Politecnico di Torino

Master thesis

Front-end for capacitive sensors

Master degree in Embedded System

Student:
Giorgia Subbicini (259042)

Supervisor and co-supervisor:
Prof. Luciano Lavagno,
Prof. Mihai Teodor Lazarescu

July 2020

To my family, to my angels, to my brother Matteo and above all to my mum Donatella,
she gives me the strength to go on even when the clouds came floating into my life.

Abstract

This research focuses on a long-range capacitive sensors front-end able to reject environmental noise.

Capacitive sensors can be used for indoor person localization, therefore they can be adopted in many location-based applications that improve life quality and safety. Room occupancy information can help reducing energy consumption by controlling the ambient temperature, lighting and water. Assisted living systems can play an increasingly important role, since the average human lifespan is of about eighty years. The sensors can be employed also in medical field, indeed, monitoring human activity for extended periods can detect behavioral changes, which can help recognizing the early onset of diseases like Parkinson disease. Moreover, another field of interest is police and security, since they can detect unauthorized intrusions.

Technologies for localization are classified as active systems and passive systems. The former require interaction to localize people and can raise privacy concerns, the latter are less intrusive and they do not need specific interaction. Systems based on video or cameras belong to the first cathegory. They often require high computational, networking and energy resources, a direct line of sight, and adequate lightning, which increase the installation complexity and system cost. They also raise significant privacy concerns.

Other active solutions are based on Ultra-Wide Band (UWB) radios or on the measurement of received signal strength variations on narrow channels. These systems require an active tag or the installation of many mains-powered sensors, affecting the system reliability and usability.

Ultrasonic systems have also been used, however, they need long-term exposure to ultrasonic noise that can cause harmful health effects.

Other systems attach tags to the objects that are routinely used by the person to monitor their movements, nevertheless, if the person does not interact with the monitored objects the system will fail.

A possible tagless solution can be a systems based on passive infrared sensors (PIR). It requires a large number of sensors for effective localization, which increase the installation cost and reduce the user acceptance and may induce stress, because they visually remind them that they are being monitored. Moreover, PIR sensors can give false readings if they are exposed to common infrared (IR) sources.

Load-mode single-plate capacitive sensors belong to passive systems and they provide several advantages, such as simplified installation, reduced overall cost, and no significant privacy concerns.

Capacitive sensors, operating in load mode, use just one-plate transducer and the human body as a constant-potential plate. However, load mode capacitive sensors are generally used up to distances comparable to their plate dimensions, which are too short for indoor localization.

Long-range capacitive sensors need to sense very small variations of the capacitance of the sensor plate. Their sensitivity and accuracy are often limited by the level of noise induced by environmental sources. To mitigate these effects, the sensor plate can be

guarded by auxiliary fields to reduce the unwanted couplings of the sensor plate with the surrounding objects, and by post-processing sensor data to improve the reliability of long-range measurements.

An existing technique to reduce environmental noise is based on measuring changes in the capacitance of sensor by gauging the free running frequency of an astable multivibrator, which repeatedly charges and discharges the capacitor of the sensor. The technique uses also low-pass filters to reduce high frequency noise and high-pass filters for reducing low-frequency drift. However, the environmental noise can change the oscillation period, hence introduce errors in the measured capacitance value.

Here, I study a technique based on differential single plate measurements. This technique can reject low frequency noise which is a common mode signal for the differential acquisition and processing of sensor data. It is based on having a constant current, that charges-discharges the plate, driven by a square wave. Thus the plate voltage is a triangular wave with slopes proportional to the capacitance of sensor. An Analog to Digital Converter, synchronized with the driving square wave, is used to sample the plate voltage. To improve the accuracy of the measurements and the rejection of drift noise I use the oversampling and decimation technique that allows to increase the resolution by four bits (from 12 bit to 16 bit) of the analog to digital converter and reduce the quantization error.

After setting up the system for differential measurements, I implement an analog low-pass antialias filter, with cut-off frequency between 15 kHz and 20 kHz, which is about ten times higher than the measurement frequency to minimize the degradation of front-end waveforms.

The filter uses the operational amplifier available in the microcontroller and external capacitors and resistors. Due to the non-linearities of the active components, the measurements after the filtering show some distortions, thus I evaluate the errors.

Another element in the front end is a digital filter to reduce the noise outside the frequency bands of interest. A digital filter is a system that performs mathematical operations on a sampled, discrete-time signal to reduce or enhance certain aspects of that signal. They may be more expensive than an equivalent analog filter but they can reach very high orders. Many digital filters are based on the Fast Fourier Transform. The mask of the filter consists in a cut off frequency of 1 Hz or 3 Hz since the frequency band of interest is between 0 Hz-3 Hz, and -80 dB magnitude reduction in stopband.

I implement the system by using the Ultra low power Nucleo-L412KB microcontroller produced by STMicroelectronic. Of its peripherals, I use the Analog to Digital Converter (ADC), the Direct Memory Access (DMA), a Timer (TIM2), a Universal Synchronous-Asynchronous Receiver/Transmitter communication channel (USART) and the Operational Amplifier (OPAMP). To comply with the requirements of the differential single plate measurement technique, microcontroller runs at 80 MHz, and the timer (TIM2) is triggered by the falling edge of the external square wave which drives the triangular wave to be sampled. For 2 MHz ADC sampling the timer counts up to 40 and once it achieves max count it triggers a sample of the Analog to Digital Converter. The analog to digital converter works with the Direct Memory Access element to reduce the CPU workload. Once it samples 1250 values over a period of the triangular wave, it wakes up the CPU to calculate the capacitance from the slopes of the triangular waveform, then the measurement is sent through the USART channel to the host computer.

I calculate the capacitance values from the ADC samples using averaging technique and I compare them with the capacitance values calculated using oversampling and decimation. Averaging is a simple and effective way to smooth the input waveform and reduce sensor noise, whereas, oversampling and decimation combines multiple samples to compute a result with greater accuracy than a single real sample can provide. It also helps to improve the signal-to-noise ratio of the input signal, which depends on the number of bits. Thus, the standard deviation computed over the capacitance values calculated with oversampling is smaller than standard deviation of the calculations using averaging. Standard deviation is further reduced using the low-pass antialiasing filter before the ADC, as it further reduces high frequency noise of the signal. However, as explained before, the filter introduces a delay in the time domain that can decrease the accuracy of the measurements. To fix the problem, I delay the analog to digital converter sampling start by 10 us for 15 kHz filter and 7.5 us for 20 kHz.

To analize the noise and filtering I compute the Fast Fourier transform (FFT) of a 5 minute trace with different digital filters: Finite Impulse Response, Chebyshev, Elliptic and without filter. They are designed with the mask reported above and with the minimum order to match the stopband characteristic. From the results I find that the Elliptic filter has the steepest roll-off rate, and it requires a lower order, FIR has the worst roll-off rate and it is computationally expensive while Chebyshev has a good performance even if it requires a bigger order than Elliptic filter.

To conclude, I can state that differential single plate measurements technique with oversampling and decimation and antialias analog filter can reduce the high-frequency noise and reject drifts and jitters, whereas the Elliptic digital filter is the most efficient to reduce the noise outside the band of interest.

Contents

List of Figures

Chapter 1

Introduction

Indoor person localization is useful for several location-based services that can improve life quality and safety, such as tracking the movements of elderly people, or people with special needs in order to avoid dangerous accidents or to collect medical information, or even controlling light and heat consumption automatically based on person presence and physical location.

Localization can be implemented in an active way or through passive systems which do not require interaction to localize people and are less intrusive. Either way, they should not raise privacy concerns.

Some active methods can be based on video or pictures, which often require high computational, networking and energy resources, a direct line of sight, and adequate lightning, which increase the installation complexity and system cost. They also raise significant privacy concerns.

Other solutions of the same category are based on Ultra-Wide Band (UWB) radios or on the measurement of received signal strength variations on narrow channels. These systems require a person to wear an active tag or the installation of many mains-powered sensors, which can have a negative impact over reliability and usability.

Ultrasonic systems have also been used, however, they need long-term exposure to ultrasonic noise that can cause harmful health effects on persons or pets.

Other systems attach tags to the objects that are regurarly used by the person to monitor their movements. Nevertheless, if the person does not interact with those objects the system will fail.

An effective tagless solution can be a system based on passive infrared sensors (PIR). However, it requires a large number of sensors for effective localization, which increase the installation cost, can be intrusive and reduce the user acceptance, because they visually remind them that they are being monitored. Moreover, PIR sensors can give false readings if they are exposed to common infrared (IR) sources, such as sunlight, good heat conductors, IR radiation reflectors, incandescence light bulbs [3].

A passive system can be implemented using "capacitive sensors", that allow the detection and tracking of conductive and non-conductive objects. It can be used stand-alone or in combination with, e.g., the largely complementary PIR sensors.

As reported in [1], capacitive sensors use capacitive transducers in load operation mode that use one-plate transducer and the human body as a constant-potential plate. However, the sensing range of load mode capacitive sensors is typically comparable to their plate dimensions, which is generally too short for indoor localization applications.

Long-range capacitive sensors need to sense very small variations of the capacitance of the sensing plate (below 0.1%). Their sensitivity is often limited by the level of noise induced by environmental sources. Hence, techniques that reduce measurement noise effectively increase the sensing range and stability of capacitive sensors.

Electrical capacitance is often determined by measuring the time in which the plate voltage varies between pre-defined thresholds while the amount of electrical charge on plate is varied using controlled electrical currents. Notably, plate voltage is susceptible to environmental noise from various electrical and electrostatic noise sources, noise which often translates in drift and jitter of capacitance measurements that limit the sensor range and sensing stability. Indeed, distribution or redistribution of electric charges or fields in the sensor environment due to external causes (which do not change the capacitance of the sensor plate) often influence its measurement through the voltages or currents which are used for capacitance calculation. Some existing techniques for the measurement of the electrical capacitance typically employ guard fields (for relatively short-range measurements) or differential measurements using multiple sensor plates.

Guard electric fields are generated by special guard electrodes (plates) of the sensors around the electric field of the measurement plate, such way to not interfere with the latter while protecting it from environmental interferences. Besides the added constructive and measurement complexity, this technique is effective only for relatively short measurement distances, comparable or less than the diameter (or an equivalent length) of the sensor, because the integrity of the auxiliary field degrades rapidly with the distance.

Low noise differential capacitance measurement using multiple plates relies on the fact that effects of environmental noise are nearly identical on matched plates packed closely together in the sensor layout. However, the differential nature of the reading requires to drive the plates in counterphase, which causes their electric fields to interfere and reduce the sensitivity (thus the measurement range) of the sensor.

In order to extend the sensing range, capacitance measurements are enhanced by baseband digital filters to attenuate the noise captured by sensor, and localization algorithms to infer the position of the person using the data from several load-mode capacitive sensors within one or several rooms.

This research focuses on the implementation of a new front-end for capacitive sensors with long range capabilities. Differential single plate measurements technique is used to improve the sensitivity. It is based on forcing a current through the sensor plate that is constant and with sign that changes periodically according to the levels of a square wave. This changes the voltage plate linearly in time with a slope proportional to the capacitance of the sensor plate. The capacitance value can be retrieved from the measurements of slopes which are sampled by an analog to digital converter (ADC). Accuracy and noise level are improved using oversampling and decimation. The front-end is also composed of an analog low-pass filter placed before the ADC, which, together with the oversampling and decimation can reduce high-frequency noise. Then, after the calculation and the collection of the capacitance values a digital filter reduces the noise outside the bands of interest. These improvements can lead to a system that is able to localize people with a good precision even if they are far from the device, without requiring the persons to wear tags and respecting their privacy.

Chapter 2

Front-ends for capacitive sensors

As explained in the introduction section, plate voltages are subjected to environmental noise that is then translated into drift and jitter of capacitance measurements. The front end should assess the stability by reducing the sensitivity of measurements of plate capacitance to environmental electric and electrostatic noise.

2.1 Existing techniques

In [1] the authors present a localization system composed of four capacitive sensors working in load mode. Changes in the capacitance of sensors are measured by gauging the free running frequency of an astable multivibrator, which repeatedly charges and discharges the capacitor of the sensors.

The operation of the system consists of:

- indirect measurement of the capacitance of the transducer by measuring the frequency of a relaxation oscillator whose electrical and timing characteristics depend on transducer capacitance;
- use of baseband digital filters to attenuate the noise captured by the sensor;
- use of localization algorithms to infer the position of the person using the data from several load-mode capacitive sensors within one or several rooms.

They use sensors with different copper clad plate size, as the external capacitor, and 555 integrated circuit in astable multivibrator configuration, whose oscillation frequency is:

$$\frac{1}{0.7(R_1 + 2R_2)C} \tag{2.1}$$

They use R_1= 200 kOhm and R_2= 560 kOhm in order to provide a good trade-off between the sensor size and its sensitivity.

The frequency is measured by counting the full periods of the oscillator in an interval of one second, using an Arduino Uno board. Then the measurements are sent to a central node through ZigBee module for post processing. The data from the sensors can be affected by two types of noise:

- high-frequency noise from appliances and light switches.
- a low-frequency drift, where the DC component varies very slowly by a significant amount (larger than the useful signal component), due to slow leak of static charges, temperature and humidity changes, and so on.

Authors use low-pass filters to reduce the first noise type, whereas they use high-pass filters for the second noise type .

The filtering block consists of a moving median filter with a window of 5 input data points as a digital low-pass filter to attenuate the high frequency noise and to remove a slowly varying DC bias from the signal. For high-pass filtering they subtract the mean of the first 10 samples (where the person was far from all sensors) from the following 150 samples in each data set (for a total of 160 samples). After the filtering phase, the sensor data pass through the localization block, which determines the position of the person within the room. The experiments done are based on tracking the position of a single person in a 3 m x 3 m room. They achieve very good results with 16 cm x 16 cm plates and Naive Bayes (NB) algorithm.

In [3] a similar system is proposed with some differences in filtering. The system is composed of 4 sensors with 8 cm x 8 cm plate each, attached to a 555 integrated circuit in astable multivibrator configuration. The frequency is measured by an Arduino Board and measurement is sent to a base station through an XBee 802.15.4 modem. To compensate for changes of frequency, the authors calculate the standard deviation of all the samples in a given set, then they calculate the average only for the samples within the bounds of the standard deviation, and finally they subtract the average value from all the samples of each set. Then this set passes through the Weka ML algorithms to infer the person location. The processing uses a median filter as in the system above, with a window of 128 seconds. This system is different from the first one for two main reasons:

- data are not denoised with low-pass filters;
- better implementation of the machine learning algorithms are used (from Weka collection).

They conclude that the Random Forest algorithm performs best overall, while AdaBoostMI requires much less time for inference at the cost of a small accuracy loss.

However, better sensor nodes with high resistance to environmental noise must be used, in order to improve the quality of the results. A new technique based on differential single plate measurements can be exploited to reduce environmental noise induced in capacitive sensor measurement (both low frequency drift and high-pitch jitter). Noise reduction effectively increases sensor sensitivity and sensing range, which are very important for applications that need long-range measurement of distance to slow-moving or stationary bodies. The main advantages and the most important features of "differential single place measurement" technique are:

- It preserves the sensitivity of the sensor while reducing most important sources of noise.

- Can be used on existing capacitive sensors without any physical modification of sensor plate(s)
- Can be used on single-plate sensors.
- It does not rely on auxiliary electric fields for shielding, which may loose effectiveness with the distance.
- Drift reduction comparable to dual-plate differential measurement techniques.
- Reduction of measurement jitter because it does not use reference thresholds.

Next section is dedicated to the presentation and theoretical explanation of the technique, all the informations are retrieved from [6].

2.2 Differential measurements technique

The differential single plate measurements technique consists of a processing method for capacitive sensor reading that significantly reduces external (environmental) noise captured by the sensor. It rejects low frequency noise which is a common mode signal for the differential acquisition and processing of sensor data. High-frequency noise is reduced by two types of low-pass filters: analog (antialiasing) followed by digital averaging. Noise outside the bands of interest for specific applications can be subsequently further filtered using analog or digital filters.

Figure 2.1 shows the block schematic of the processing method. An analog front-end periodically charges-discharges the sensor plate using constant currents, driven by a square wave measurement signal generated by a square wave generator.

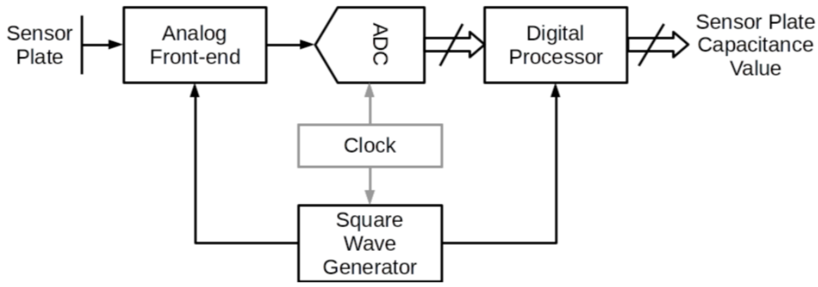

Figure 2.1. Block schematic of a possible implementation of the processing chain to reject environmental low frequency noise (drift) and reduce high frequency noise in single-plate capacitive sensors readings.

The current Ic through sensor plate is proportional to the voltage level of the square wave Vm. In this way the current is constant and its sign changes periodically according to the levels of Vm. Hence, the plate voltage Vc changes in time with a slope that is proportional to the capacitance of sensor C, and the value and sign of Ic. The waveforms of the main signals are shown in Figure 2.2.

