADC_Start_DMA(&hadc1, (uint32_t*)adc_buffer, N_SAMPLES);
58
59     while(1)
60     {
61         if(cpt_flag)
62         {
63             if(feat == FFT)
64             {
65                 arm_rfft_fast_f32(&fft_inst,fft_input,fft_output,0);
66
67                 fft_output[0] = 0;
68                 fft_output[1] = 0;
69
70                 if(spec == MAGNITUDE)
71                 {
72                     arm_cmplx_mag_f32(fft_output,fft_feat, N_SAMPLES);
73                 }
74                 else if(spec == PHASE)
75                 {
76                     phase(fft_output,fft_feat, N_SAMPLES);
77                  }
78                 if(pk_dtct)
79                 {
80                     peak_detection(fft_output,fft_feat,indexes,0,
                           N_SAMPLES/2,N_PEAKS,N_SAMPLES/2);
81                 }
82             }
83             else if(feat == WAVELET)
84             {
85                 switch(wav)
86                 {
87                     case DAUBECHIES:
88                         if(!wavelet_centered) gsl_wavelet_init(&w,
```

```
                                gsl_wavelet_daubechies, n_coeff);
89                          else gsl_wavelet_init(&w,
                                gsl_wavelet_daubechies_centered, n_coeff);
90                          break;
91                      case HAAR:
92                          if(!wavelet_centered) gsl_wavelet_init(&w,
                                gsl_wavelet_haar, n_coeff);
93                          else gsl_wavelet_init(&w,
                                gsl_wavelet_haar_centered, n_coeff);
94                          break;
95                      case BSPLINE:
96                          if(!wavelet_centered) gsl_wavelet_init(&w,
                                gsl_wavelet_bspline, n_coeff);
97                          else gsl_wavelet_init(&w,
                                gsl_wavelet_bspline_centered, n_coeff);
98                          break;
99                      default:
100                         break;
101                 }
102                 gsl_wavelet_transform_forward (&w, fft_input, 1,
                        N_SAMPLES, &work);
103                 arm_abs_f32(fft_input, fft_feat, N_SAMPLES);
104             }
105             cpt_flag = 0;
106         }
107     }
108 }
```

ATTACHMENT A - APPLE'S MAIL ATAN2 ALGORITHM

```
#define PI_FLOAT     3.14159265f
#define PIBY2_FLOAT  1.5707963f
// |error| < 0.005
float fast_atan2f( float y, float x )
{
        if ( x == 0.0f )
        {
                if ( y > 0.0f ) return PIBY2_FLOAT;
                if ( y == 0.0f ) return 0.0f;
                return -PIBY2_FLOAT;
        }
        float atan;
        float z = y/x;
        if ( fabsf( z ) < 1.0f )
        {
                atan = z/(1.0f + 0.28f*z*z);
                if ( x < 0.0f )
                {
                        if ( y < 0.0f ) return atan - PI_FLOAT;
                        return atan + PI_FLOAT;
                }
        }
        else
        {
                atan = PIBY2_FLOAT - z/(z*z + 0.28f);
                if ( y < 0.0f ) return atan - PI_FLOAT;
        }
        return atan;
}
```