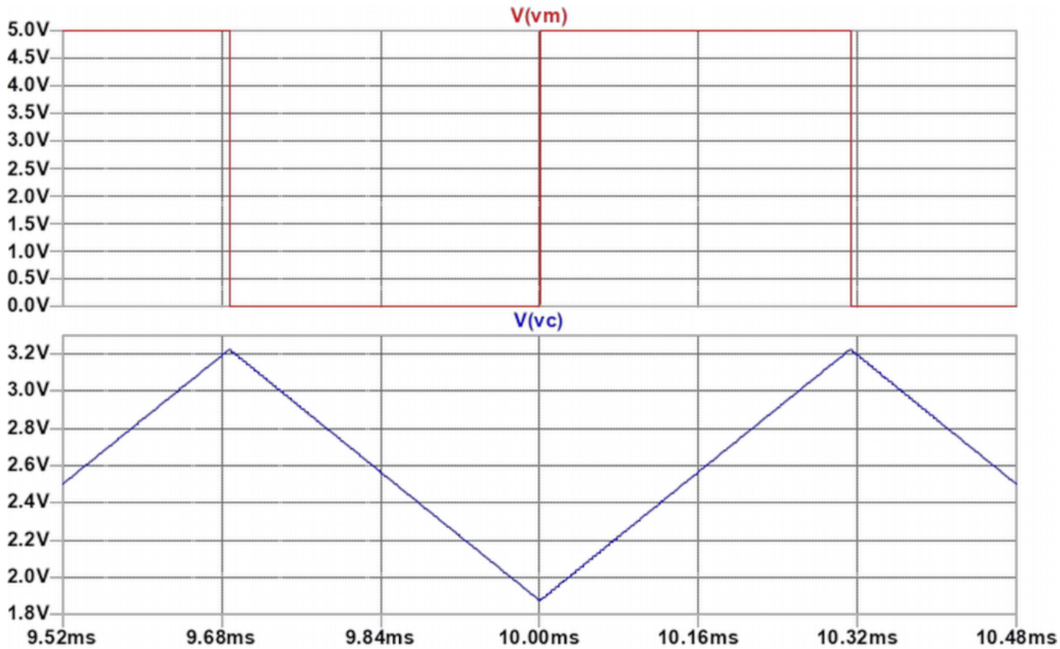

Figure 2.2. Square wave and relative plate voltage behavior

The capacitance of the plate can be calculated from the slopes of the triangular wave. As shown in Figure 2.3, an analog to digital converter ADC is used to sample Vc synchronized to the transitions of the square wave signal, Vm. The ADC sampling frequency and the measurement frequency, are selected such way to:

- ADC samples the transducer voltage, Vc, an even integer number of times, 2NR, during each period, Tm, of Vm, $Tm = \frac{1}{fm}$.

- The number of ADC samples acquired on each ramp of Vc, NR, can be split in two segments S to satisfy the following conditions:

  - The total length of the two segments should cover a large part of the ramp, but not all of it (e.g., 85% of the ramp): $2NS < NR$.

  - The parts toward the extremes of the ramp that are not covered by the segments are called "guard intervals" (against possible nonlinearities at the ends of ramps) and each is covered by NG ADC samples.

  - The number of ADC samples on each segment, NS, is an integer power of four: $NS = 4^n$.

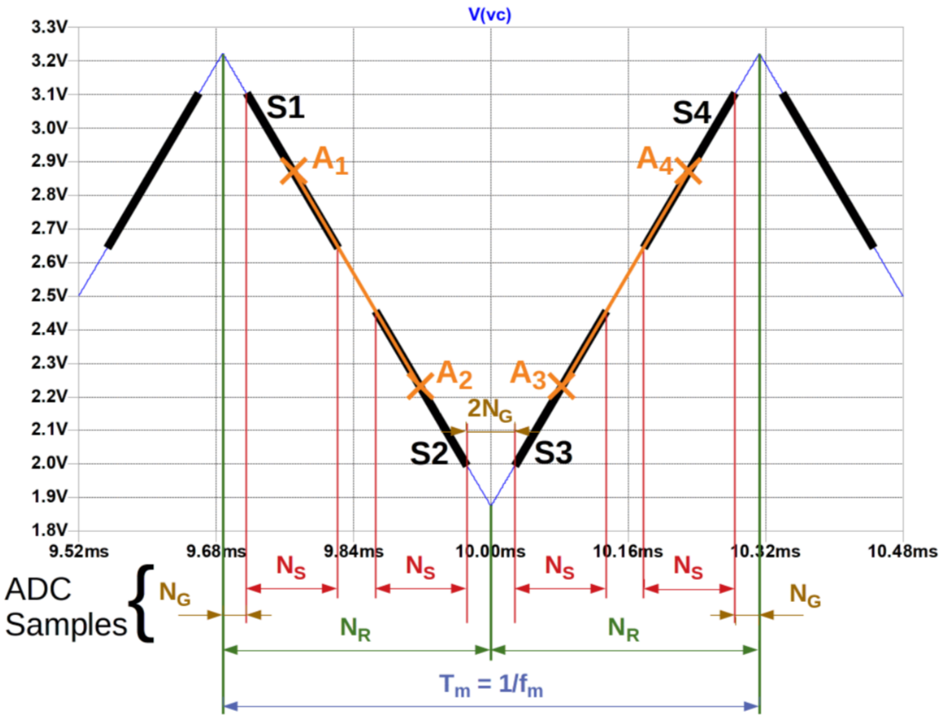

Figure 2.3. Example of acquisition of capacitive transducer readings

The capacitance of the sensor plate is determined from the value of the slopes of the two ramps of each period,Tm, of the square wave signal, Vm, as follows. During a half period of the square wave signal, the charge ΔQ injected (or extracted) from the sensor plate by the constant current Ic is:

$$\Delta Q = Ic\frac{Tm}{2} \tag{2.2}$$

Plate capacitance (C) is defined as the change of plate charge ΔQ divided by the change of plate voltage ΔVc:

$$C = \frac{\Delta Q}{\Delta Vc} \tag{2.3}$$

By substituting 2.1 into 2.2, the change of plate voltage ΔVc becomes:

$$\Delta Vc = \frac{Ic}{C}\frac{Tm}{2} \tag{2.4}$$

Variations of nearby electric potential or changes in the intensity of electric field incident on the plate, may influence the migration of plate charges and so may cause a quasi-constant drift current Ie. The sign of externallyinduced Ie does not change, hence, the slants of two adjacent ramps of Vc are:

$$S_L = \frac{1}{C}(+Ic + Ie) \tag{2.5}$$

$$S_R = \frac{1}{C}(-Ic + Ie) \tag{2.6}$$

The average slant S_A is calculated as the average of the modules of the right and left slants:

$$S_A = \frac{|S_L| + |S_R|}{2} \tag{2.7}$$

Thanks to this calculus, the common mode quasi-constant drift current Ie is canceled while the full sensitivity is preserved:

$$S_A = \frac{1}{C}\frac{(Ic + Ie) - (-Ic + Ie)}{2} \tag{2.8}$$

$$S_A = \frac{Ic}{C} \tag{2.9}$$

From this last formula, it's evident that the average of the modulus of slants of adjacent ramps:

- is proportional to the inverse of plate capacitance;
- does not depend on quasi-constant variations of plate charge due to sensor environment.

Drift noise rejection depends on the match of the Ie noise component of Ic on the two Vc ramps considered. Hence, ramp slant measurement accuracy is important for a good match and cancellation of the Ie component. In order to reduce the influence of ADC discretization errors and high-pitch Vc noise on the measurements, Vc slant is retrieved as difference in voltage divided by difference in time of two points on each ramp (with the notations in Figure 2.3):

$$S_L = \frac{V_{A2} - V_{A1}}{t_{A2} - t_{A1}} \tag{2.10}$$

$$S_R = \frac{V_{A4} - V_{A3}}{t_{A4} - t_{A3}} \tag{2.11}$$

To determine S_L and S_R with high accuracy, the ADC should assess each V_{An}, with $n = 1,2,3,4$, with great detail, that can be improved using oversampling and decimation. As mentioned before, each segment Sn of Vc in Figure 2.3, with n = 1,2,3,4, is sampled NS times, with NS selected to be an integer power of four, NS = 4^n. Vc is oversampled NS times by the ADC. These additional samples of Vc are subsequently processed to reduce both ADC quantization error (increasing its resolution by n bits) and to average Vc noise as follows:

- We add all NS samples of Vc into an accumulator A.
- We divide A by 2^n to average Vc random noise.
- The value left in A represents the calculated value of V_{An} of the corresponding S_n segment, with $n = 1,2,3,4$.

Another important rule to increase the accuracy of ramp slants calculations is to extend as much as possible the time difference between two segments. Two factors need to be considered:

- A segment should include as many ADC samples as possible to increase the effectiveness of reduction of quantization noise and external Vc noise through averaging.
- Segments of a ramp should be sufficiently separated from ramp ends to avoid non-linearities. In Figure 2.3 this separation is shown as NG ADC samples.

Non-linearities can be due to active components in the sensor front-end circuits or due to low-pass anti-alias filters.

The goal of my thesis is implementing this new system. In particular I put the attention on ADC configuration, software for elaborating ADC samples in averaging mode and oversampling and decimation mode, analog filters and digital filters with spectrum analysis.

In the following chapters firstly I will present the implementation together with description of components and instruments used, secondly I will display all the experimental results collected with further explanations over the reasons and in the end some conclusions and possible future developments will be highlighted.

Chapter 3

Instruments and implementation

Figure 3.1 displays the slope modulation measurement circuit. Cplate charge is changed by a current, I, generated by a source controlled with a square wave, v. The e_i represents constant charge migration current due to environmental noise. The charge and discharge time is fixed, since it depends on the square drive signal, regardless of V_c noise. Figure 3.2 is an enlargement of the right part (after buffer point) of the whole system, which represents the processing chain to reject environmental low frequency noise (drift) and reduce high frequency noise in single-plate capacitive sensors readings.

The first block is an analog antialias filter. It is implemented as active low-pass filter by using an operational amplifier (OPAMP) available in the microcontroller and external resistors and capacitors. It is used to reduce the high-frequency noise and the ADC aliasing. The input of the block is a triangular wave generated by a signal generator and the output is already a triangular wave. The second block is an Analog to Digital Converter (ADC). Its samples are used to determine the slopes of the triangular wave with high accuracy by exploiting oversampling and averaging and to calculate the capacitance as explained in Chapter 2. The ADC used belongs to the STM Nucleo-L412KB uC, and its characteristics and settings will be explained in the following sections. The ADC is synchronized with the driver square wave and it samples the triangular wave. The third and last block is a digital filter. Digital filters are implemented through a tool for technical computing (MATLAB) and they are used to filter noise outside the bands of interest. This last processing is done in frequency domain by evaluating the Fast Fourier Transform (FFT) over a long trace of measured values. In the following sections I will present the components and the tools used, as well as how they are configured.

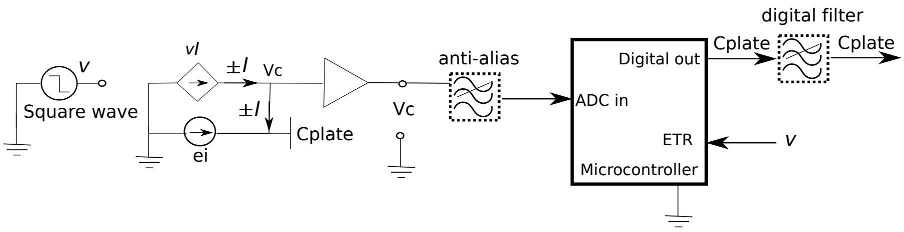

Figure 3.1. Whole system of single plate differential measurements technique implementation

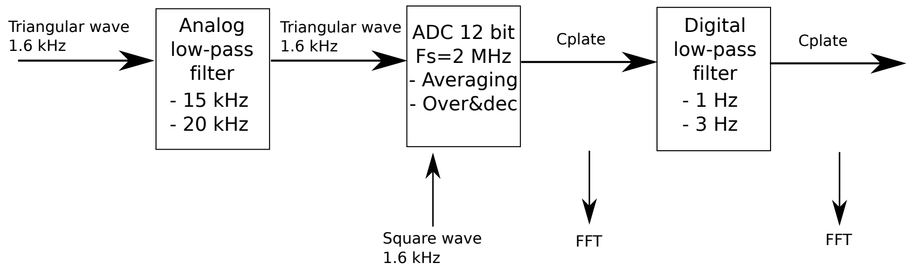

Figure 3.2. Block scheme of the processing chain of single plate differential measurements technique

3.1 Instruments

3.1.1 The microcontroller

The microcontroller used to implement the system is the Ultra-low-power Nucleo-L412KB supplied by STmicroelectronics. As reported by the datasheet [7], the device is based on the high-performance Arm® Cortex®-M4 32-bit RISC core operating at a frequency of up to 80 MHz. The Cortex-M4 core features a Floating point unit (FPU) single precision which supports all Arm® single-precision data-processing instructions and data types. It also implements a full set of digital signal processor (DSP) instructions and a memory protection unit (MPU) which enhances application security. The STM32L412KB embeds high-speed memories (Flash memory up to 128 KB, 40 KB of SRAM), a Quad serial peripheral interface (SPI) flash memories interface (available on all packages) and an extensive range of enhanced I/Os and peripherals connected to two advanced peripheral bus (APB) buses, two advance high-performance bus (AHB) buses and a 32-bit multi-AHB bus matrix. The STM32L412KB embeds also several protection mechanisms for embedded Flash memory and SRAM: readout protection, write protection, proprietary code readout protection and Firewall. The device offers two fast 12-bit analog to digital converters (ADC) (5 Msps), two comparators, one operational amplifier, a low-power real time clock (RTC), one general-purpose 32-bit timer, one 16-bit pulse width modulator (PWM) timer dedicated to motor control, four general-purpose 16-bit timers, and two 16-bit low-power timers. In addition, up to 12 capacitive sensing channels are available. It also features standard and advanced communication interfaces.

The STM32L412KB operates in the -40 °C to +85 °C (+105 °C junction) and -40 °C to +125 °C (+130 °C junction) temperature ranges from a 1.71 to 3.6 V VDD power supply when using internal low-dropout (LDO) regulator and a 1.00 V to 1.32V VDD12 power supply when using external switched-mode power supply (SMPS). A comprehensive set of power-saving modes allows the design of low-power applications. Some independent power supplies are supported: analog independent supply input for ADC, OPAMP and comparator. A VBAT input allows to backup the RTC and backup registers. Dedicated VDD12 power supplies can be used to bypass the internal LDO regulator when connected to an external SMPS. Figure 3.3 displays the schematic of the board while Figure 3.4 shows a table of connectors and Figure 3.5 is about pin assignment.

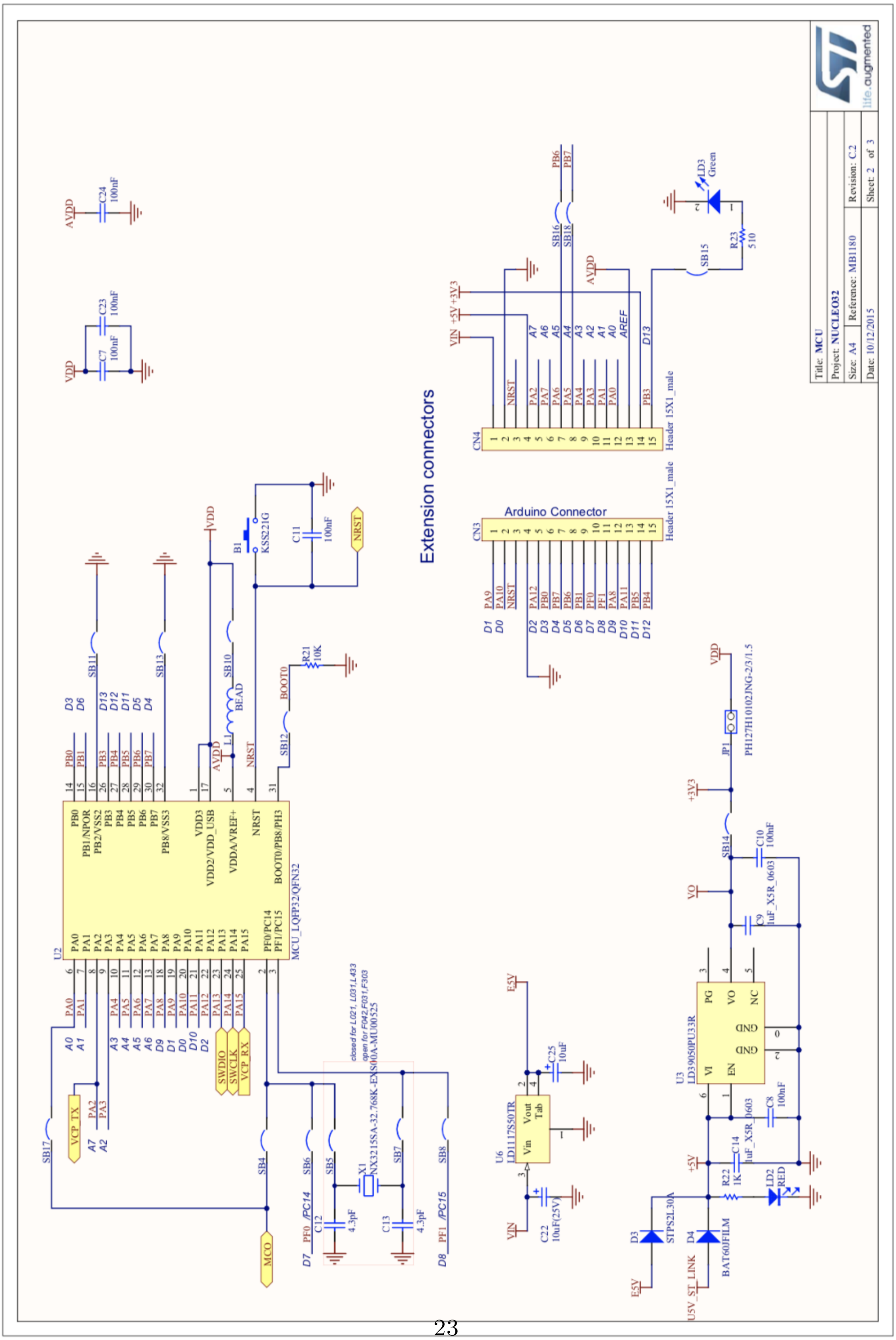

Figure 3.3. Microcontroller board schematic

Connector	Pin number	Pin name	STM32 pin	Function
Left connector				
CN3	1	D1	PA9	USART1_TX
	2	D0	PA10	USART1_RX
	3	RESET	NRST	RESET
	4	GND	-	Ground
	5	D2	PA12	-
	6	D3	PB0	TIM1_CH2N[1]
	7	D4[2]	PB7	-
	8	D5[2]	PB6	TIM16_CH1N[1]
	9	D6	PB1	TIM1_CH3N[1]
	10	D7[3]	PC14	-
	11	D8[3]	PC15	-
	12	D9	PA8	TIM1_CH1
	13	D10	PA11	SPI_CS[4] || TIM1_CH4
	14	D11	PB5	SPI1_MOSI || TIM[5]
	15	D12	PB4	SPI1_MISO
Right connector				
CN4	1	VIN	-	Power input
	2	GND	-	Ground
	3	RESET	NRST	RESET
	4	+5V	-	5 V input/output
	5	A7	PA2	ADC1_IN7[6]
	6	A6	PA7	ADC1_IN12
	7	A5[2]	PA6	ADC1_IN11 || I2C1_SCL
	8	A4[2]	PA5	ADC1_IN10 || I2C1_SDA
	9	A3	PA4	ADC1_IN9
	10	A2	PA3	ADC1_IN8
	11	A1	PA1	ADC1_IN6
	12	A0	PA0	ADC1_IN5
	13	AREF	-	AVDD
	14	+3V3	-	3.3 V input/output
	15	D13	PB3	SPI1_SCK

Figure 3.4. Table of connectors of microcontroller board

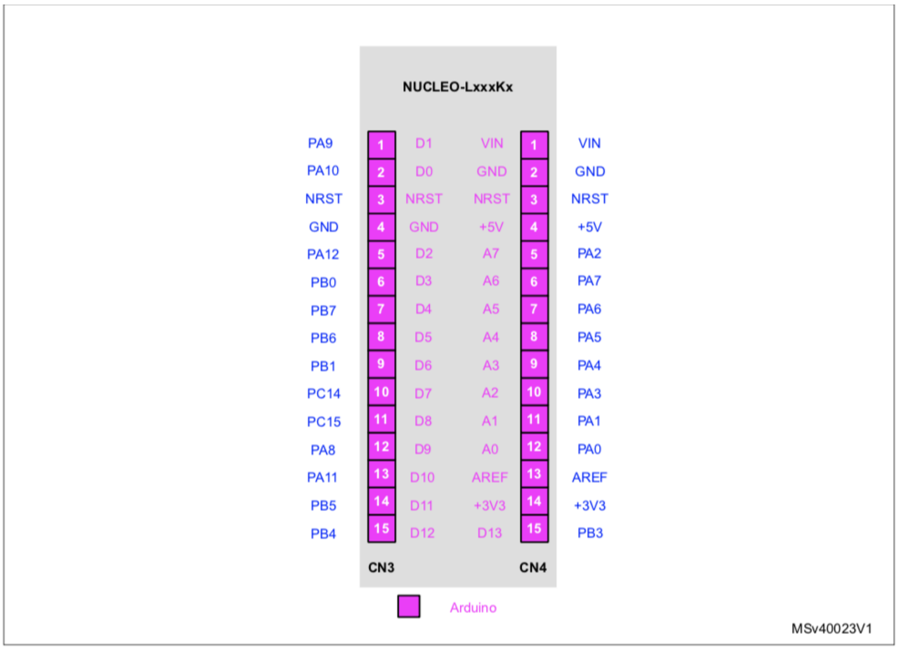

Figure 3.5. Pins of the microcontroller board

I present in more details the performance and main settings of the systems that I used: ADC, DMA, OPAMP, TIM2, and USART.

Analog To Digital Converter

The device embeds 2 successive approximation analog-to-digital converter with the following features:

- 12-bit native resolution, with built-in calibration
- 5.33 Msps maximum conversion rate with full resolution
- Up to 16 external channels, some of them shared between ADC1 and ADC2.
- 3 internal channels: internal reference voltage, temperature sensor, VBAT/3.
- One external reference pin is available on some package, allowing the input voltage range to be independent from the power supply
- Single-ended and differential mode inputs
- Low-power design
- Highly versatile digital interface

Direct Memory Access

The device embeds 2 direct memory access DMAs. DMA is used in order to provide high-speed data transfer between peripherals and memory as well as memory to memory. Data can be quickly moved by DMA without any CPU actions. This keeps CPU resources free for other operations. The two DMA controllers have 14 channels in total, each dedicated to managing memory access requests from one or more peripherals. Each has an arbiter for handling the priority between DMA requests. The DMA supports:

- 14 independently configurable channels (requests)
- Each channel is connected to dedicated hardware DMA requests, software trigger is also supported on each channel. This configuration is done by software.
- Priorities between requests from channels of one DMA are software programmable (4 levels consisting of very high, high, medium, low) or hardware in case of equality (request 1 has priority over request 2, etc.)
- Independent source and destination transfer size (byte, half word, word), emulating packing and unpacking. Source/destination addresses must be aligned on the data size.
- Support for circular buffer management
- 3 event flags (DMA Half Transfer, DMA Transfer complete and DMA Transfer Error)
- Logically ORed together in a single interrupt request for each channel
- Memory-to-memory transfer
- Peripheral-to-memory and memory-to-peripheral, and peripheral-to-peripheral transfers
- Access to Flash, SRAM, APB and AHB peripherals as source and destination
- Programmable number of data to be transferred: up to 65536.

Operational Amplifier

The STM32L412KB embeds one operational amplifier OPAMP with external or internal follower routing and PGA capability. The operational amplifier features are:

- Low input bias current
- Low offset voltage
- Low-power mode
- Rail-to-rail input

Timer

There are up to three synchronizable general-purpose timers embedded. Each general-purpose timer can be used to generate pulse-width modulation PWM outputs, or act as a simple time base. TIM2 It is a full-featured general-purpose timers:

- TIM2 has a 32-bit auto-reload up/downcounter and 32-bit prescaler.
- This timers feature 4 independent channels for input capture/output compare, PWM or one-pulse mode output. They can work with the other general-purpose timers via the Timer Link feature for synchronization or event chaining.
- The counters can be frozen in debug mode.
- All have independent DMA request generation and support quadrature encoder.

Universal Synchronous-Asynchronous Receiver/Transmitter

The STM32L412 devices have three embedded universal synchronous receiver transmitters (USART1, USART2 and USART3).

These interfaces provide asynchronous communication, IrDA SIR ENDEC support, multiprocessor communication mode, single-wire half-duplex communication mode and have LIN Master/Slave capability. They are able to communicate with a speed up to 10 Mbit/s. USART1, USART2 and USART3 also provide Smart Card mode (ISO 7816 compliant) and SPI-like communication capability. All USART have a clock domain independent from the CPU clock, allowing the USARTx (x=1,2,3) to wake up the MCU from Stop mode using baudrates up to 204 Kbaud. The wake up events from Stop mode are programmable and can be:

- Start bit detection
- Any received data frame
- A specific programmed data frame
- All USART interfaces can be served by the DMA controller

3.1.2 Tools for configuration and programming of the microcontroller

In order to configure properly the STM32L412KB device I used STM32CubeMX tool to generate code to initialize the system, peripherals and middleware. The C code based on the configuration selected was then used within MDKARM development environment. STM32Cube is an STMicroelectronics original initiative to significantly improve designer's productivity by reducing development effort, time and cost.

STM32Cube includes [9]:

- A set of user-friendly software development tools to cover project development from the conception to the realization, among which:

  - STM32CubeMX, a graphical software configuration tool that allows the automatic generation of C initialization code using graphical wizards
  - STM32CubeIDE, an all-in-one development tool with peripheral configuration, code generation, code compilation, and debug features
  - STM32CubeProgrammer (STM32CubeProg), a programming tool available in graphical and command-line versions
  - STM32CubeMonitor-Power (STM32CubeMonPwr), a monitoring tool to measure and help in the optimization of the power consumption of the MCU

- STM32Cube MCU MPU Packages, comprehensive embedded-software platforms specific to each microcontroller and microprocessor series (such as STM32CubeL4 for the STM32L4 Series and STM32L4+ Series), which include:
  - STM32Cube hardware abstraction layer (HAL), ensuring maximized portability across the STM32 portfolio
  - STM32Cube low-layer APIs, ensuring the best performance and footprints with a high degree of user control over the HW
  - A consistent set of middleware components such as FAT file system, RTOS, USB Host and Device, Touch library, and Graphics
  - All embedded software utilities with full sets of peripheral and applicative examples

The MDK-ARM (Microcontroller Development Kit) is the complete software development environment for ARM7TM, ARM9TM, CortexTM-M, and Cortex-R4 processor-based devices. MDK is specifically designed for microcontroller applications and combines the ARM C/C++ Compiler with the Keil RTX real-time operating system and middleware libraries. All tools are integrated into µVision which includes project management, editor and debugger in a single easy-to-use environment. The fully integrated ARM C/C++ Compiler offers significant code-size and performance benefits to the embedded developer, however, MDK can also be used with the GNU GCC Compiler.

3.1.3 Tool for technical computing

Another important tool used to evaluate results especially for analog and digital filters part is MATLAB.

MATLAB (matrix laboratory) is a multi-paradigm numerical computing environment and proprietary programming language developed by MathWorks. MATLAB allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages. Although MATLAB is intended primarily for numerical computing, an optional toolbox uses the MuPAD symbolic engine allowing access to symbolic computing abilities. An additional package, Simulink, adds graphical multi-domain simulation and model-based design for dynamic and embedded systems [10].

3.2 Implementation

3.2.1 Analog To Digital block implementation

To assess the front-end rejection of low-frequency common-mode environmental noise, in according to differential measurement technique described in Chapter 2, I set the following parameters:

- ADC sampling frequency: 2 MHz;
- Charge-discharge frequency, Fm: 1600 Hz;
- Number of ADC samples per segment: $256 = 4^4$ (n = 4 additional accuracy bits after processing);
- Variable guard time between segments and ramp ends, NG, up to 55 ADC samples (Figure 2.3).

The Analog to Digital converter has to be synchronized with a square wave whose frequency is 1600 Hz due to the fact that it drives the triangular wave. First, for the programming part, I use STM32Cube tool, once I select the proper board (STM Nucleo-L412KB) the tool shows the virtual board with all the pins available. In order to comply with the specifications listed before the microcontroller is programmed as follow: The timer TIM2 is driven by internal clock source at 80 MHz. It is triggered by external source ETR1 mapped to PIN5 (where square wave is applied). TIM2 has to trigger each sample of the ADC, since we need a 2 MHz sampling frequency and the internal clock is set to 80 MHz the counter period is set to 40 (Figure 3.6).

ADC1 channel 6, that corresponds to PIN PA1 is configured in single-ended input mode, so the analog voltage to be converted for the channel is the difference between the external voltage VINP[i] (positive input) and VREF . ADC has 12 bit resolution, conversion is triggered by rising edge of update event of TIM2. ADC works together with DMA in order to save power and not involve CPU (Figure 3.7 and Figure 3.8).

DMA is configured in circular mode to handle circular buffers and continuous data flows. The data size selected is Word since we have 12 bits to transfer and the memory pointer is automatically incremented after each transfer as can be seen from Figure 3.9.

USART2 is configured in asynchronous mode, with a Baud Rate of 115200 Bits/s, a word length of 8 bits including Parity, no Parity bit and 1 stop bit. Communication channel is used to send data from sensor to central host in order to perform further elaboration and to verify results (Figure 3.10).

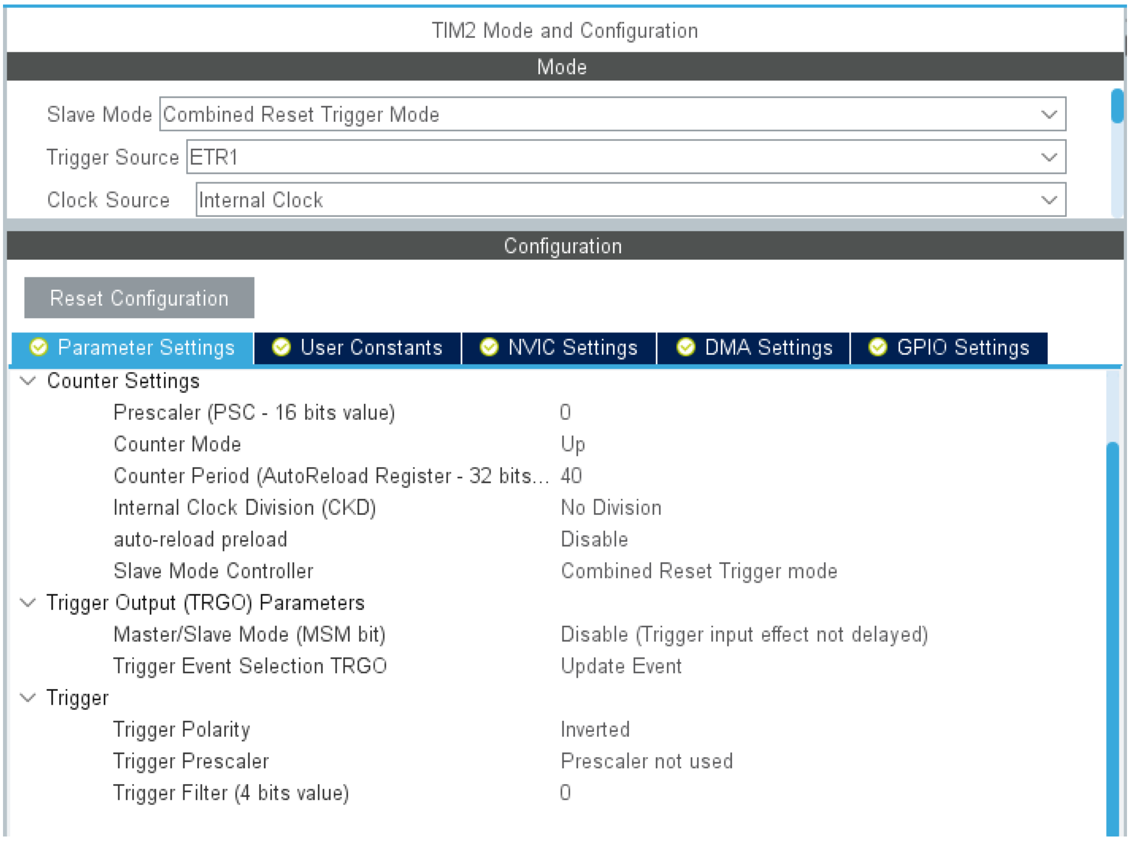

Figure 3.6. Timer TIM2 configuration view

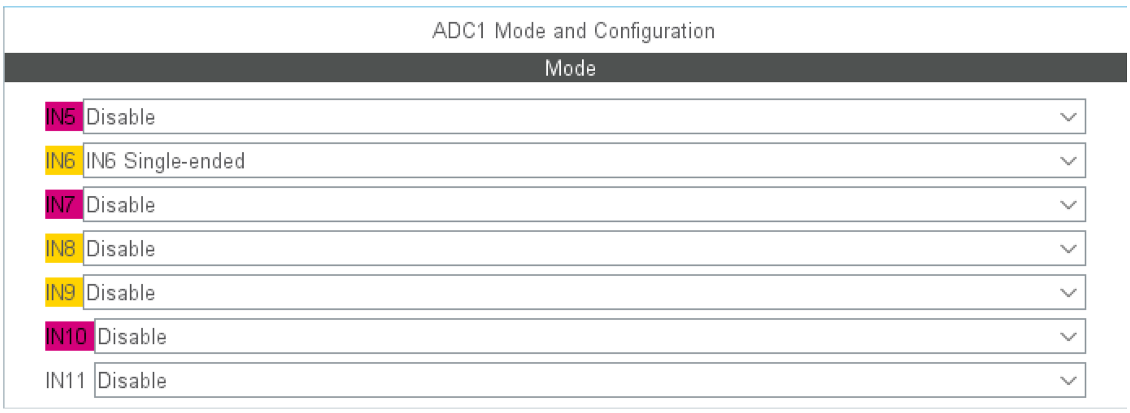

Figure 3.7. Analog to Digital Converter channel configuration view

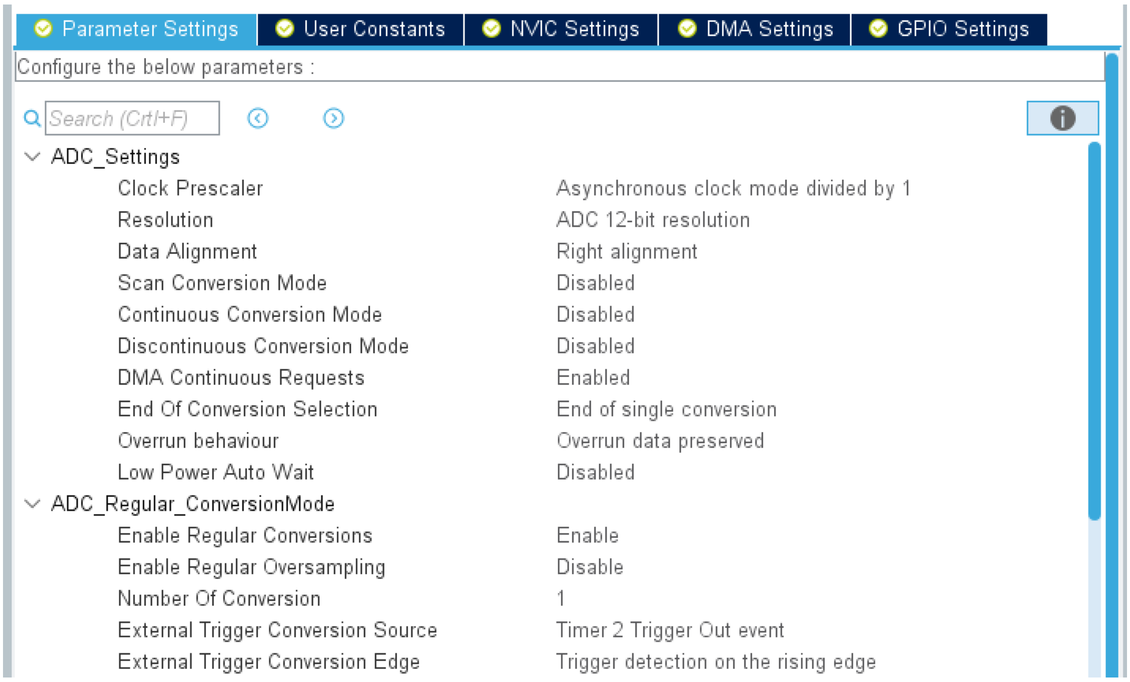

Figure 3.8. Analog to Digital Converter configuration view

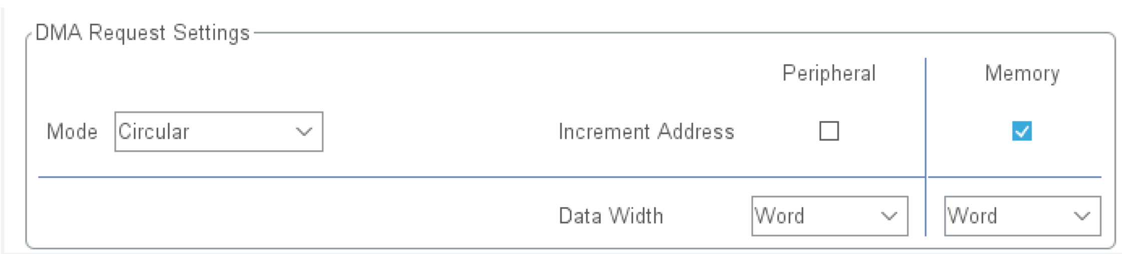

Figure 3.9. Direct Memory Access configuration view

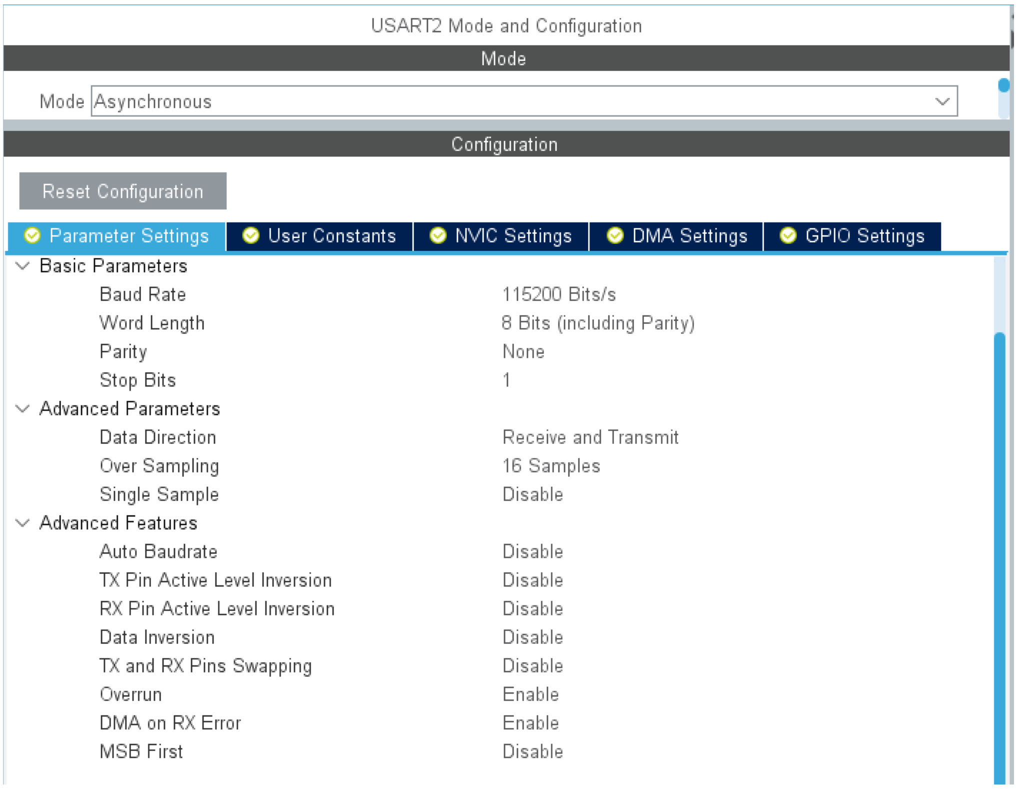

Figure 3.10. Universal Synchronous-Asynchronous Receiver/Transmitter configuration view

Figure 3.11 shows a schematic view of the configurations and interconnections of the elements described above.

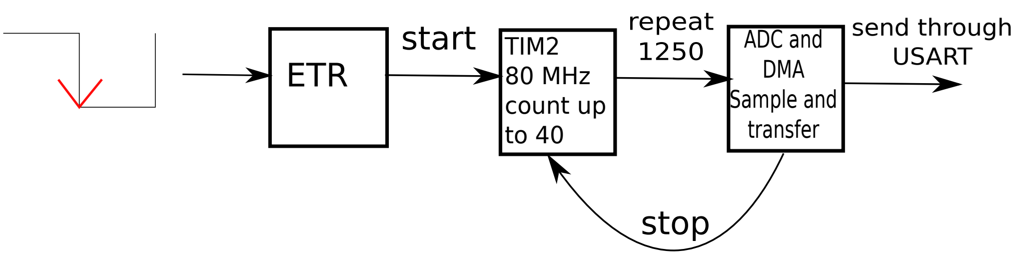

Figure 3.11. Block schematic of the ADC system operation

After the hardware configuration of the components, STM32Cube generates C code. At this point I use MDK-ARM in order to develop the application and debug the board. I generate different programs for different experiments that are discussed in Chapter 4.

3.2.2 Analog Filter block implementation

The continuous signal to be sampled must not include significant frequency components greater than the Nyquist frequency. For this purpose, it is recommended to low-pass filter the continuous signal before sampling, especially in the presence of high-frequency noise. The analog low-pass filter used for this purpose is known as the antialiasing filter.

Because a low-pass filter can slow down the system by attenuating high-frequency dynamics, the cut-off frequency of the low-pass filter must be higher than the bandwidth of the signal so as not to degrade the transient response. The cut-off frequency of the low-pass filter can be chosen as 10 times the bandwidth of the signal to minimize its effect, and then the sampling frequency can be chosen 10 times, or more, higher than the filter cut-off frequency so there is a sufficient attenuation above the Nyquist frequency [11]. For the reasons above, I am interested in implementing active first order analog low-pass filters with cut-off frequency of 15 kHz and 20 kHz that is about 10 times higher than the measurement frequency (1.6 kHz). Analog filter can reduce high frequency noise and the ADC aliasing. The characteristic plots of the filters are shown in Figure 3.12 and Figure 3.13.

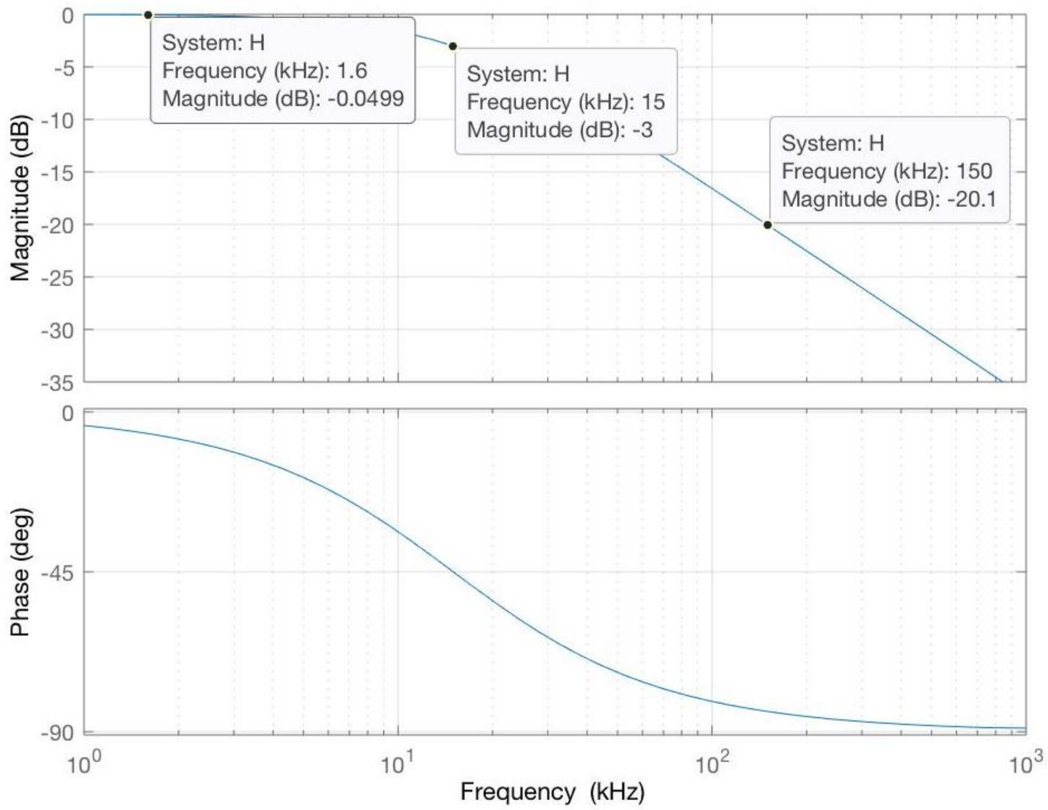

Figure 3.12. Characteristic plot of first order low-pass analog filter with cut off frequency 15 kHz

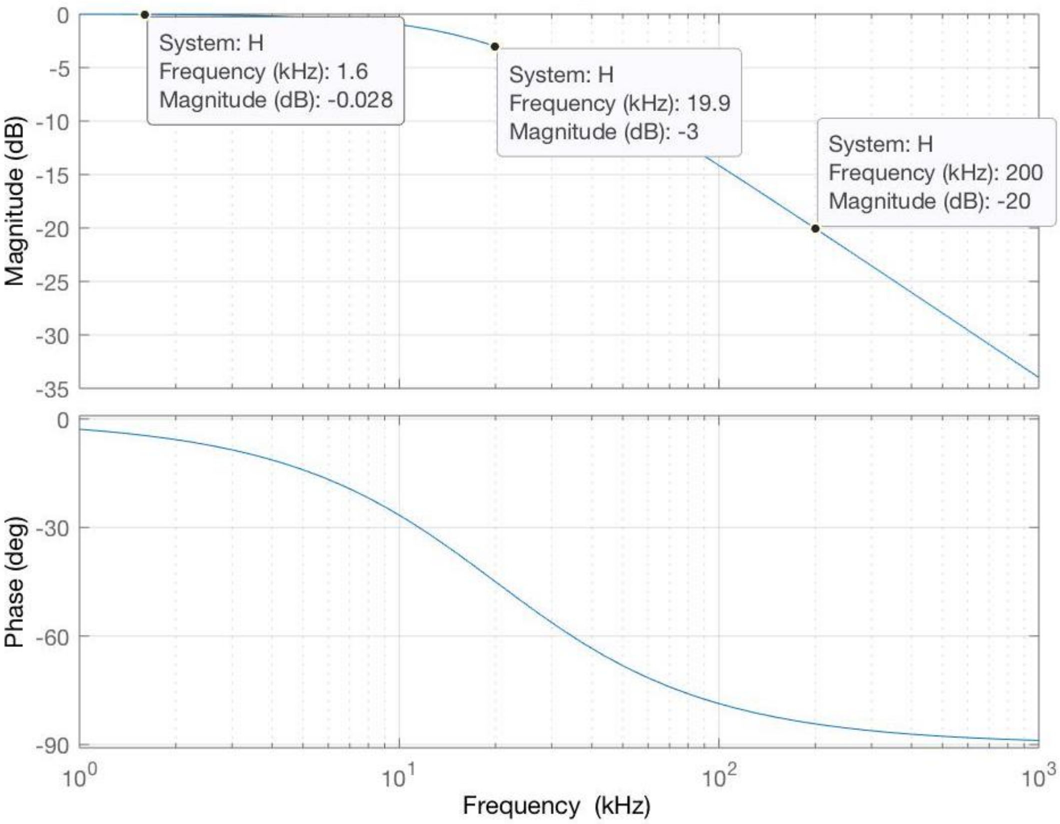

Figure 3.13. Characteristic plot of first order low-pass analog filter with cut off frequency 20 kHz

The cutoff frequency f_c depends on resistance R and capacitance C of the network:

$$f_c = \frac{1}{2\pi RC} \tag{3.1}$$

thus I obtain 20 kHz by using one 3.3 nF capacitor and a series of one 2 kOhm resistor, four 100 Ohm resistors and one 10 Ohm resistor thus obtaining a total resistance of R=2.41 kOhm, whereas I obtain 15 kHz by using one 3.3 nF capacitor and a series of one 1 kOhm resistor, one 2 kOhm resistor and one 220 Ohm resistor for a total of R=3.22 kOhm.

I use the operational amplifier (OPAMP) available in the microprocessor in follower mode, since I want a non inverter filter, with no amplification. Thus, the inverting input is directly connected to output of the amplifier by creating a closed-loop, while at the non inverting input I apply the RC network. Figure 3.14 shows an implementation scheme of the analog lowpass filter with board pins involved whereas Figure 3.15 displays the configuration settings of the OPAMP.

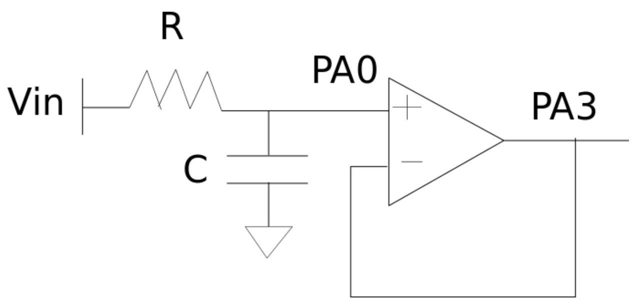

Figure 3.14. Implementation scheme of Analog lowpass filter with pins referencing to microcontroller

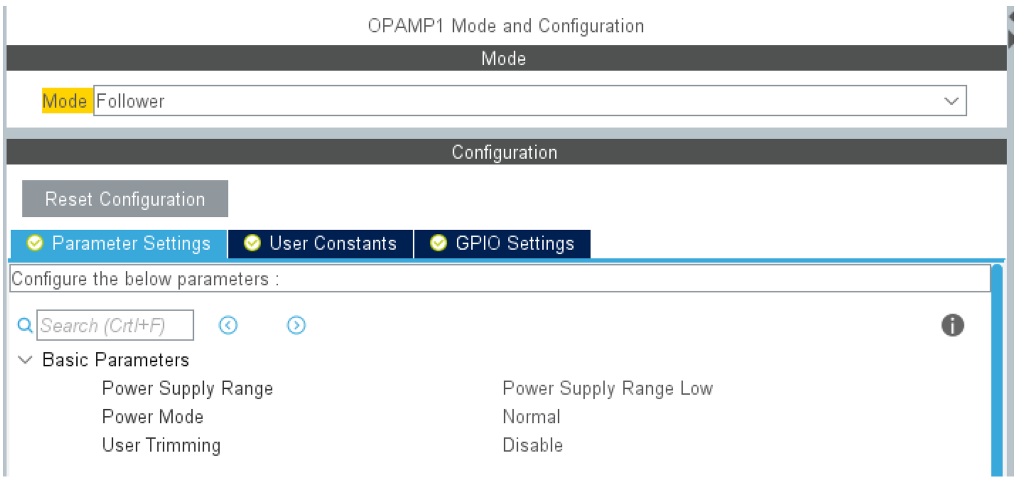

Figure 3.15. Operational Amplifier configuration view

3.2.3 Digital Filter block implementation

A digital filter is used to reduce the noise outside the frequency bands of interest, that is between 0 Hz an 3 Hz.

A digital filter is a system that performs mathematical operations on a sampled, discrete-time signal to reduce or enhance certain aspects of that signal. Digital filters may be more expensive than an equivalent analog filter due to their increased complexity, but they make practical many designs that are impractical or impossible as analog filters. Digital filters can often be made very high order, and are often finite impulse response filters which allows for linear phase response. However, digital filters do introduce a higher fundamental latency to the system.

Many digital filters are based on the fast fourier transform (FFT), a mathematical algorithm that quickly extracts the frequency spectrum of a signal, allowing the spectrum to be manipulated (such as to create very high order band-pass filters) before converting the modified spectrum back into a time-series signal with an inverse FFT operation [12].

In my research I decide to evaluate three different types of digital filters: FIR, Chebyshev2 and Elliptic. Some characteristics are described below:

- FIR: FIR stands for Finite Impulse Response. The term finite impulse response arises because the filter output is computed as a weighted, finite term sum, of past, present, and perhaps future values of the filter input. A FIR filter is based on a feed-forward difference equation. Feed-forward means that there is no feedback of past or future outputs to form the present output, just input related terms:

$$y[n] = \sum_{k=-M1}^{M2} b_k x[n-k] \text{ M1 and M2 are finite} \tag{3.2}$$

Chebyshev2 and Elliptic belongs to infinite impulse response (IIR) digital filters. IIR is a type of filter which re-uses one or more of its outputs as an input. This feedback typically results in an unending impulse response (commonly referred to as infinite impulse response (IIR)), characterized by either exponentially growing, decaying, or sinusoidal signal output components.

- Chebyshev2: Chebyshev filters minimize the error between the idealized and the actual filter characteristic over the range of the filter but with ripples in the stopband. They are not as sharp as the elliptic one, but they show fewer ripples over the bandwidth.

- Elliptic: An elliptic filter (also known as a Cauer filter) is a signal processing filter with equalized ripple (equiripple) behavior in both the passband and the stopband. The amount of ripple in each band is independently adjustable, and no other filter of equal order can have a faster transition in gain between the passband and the stopband, for the given values of ripple (whether the ripple is equalized or not).

I design filters by using the "Filter designer" app available in MATLAB. First I present ideal characteristic mask of digital filter with cut off frequency of 1 Hz (Figure 3.16), and then digital filters implementations with minimum order to match the magnitude reduction requirement that is set to -80 dB (Figure 3.17, Figure 3.18, Figure 3.19). Second I present digital filters with cut off frequency of 3 Hz and minimum order to match the stopband magnitude reduction -80 dB (Figure 3.21, Figure 3.22, Figure 3.23), characteristic mask is shown in Figure 3.20).

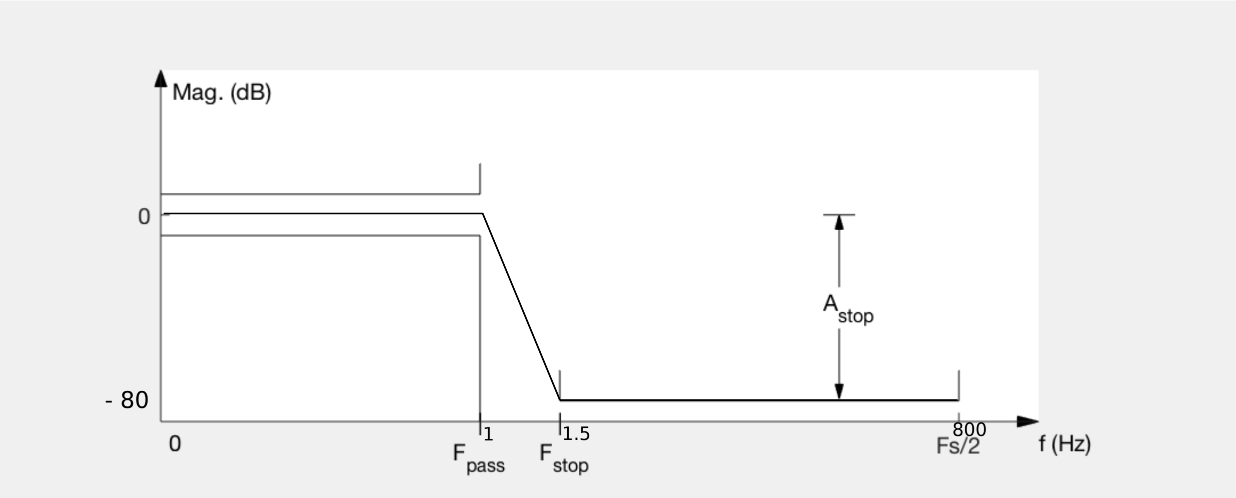

Figure 3.16. Characteristic mask of digital filter with cut off frequency 1 Hz and (Astop) -80 dB reduction in stopband

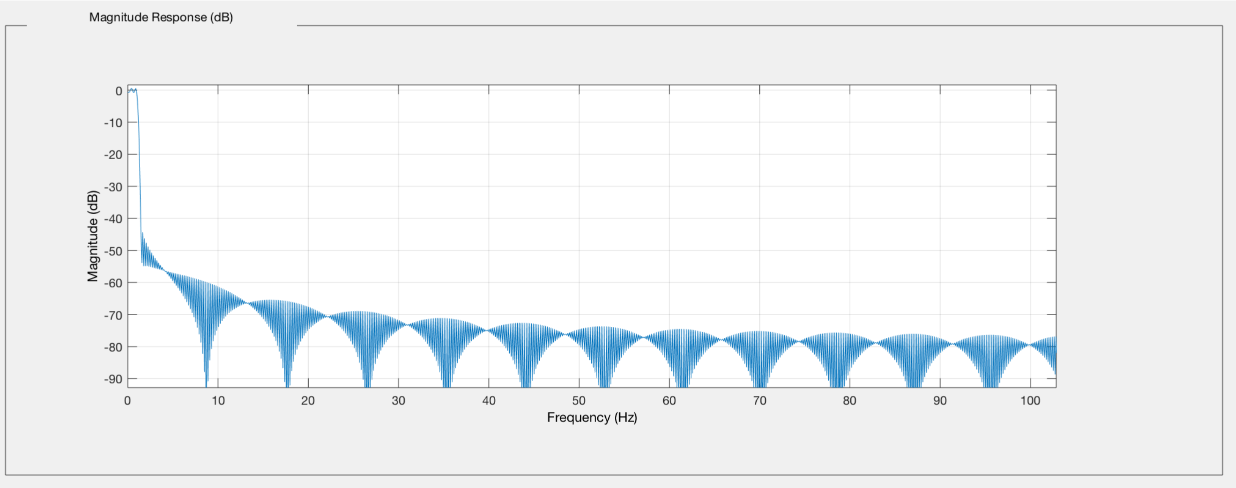

Figure 3.17. FIR equiripple filter with cutoff frequency of 1 Hz and order 8095

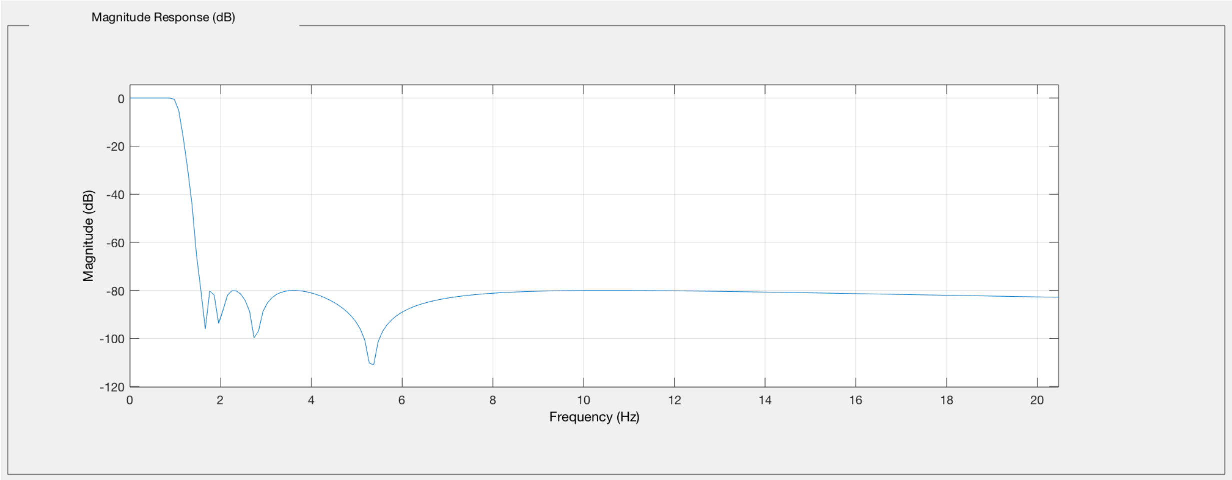

Figure 3.18. Chebyshev filter with cutoff frequency of 1 Hz and order 11

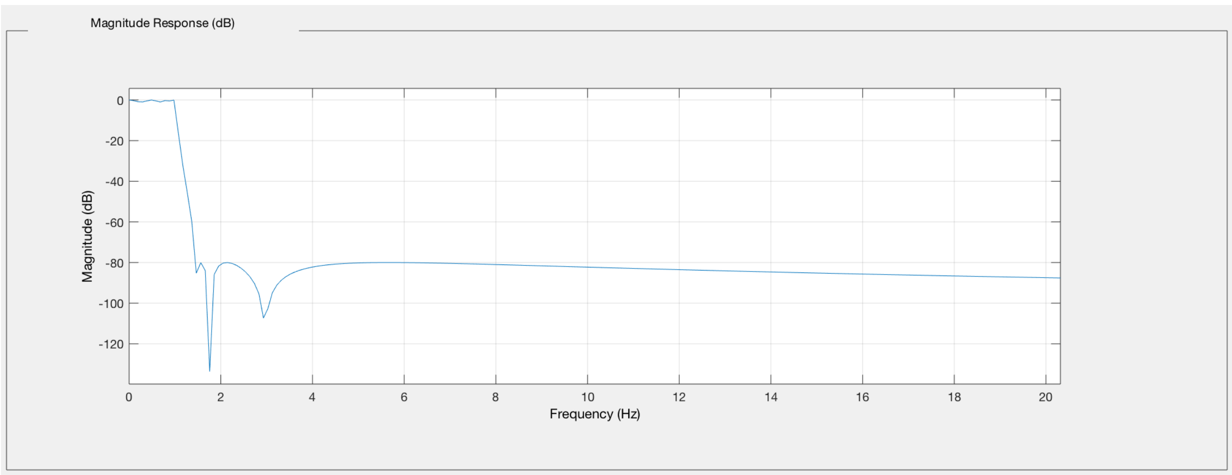

Figure 3.19. Elliptic filter with cutoff frequency of 1 Hz and order 7

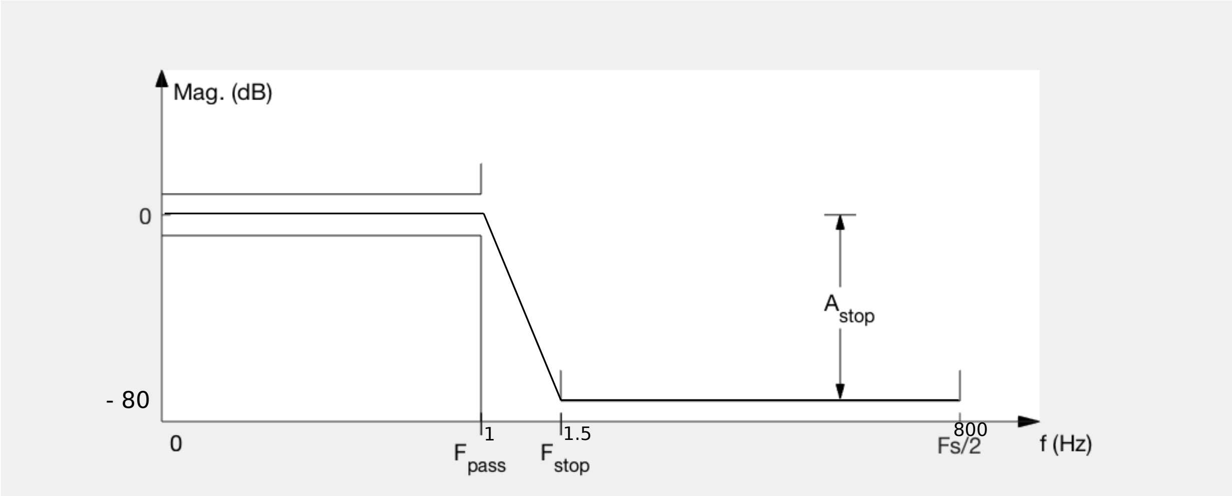

Figure 3.20. Characteristic mask of digital filter with cut off frequency 3 Hz and (Astop) -80 dB reduction in stopband

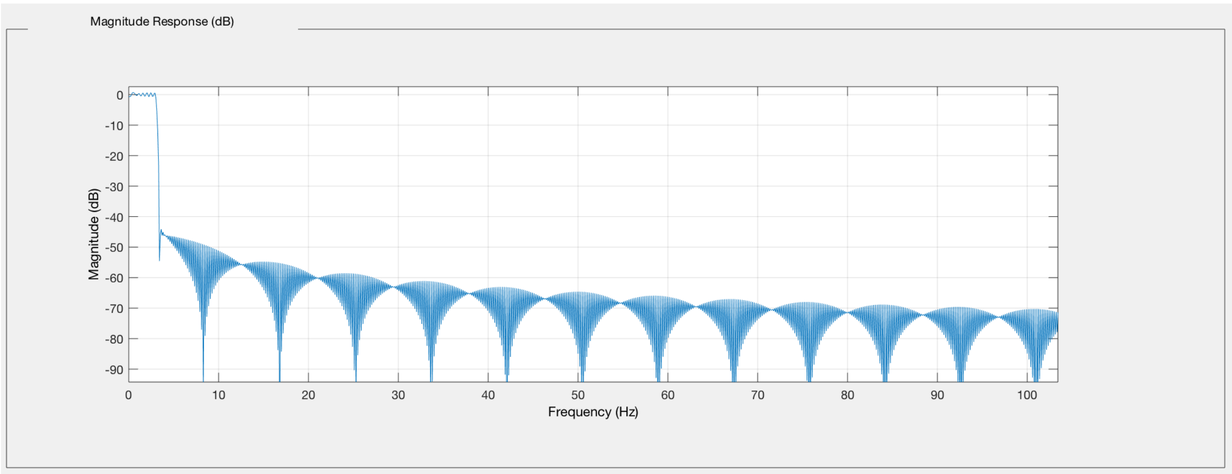

Figure 3.21. FIR equiripple filter with cutoff frequency of 3 Hz and order 8095

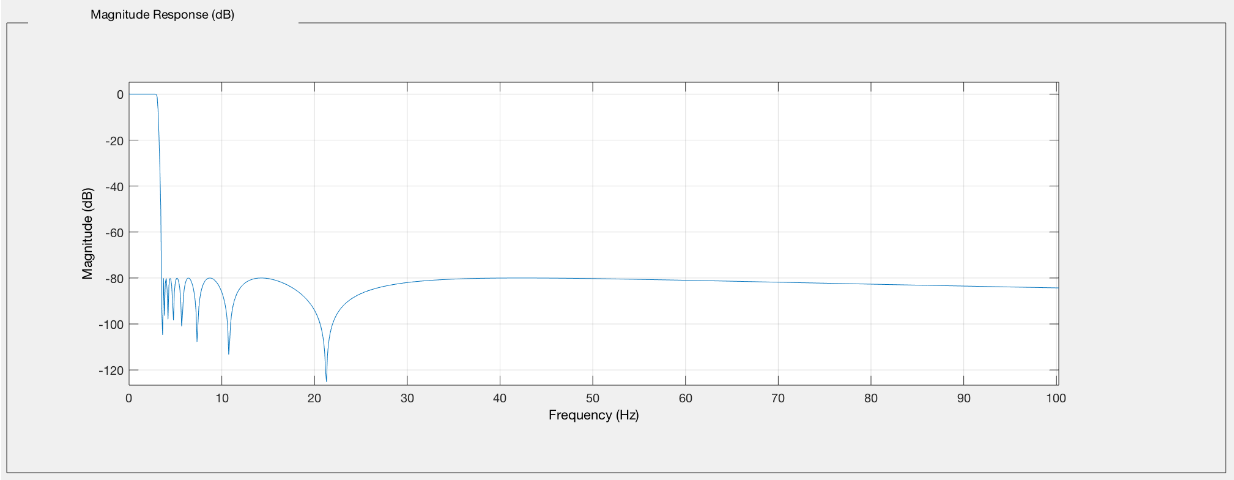

Figure 3.22. Chebyshev filter with cutoff frequency of 3 Hz and order 19

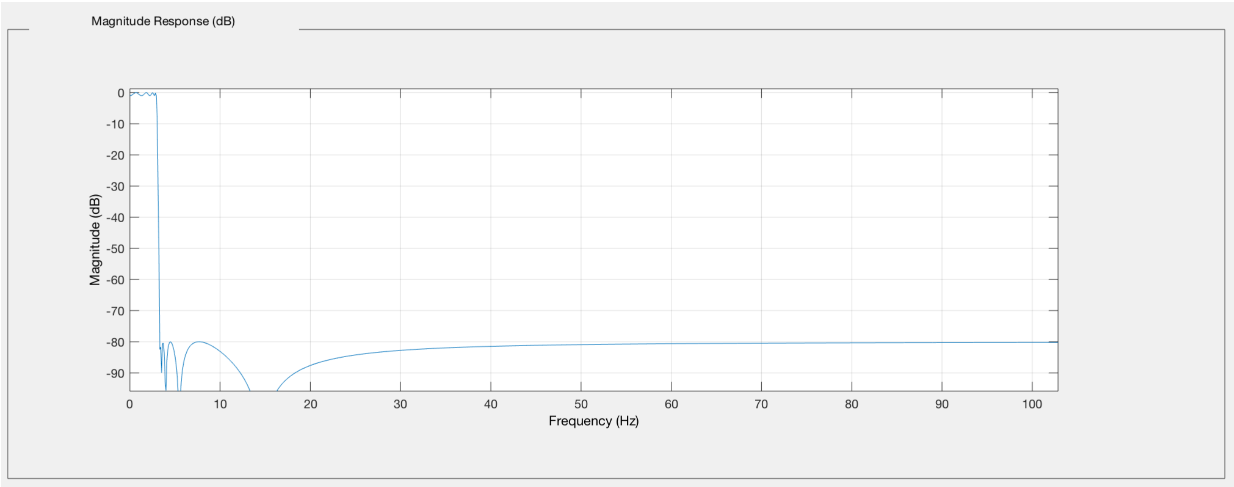

Figure 3.23. Elliptic filter with cutoff frequency of 3 Hz and order 10

Chapter 4

Experimental results

This chapter focuses on experiments and their results. With reference to block schematic (Figure 3.2), first I describe experiments related to the Analog to Digital Converter (ADC) block. Second, I describe how I test the behavior of the analog filters. This presentation is followed by discussion on results collected in the output of filters. Third, I discuss about the last block: digital filter. Results obtained from different implemented digital filters (FIR, Chebyshev and Elliptic) are comparatively discussed.

In absence of the hardware implementation of the analog front-end interfacing with the sensor plate, I have emulated its operation using a triangular wave from a signal generator to test the operation and the performance of the analog to digital interface of the microcontroller and several signal conditioning and post-processing techniques.

4.1 Testing Analog to Digital Converter operation and performance

4.1.1 Synchronization and timing

In the Implementation Section, I describe how the ADC is configured in order to be compliant with the front-end specifications.

The first test is based on applying a triangular wave to the input of analog to digital converter and a square wave, perfectly synchronized, to the pin that is used to trigger the timer (with reference to Figure 3.11). Test goals are:

- Checking that ADC sampling starts when the driver square wave goes to low level and so when triangular wave starts decreasing.
- Checking that on each slope there are exactly 625 ADC samples.

I generate a square wave between 0 V and 3 V with a frequency of 1.6 kHz using a signal generator and I apply it to PIN PA5 (trigger source for TIM2) of the board, then I generate a triangular wave between 0 V and 3 V with frequency 1.6 kHz with the same signal generator and perfectly synchronized with the square wave (During high state of the square the triangular rises up, while during the low state of square wave the triangular falls down), and I apply it to PIN PA1 (input of ADC).

In the main.c file after the configuration of the system and peripherals, I start the timer TIM2 by invoking proper HAL function and then I start the ADC with DMA. In this first step I want to check the synchronization and the number of samples for each slope of the triangular wave, thus I create a buffer called adcVal of 2500 values (2 periods of triangular wave) and then once the DMA finishes its work I send all the values through USART to my computer.

```
/* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* Configure the system clock */
  SystemClock_Config();

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_DMA_Init();
  MX_ADC1_Init();
  MX_TIM2_Init();
  MX_USART2_UART_Init();

  /* USER CODE BEGIN 2 */
  HAL_TIM_Base_Start(&htim2);
  HAL_ADC_Start_DMA(&hadc1,adcVal,2500);
```

```
/* USER CODE BEGIN 4 */
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef *hadc)
{ int a;
  HAL_TIM_Base_Stop(&htim2);
  HAL_UART_Transmit(&huart2,(uint8_t *)"start\r\n",strlen("start\r\n"),
      HAL_MAX_DELAY);
  for(a=0;a<2500;a++){
    sprintf(ch,"%d\r\n",adcVal[a]);
    HAL_UART_Transmit(&huart2,(uint8_t *) &ch,strlen(ch),HAL_MAX_DELAY);
  }
}
/* USER CODE END 4 */
```

Then I plot values with MATLAB (Figure 4.1). X axis shows time in seconds while Y axis represents voltage expressed over 12 bit, this means between value 0 an $2^{12}-1 = 4095$. To evaluate number of points in a slant I identify X value associated to the lowest point of falling slant and X value of the highest point of rising slant, I performe a subtraction and then I divide result by sampling period (Ts = 0.5 us).

As I expect the plot starts from the falling slope of the triangular and the lowest point corresponds to X = 0.0003125 as can be seen from the Figure 4.1. This means that I have exactly 625 points in the falling slant ($\frac{0.0003125}{0.0000005} = 625$). I do the same checks over the other slopes.

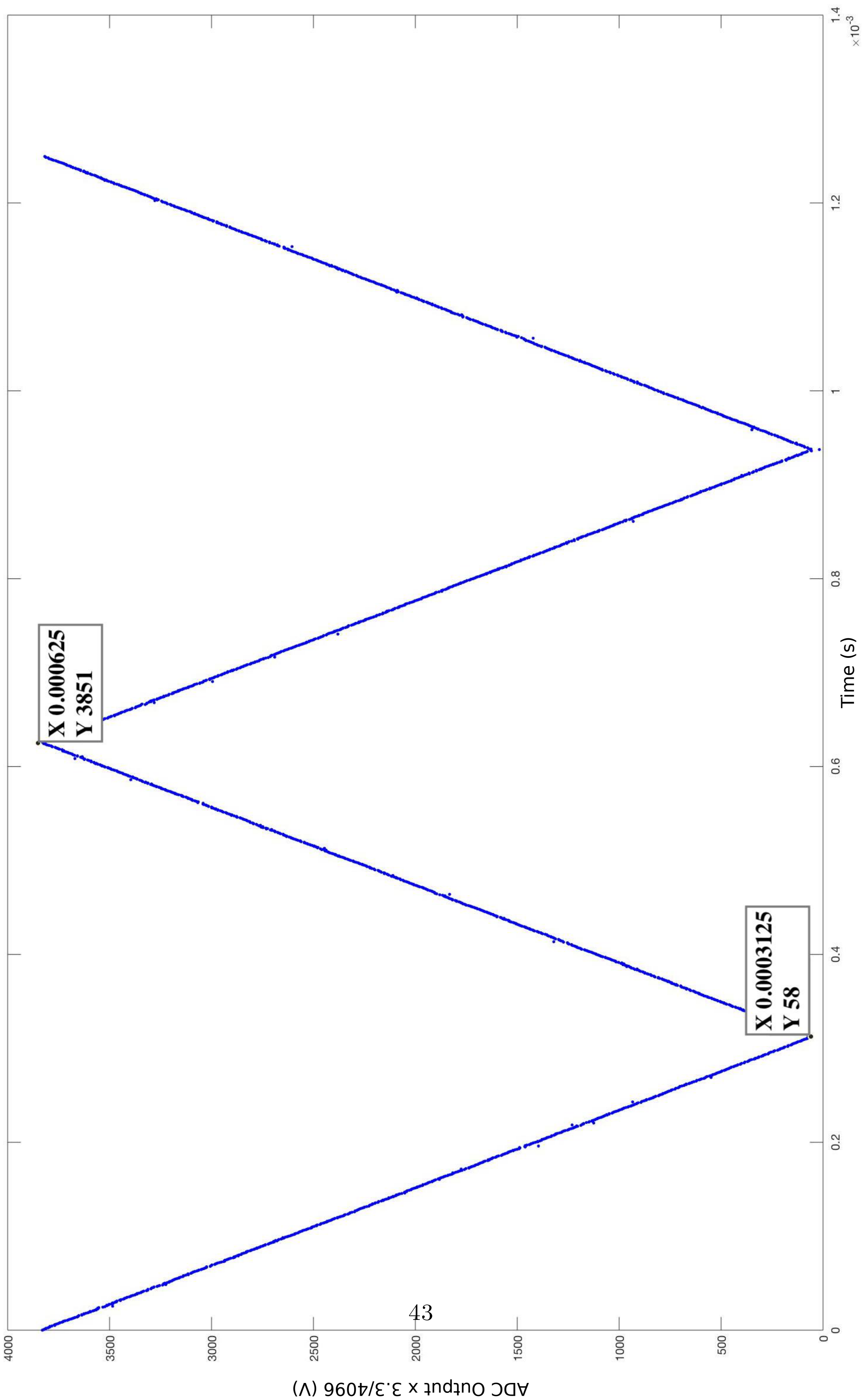

Figure 4.1. Two periods of triangular wave sampled with a sampling period of 0.5 us

The Figure also shows some non-aligned points which represent high frequency noise, however, ramp-ends seem to be well shaped. During the acquisition of the samples I switch off the square wave to verify the dependency with the sampling, ADC is stopped accordingly. Then, as soon as the square wave restarts, ETR is triggered on the falling edge and ADC begins to sample the triangular wave again.

4.1.2 Data acquisition

The second test focuses on calculating the capacitance value through differential measuring technique plus averaging or oversampling & decimation.

As described in [13], averaging consists in summing all the values of a segment in an accumulator and then averaging it. Averaging is not intended to improve the resolution of the result. Rather, it is a simple and effective way to smooth the input waveform and reduce sensor noise.

The method for increasing the effective resolution of the ADC is oversampling and decimation. This technique involves oversampling of the input signal so that a number of samples can be used to compute a virtual result with greater accuracy than a single real sample can provide. Oversampling requires sampling the signal 4^n times in addition to the minimum required frequency, where n is the number of additional bits we want to achieve. Once the oversampling is done, samples are summed and then the result is divided by the number of total bits. Such a process is commonly referred to as decimation. In addition to yielding a more accurate approximation of the signal value during a given sampling interval, decimation also helps to improve the signal-to-noise ratio (SNR) of the input signal, since it depends on the number of bits of resolution:

$$SNR(dB) = (6.02 * ENOB) \tag{4.1}$$

The test requires the generation of a triangular wave at the input of analog to digital converter and a square wave, perfectly synchronized to the triangular, at the pin used to trigger the timer. Test goals are:

- sampling the input signal.
- calculating the capacitance through averaging technique.
- calculating the capacitance through oversampling & decimation technique.
- computing the standard deviations for the two traces and comparing.

In my setting, the differential measuring technique requires two slants of triangular wave sampled 1250 times. Due to these requirements, the buffer used by the DMA is set to 1250 and as soon as it finishes the transfer by using void HAL_ADC_ConvCpltCallback (ADC_HandleTypeDef *hadc) function (that is invoked automatically at the end of DMA transfer), I stop the timer TIM2, and the ADC, I compute capacitance, I send it through USART to my computer, and then I reactivate the timer (TIM2) that is triggered by the falling edge of the square wave (see Figure 3.11).

In my system I have 256 samples each segment (see Figure 2.3), this corresponds to have oversampling n=4, which increases the ADC conversion resolution from 12 bit to 16 bit.

```c
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef *hadc)
{
  HAL_TIM_Base_Stop(&htim2);
  V1=0;
  for(i=56;i<312;i++){
    V1+=adcVal[i];
  }

  V2=0;
  for(l=313;l<569;l++){
    V2+=adcVal[l];
  }

  SLA=(V1-V2)/256;
  SLO=(V1-V2)/16;

  V3=0;
  for(j=681;j<937;j++){
    V3+=adcVal[j];
  }

  V4=0;
  for(m=938;m<1194;m++){
    V4+=adcVal[m];
  }

  SRA=(V4-V3)/256;
  SRO=(V4-V3)/16;
  Cplateavg=KA/(33*(SLA+SRA));
  Cplatedec=KO/(33*(SLO+SRO));
  sprintf(ch,"%ld  %ld\r\n",Cplateavg,Cplatedec);
  HAL_UART_Transmit(&huart2,(uint8_t *) &ch,strlen(ch),HAL_MAX_DELAY);

  HAL_TIM_Base_Start(&htim2);
}
```

Once the system is ready, I need to evaluate differences on the two methods and check if oversampling & decimation has a better sensibility to variations.

I apply a square wave between 0 V and 3 V with a frequency of 1.6 kHz to PIN PA5 (trigger source for TIM2) of the board, then I apply a triangular wave between 0 V and 3 V with frequency 1.6 kHz to PIN PA1 (input of ADC)(see Figure 3.11). I let system acquire for different slots of time (600 seconds, 10 seconds and 1 second) and calculate the capacitance using averaging and oversampling & decimation. Figure 4.2 shows the capacitance calculation in time for the two methods. Then I retrieve standard deviation for the two traces by using "std" MATLAB function (Table 4.1).

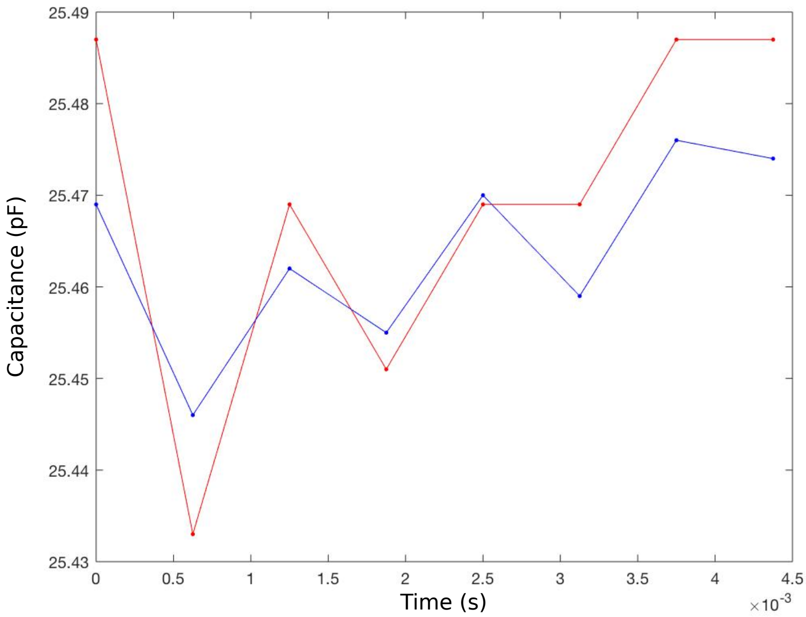

Figure 4.2. Plot of Cplate values calculated through averaging (red) and through oversampling & decimation (blue)

Table4.1 Standard deviation comparison between avg and over&dec

	STD (fF)		
Type	*600 s*	*10 s*	*1 s*
averaging	4.3621	4.3333	4.7967
over&dec	3.8692	3.6669	4.1307

From the plot I can observe that averaging (red line) sometimes is stuck at the same level for two, three points, while oversampling & decimation (blue line) never shows the same value toward following points. This means that the latter is able to capture smaller variations with respect to averaging, this statement is confirmed also by the standard deviations reported in the table. Standard deviation is a measure of the amount of dispersion of a set of values. Oversampling and decimation increases the resolution of ADC up to 16 bit, providing an LSB = 50 uV, so sensitive to very small voltage variations. Since capacitance calculation depends on voltage (see Equation 2.3), it inherits the same accuracy improvement.

4.2 Analog filter tests

As I report in the Chapter 3, I implement the analog filters by exploiting the operational amplifier (OPAMP), properly configured, available in the microcontroller. It is necessary to start also the OPAMP after the configuration of the system and the peripherals in the main.c file, as shown in the following code excerpt.

```
/* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* Configure the system clock */
  SystemClock_Config();

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_DMA_Init();
  MX_ADC1_Init();
  MX_OPAMP1_Init();
  MX_TIM2_Init();
  MX_USART2_UART_Init();

  /* USER CODE BEGIN 2 */
  HAL_OPAMP_Start(&hopamp1);
  HAL_ADC_Start_DMA(&hadc1,adcVal,1250);
  HAL_TIM_Base_Start(&htim2);
  /* USER CODE END 2 */
```

4.2.1 Noise reduction

I analyze the effects of the anti-alias low pass filter for two cutoff frequencies of 20 kHz and 15 kHz, whose implementation is reported in Chapter 3. The test is based on applying a triangular wave at the input of the filter, connecting the output of filter to the input of analog to digital converter, and calculation of the capacitance. The experiment aims at:

- evaluating triangular wave after filters and checking how much is shifted.
- computing capacitance values by using the averaging and the oversampling & decimation techniques.
- computing the standard deviation for the 4 different combinations and comparing:
  - analog filter 20 kHz and averaging.
  - analog filter 20 kHz and oversampling & decimation.
  - analog filter 15 kHz and averaging.
  - analog filter 15 kHz and oversampling & decimation.

Theoretically, after the analog filters capacitance values should be more stable.

To test the behavior, I apply a square wave between 0 V and 3 V with a frequency of 1.6 kHz to pin PA5 (ETR1), a triangular wave between 0 V and 3 V with frequency of 1.6 kHz, perfectly synchronized with the square wave, to the input of RC network of the filter, and I connect pin PA3 (output of OPAMP) to pin PA1 (input of ADC).

In Figure 4.3, I compare the triangular wave without filter and triangular waves after filter 15 kHz and filter 20 kHz. The X axis represents time in seconds whereas Y axis represents the ADC output x 3.3/4096 V per LSB.

I also compute the absolute difference to emphasize the error, and I plot it in Figure 4.4. The X axis represents the time in seconds, whereas the Y axis represents the absolute error.

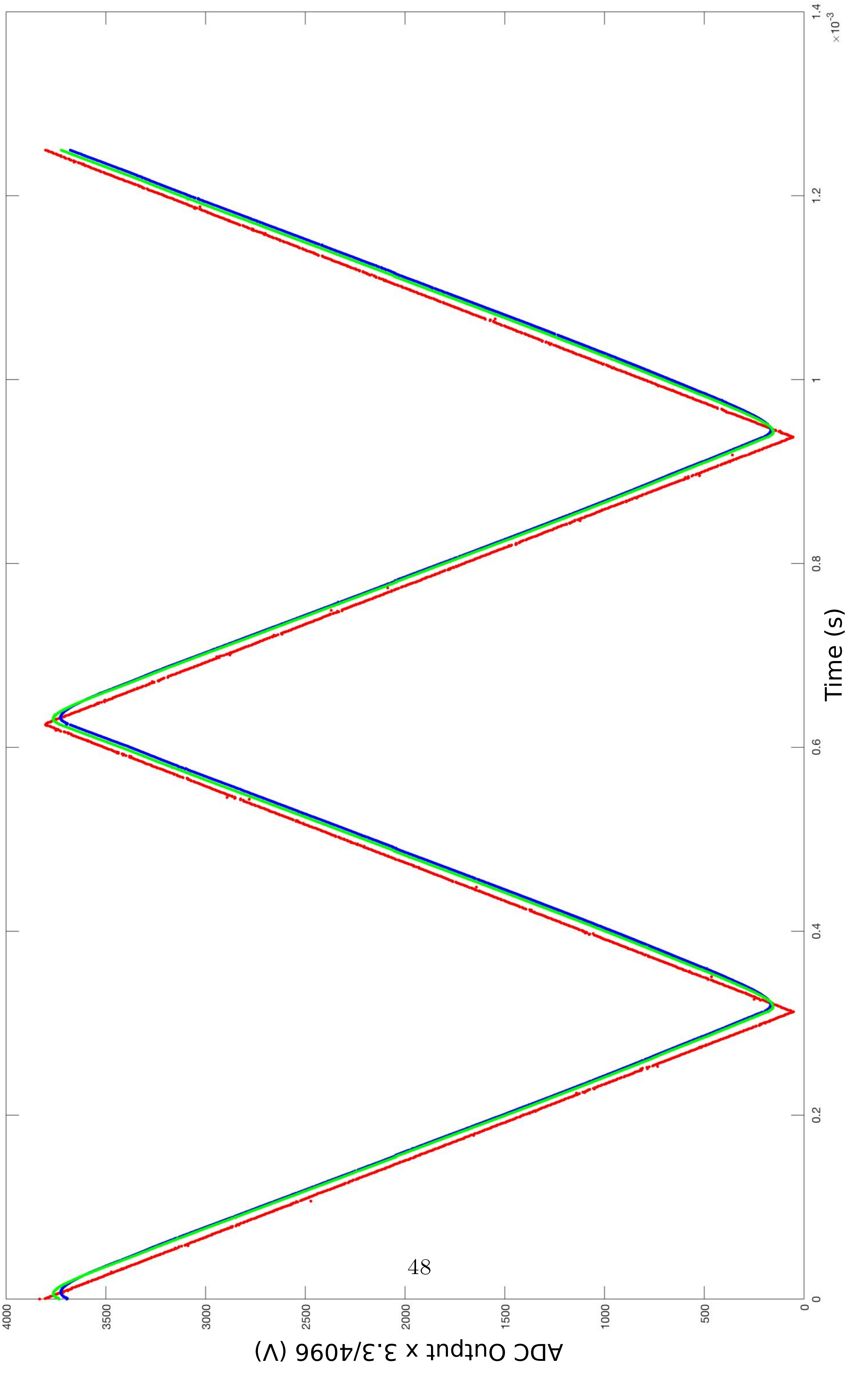

Figure 4.3. Triangular waves after analog filters. Red: original triangular; Green: triangular after 20 kHz filter; Blue: triangular after 15 kHz filter

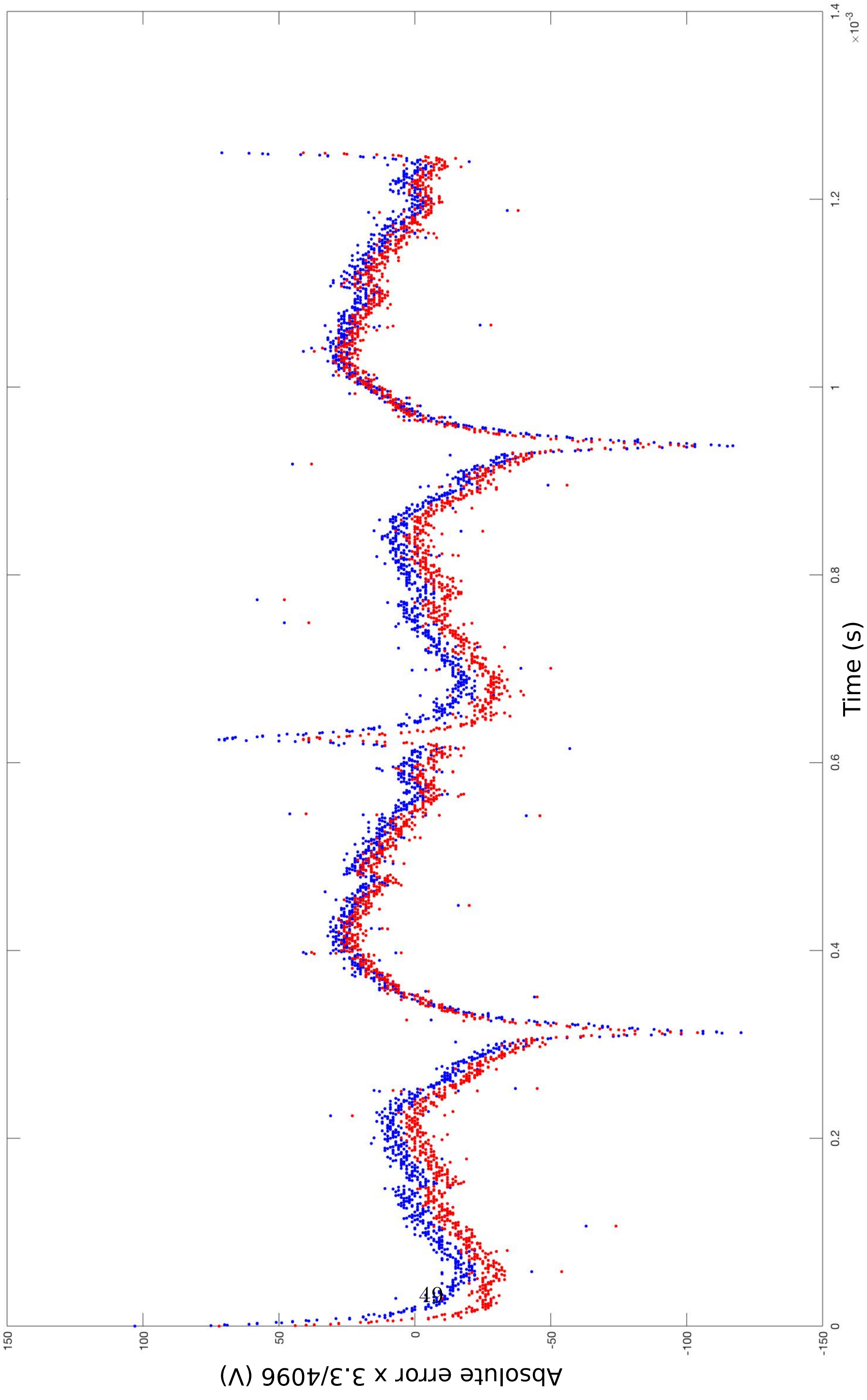

Figure 4.4. Absolute difference between original triangular wave and filtered waves. Blue: difference between original and 15 kHz filtered; Red: difference between original and 20 kHz filtered

As expected, the filtered waves are delayed and, due to the non-linearities of active components and OPAMP, they have ramp-end distortions and the slopes of the ramps are slightly changed, but the waves are less noisy. In order to realign waves, I delay the ADC sampling by 20 samples (10 us) for 15 kHz filter and by 15 samples (7.5 us) for 20 kHz filter. The result is shown in Figure 4.5.

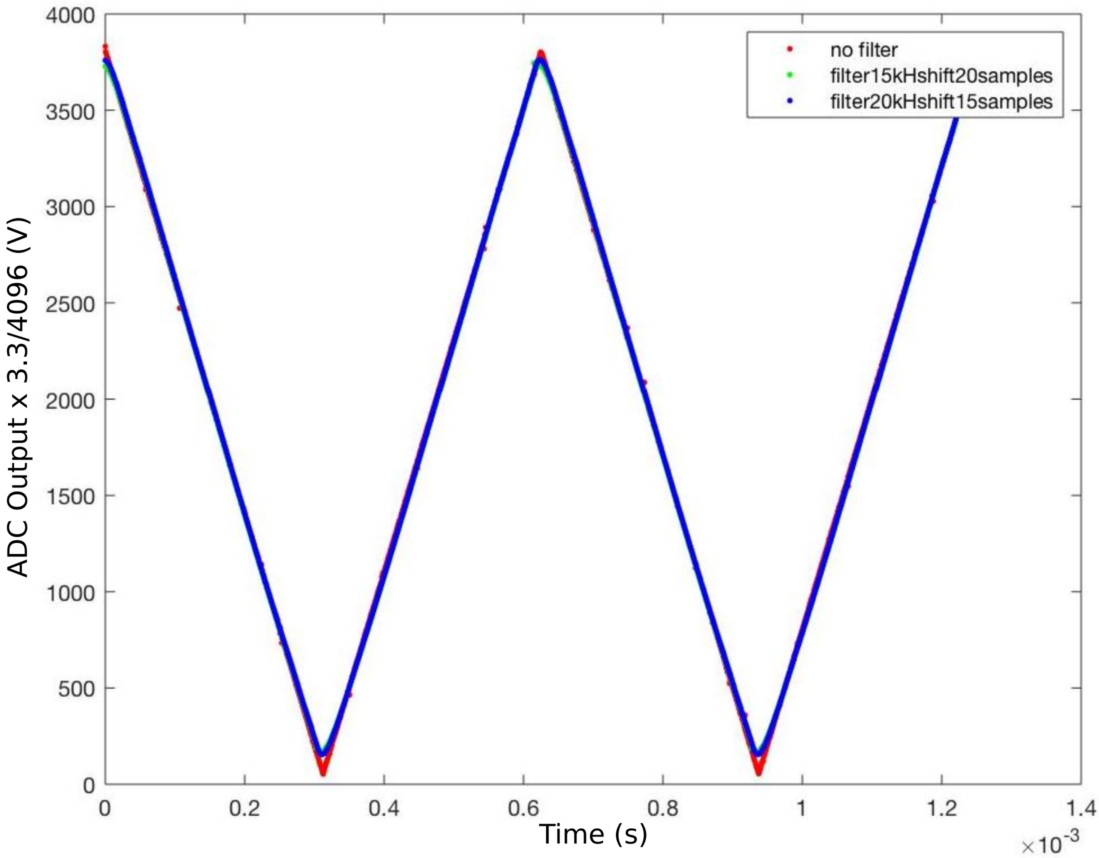

Figure 4.5. Triangular waves shifted after analog filters to realign phase. Red: original triangular; Green: triangular after 15 kHz filter and shifted by 20 samples (10 us); Blue: triangular after 20 kHz filter and shifted by 15 samples (7.5 us)

I collect traces of capacitance values after the 15 kHz and 20 kHz filters, for 600 s, 10 s and 1 s, then I evaluate the standard deviations and I compare them with the same results collected without filters. The following tables report comparison:

Table4.2.Standard deviation comparison for averaging method:

	STD (fF)		
type	*600 s*	*10 s*	*1 s*
nofilter	4.3621	4.3333	4.7967
20 kHz	3.8005	3.6864	3.6699
15 kHz	3.6203	3.5311	3.3894

Table4.3.Standard deviation comparison for oversampling and decimation method:

	STD (fF)		
type	*600 s*	*10 s*	*1 s*
nofilter	3.8692	3.6669	4.1307
20 kHz	3.0496	2.8043	2.6633
15 kHz	2.8551	2.8618	2.6876

It is evident that the standard deviation associated to filters is always smaller than the one associated to no filter, and the same observation can be retrieved from comparison between average and oversampling & decimation method. To obtain the same standard deviation results of oversampling with averaging I have to use more complex anti-aliasing filter (for example a second order low-pass filter). I can conclude that the benefits of an oversampling ADC include:

- Helping to design a simple analog anti-aliasing filter before ADC.
- Reducing the ADC noise floor with possible noise shaping so that a low-resolution ADC can be used.

Anti-aliasing filter with cut off frequency of 15 kHz is more powerful because it gives a higher magnitude reduction at 2 MHz (the sampling frequency), and so it can further reduce noise than anti-aliasing filter with cut off frequency of 20 kHz.

4.2.2 Operational amplifier characterization

At this point I carry out a deeper analysis on non-linearities of operational amplifier (OPAMP). The test aims at:

- understanding if non-linearities come from OPAMP
- understanding if errors are symmetric

The fist step of the test is based on applying a triangular wave directly to the input of analog to digital converter and sampling it, while applying a square wave to pin used to trigger timer; the second step is based on applying a triangular wave to the input of a voltage divider of $\frac{1}{2}$ followed by an amplification of 2 and sampling the output of OPAMP; the third and last step is based on applying a triangular wave to the input of the amplifier with amplification of 1.

As shown in Figure 4.6 I realize a voltage divider of $\frac{1}{2}$ by using R1=1 kOhm and R2=1 kOhm since

$$V_{out} = \frac{R2}{R1 + R2} V_{in} \tag{4.2}$$

followed by an amplification of 2 that I implement by using R3=1 kOhm and R4=1 kOhm since

$$Av = 1 + \frac{R2}{R1} \tag{4.3}$$

.

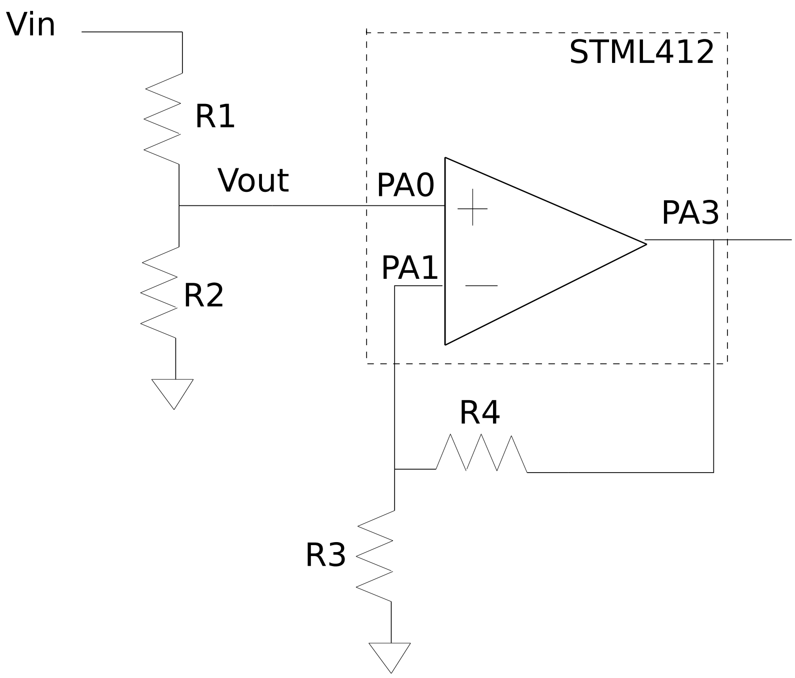

Figure 4.6. Schematic of voltage divider plus operational amplifier with amplification 2

In this case the OPAMP is programmed as STANDALONE and not as FOLLOWER, the non inverting input is at pin PA0, the inverting input is PA1 and the output is PA3. I change also the ADC input channel since the PA1 is mapped as non inverting input of the OPAMP. The new ADC channel is associated to pin PA4.

I compare the triangular wave directly sampled by ADC with triangular wave after the voltage divider and OPAMP amplification. I show them in figure 4.7 where the Y axis represents ADC Output and the X represents time in seconds. I also plot the absolute error between the two (Figure 4.8); the Y axis represents absolute error whereas the X, time in seconds.

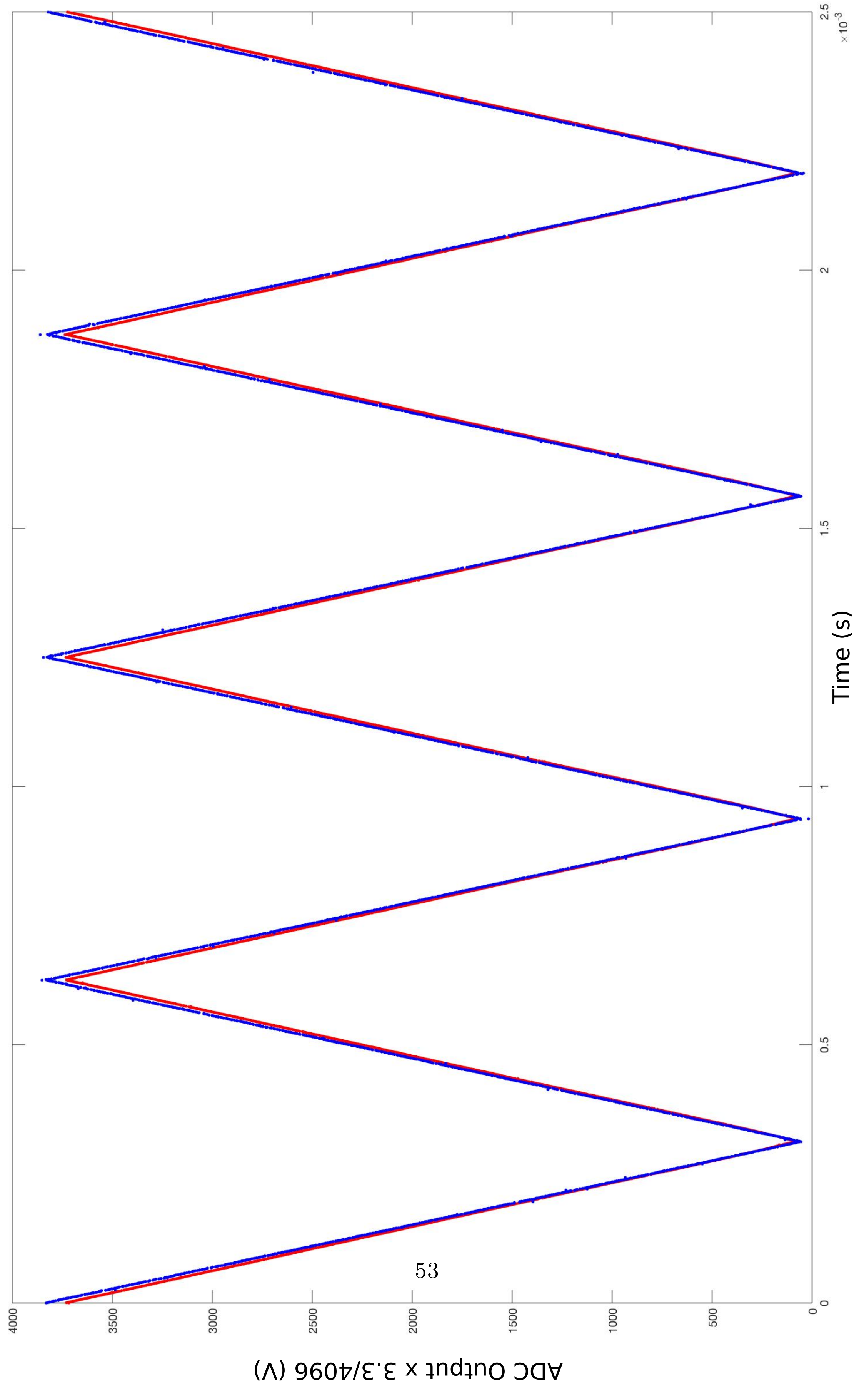

Figure 4.7. Original triangular wave (Blue) and triangular wave after voltage divider and OPAMP (Red) to evaluate the non-linearity errors

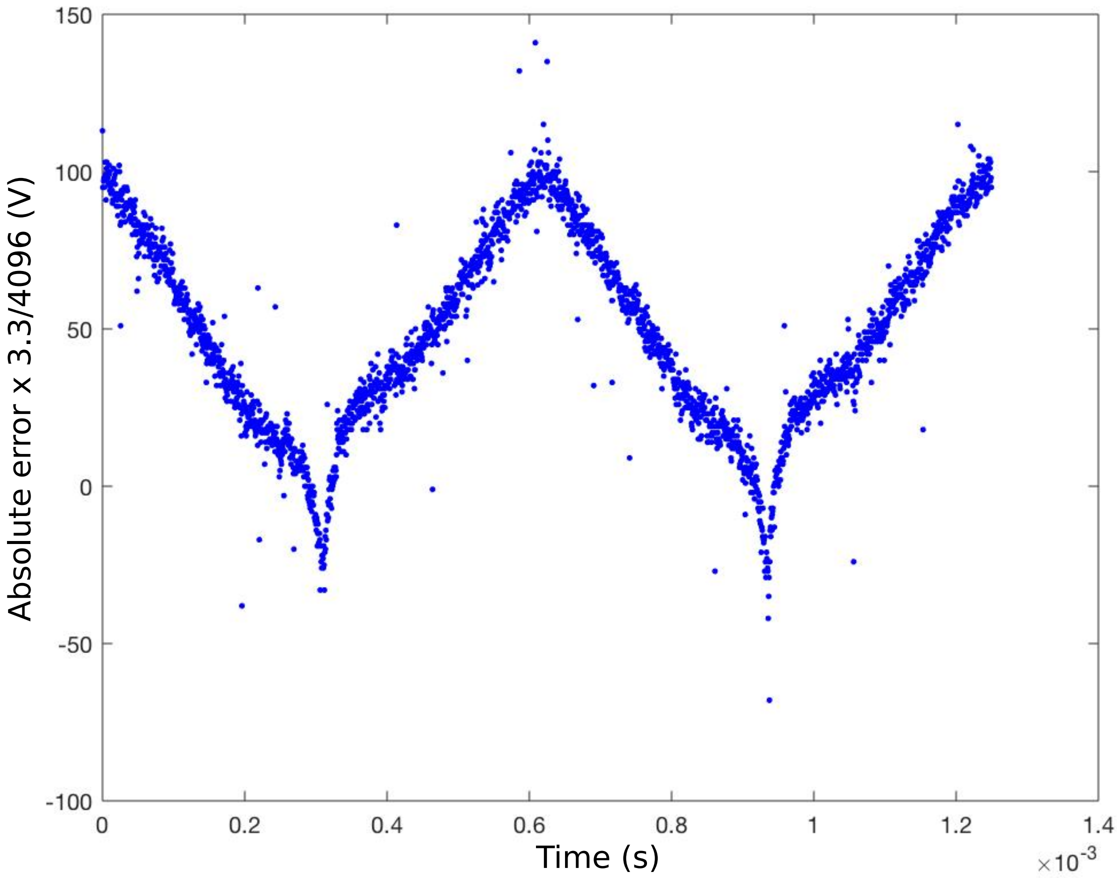

Figure 4.8. Absolute error between the original triangular wave and the triangular wave after the voltage divider and the OPAMP

From the plots I observe that the red triangular waveform does not follow the same trend of the ideal waveform (blue), the amplification is reduced in the upper corners, the slope is distorted and there are also vertical and horizontal misalignments. However, direct acquisition are more subject to high frequency noise than acquisition made through OPAMP.

Now, I want to check if distortions depend on gain OPAMP and how they affect the wave with respect to the amplification. I realize an amplification (Av) of 1 by implementing a voltage follower with direct line between inverting input and output of the OPAMP and I apply the triangular wave to no inverting input. The output of the OPAMP is then sampled by ADC. I compare it with the triangular wave directly sampled by ADC (Figure 4.9 and Figure 4.10) and the triangular wave sampled after voltage divider and the OPAMP (Figure 4.11 and Figure 4.12). All the plots have the X axis representing time in seconds. Plots showing triangular waves have the Y axis representing ADC Output, while plots showing differences have the Y axis representing absolute error.

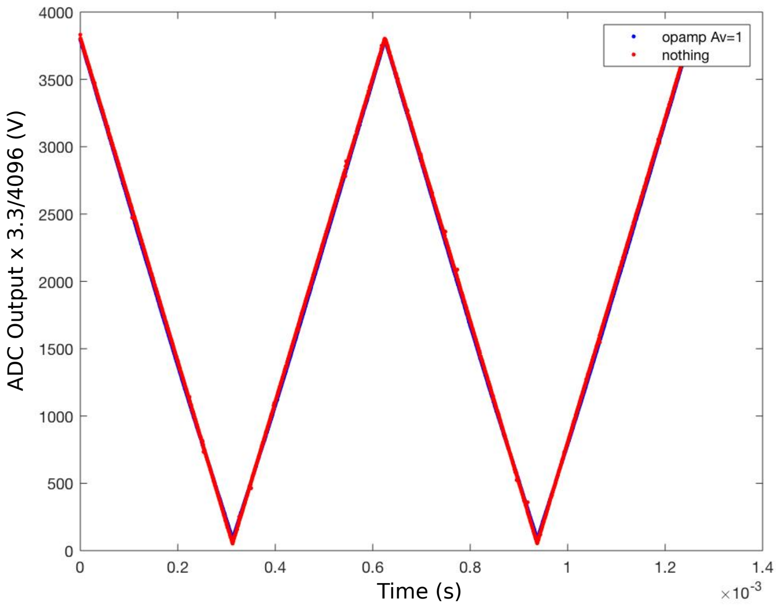

Figure 4.9. Original triangular wave (Red) and triangular wave after OPAMP with amplification (Av) 1 (Blue)

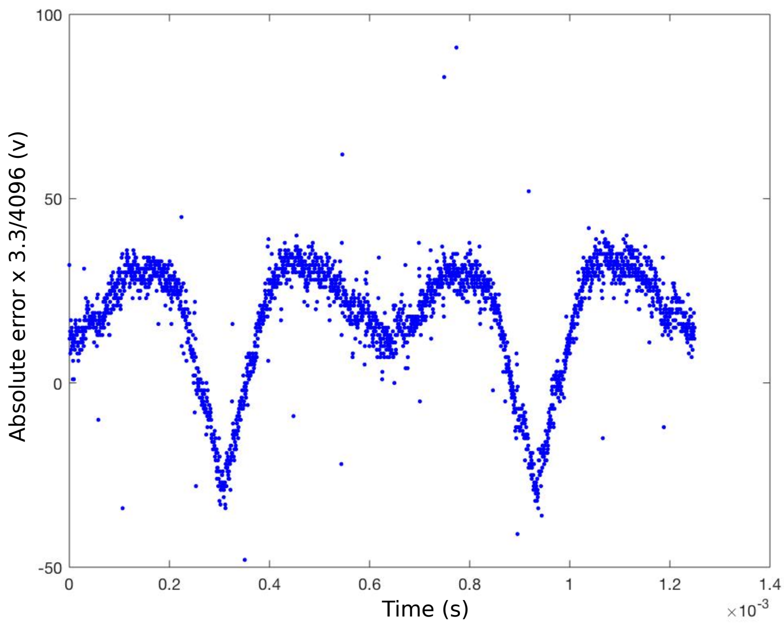

Figure 4.10. Absolute difference between original triangular wave and triangular wave after OPAMP with amplification (Av) 1

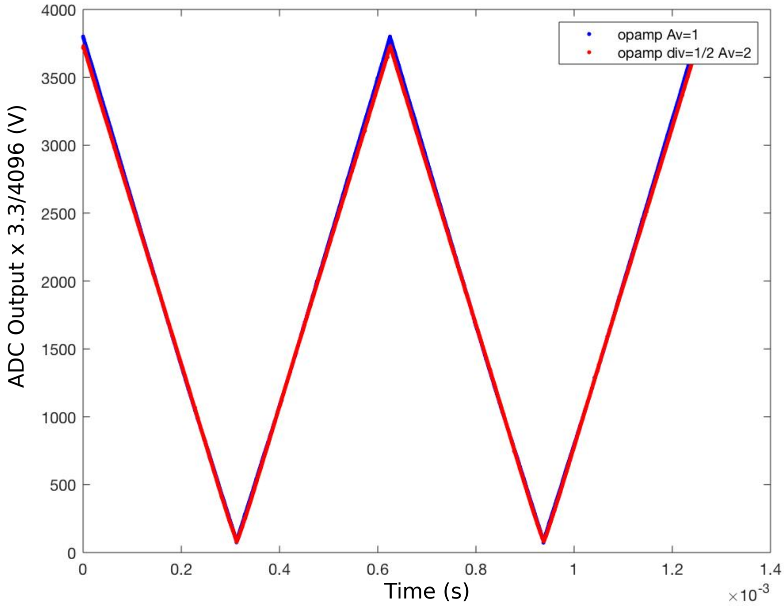

Figure 4.11. Triangular wave after OPAMP with amplification (Av) 1 (Blue) and triangular wave after voltage divider and OPAMP with amplification (Av) 2 (Red)

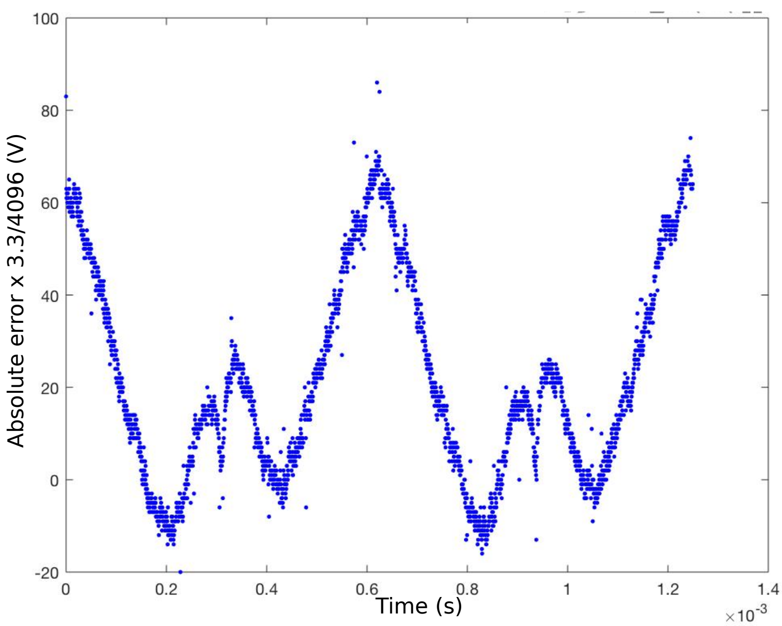

Figure 4.12. Absolute error between triangular wave after OPAMP with amplification 1 and triangular wave after voltage divider and OPAMP with amplification 2

The analysis confirms that the OPAMP can filter high frequency noise. The difference between the original triangular wave and the triangular wave after the OPAMP with

Av=1 (Figure 4.10) shows some symmetric non-linearities of the amplifier, which seem to change with the amplification factor.

4.3 Experiments related to digital filters

Digital filters reduce the noise outside the band of interest. Thus I am interested on studying the frequency spectrum of capacitance value traces.

Explanations about how I design filters and the shape of their masks are in Chapter 3, here I present tests used to control the behavior of filters.

4.3.1 Noise reduction outside the frequency band of interest

This last test is composed of collecting capacitance values for short slot of time, applying digital filters and then moving on frequency domain. Test aims at analyzing the frequency spectrum (FFT) of capacitance timing variations for different filtering levels:

- analog antialias filter 20 kHz, digital filter 1 Hz
- analog antialias filter 20 kHz, digital filter 3 Hz
- analog antialias filter 15 kHz, digital filter 1 Hz
- analog antialias filter 15 kHz, digital filter 3 Hz

I apply a square wave between 0 V and 3 V with a frequency of 1.6 kHz to PIN PA5 (trigger source for TIM2) of the board, then I apply a triangular wave between 0 V and 3 V with frequency 1.6 kHz to PIN PA1 (input of ADC)(see figure 3.11). As the frequency resolution depends on the length of acquisition, I collect a trace of about 5 min of capacitance values. Then I apply digital filters and I compare Fast Fourier Transform (FFT) before and after filtering step. The MATLAB code used to design and apply filters is available in the Appendix A. In Appendix A.4 can be observed that I eliminate the minimum value from all the data before entering the processing chain to eliminate the DC component of the FFT to visualize better the noise.

First, I use digital filters with cutoff frequency of 1 Hz. In the following I report plots that display FFT before digital filters, FFT after FIR filter, FFT after Chebyshev filter and FFT after Elliptic filter (Figure 4.13, Figure 4.14, Figure 4.15 and Figure 4.16). The X axis represents frequency in Hz, the Y axis represents Magnitude in dB.

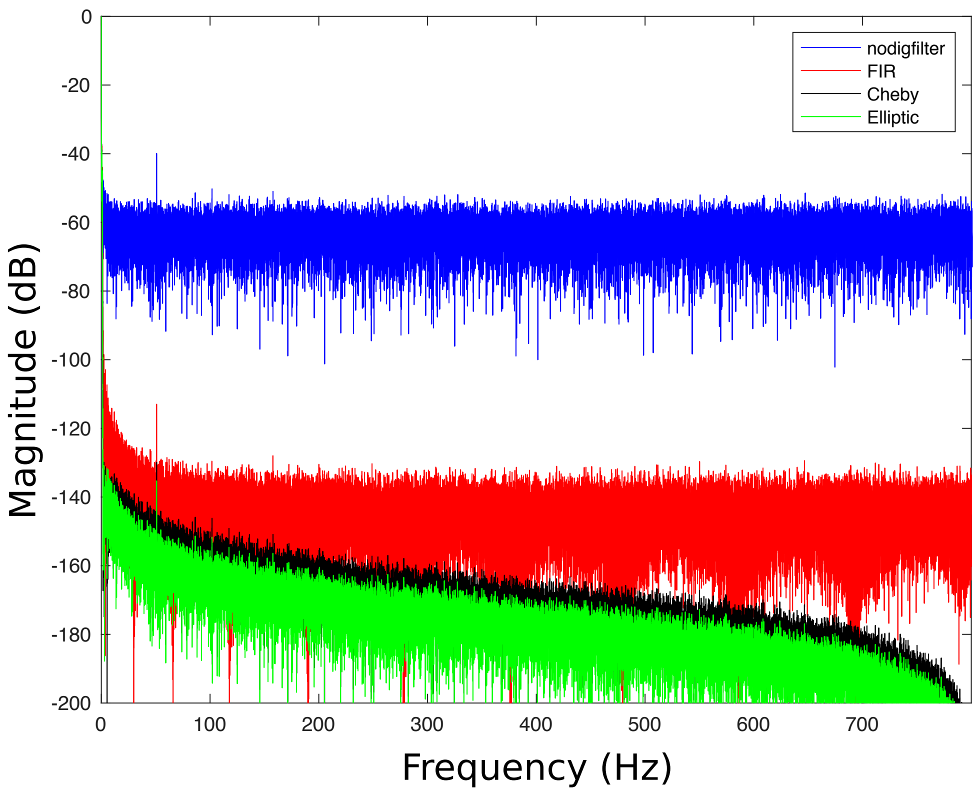

Figure 4.13. FFTs of Cplate values, calculated through averaging, with and without 1 Hz low-pass digital filters

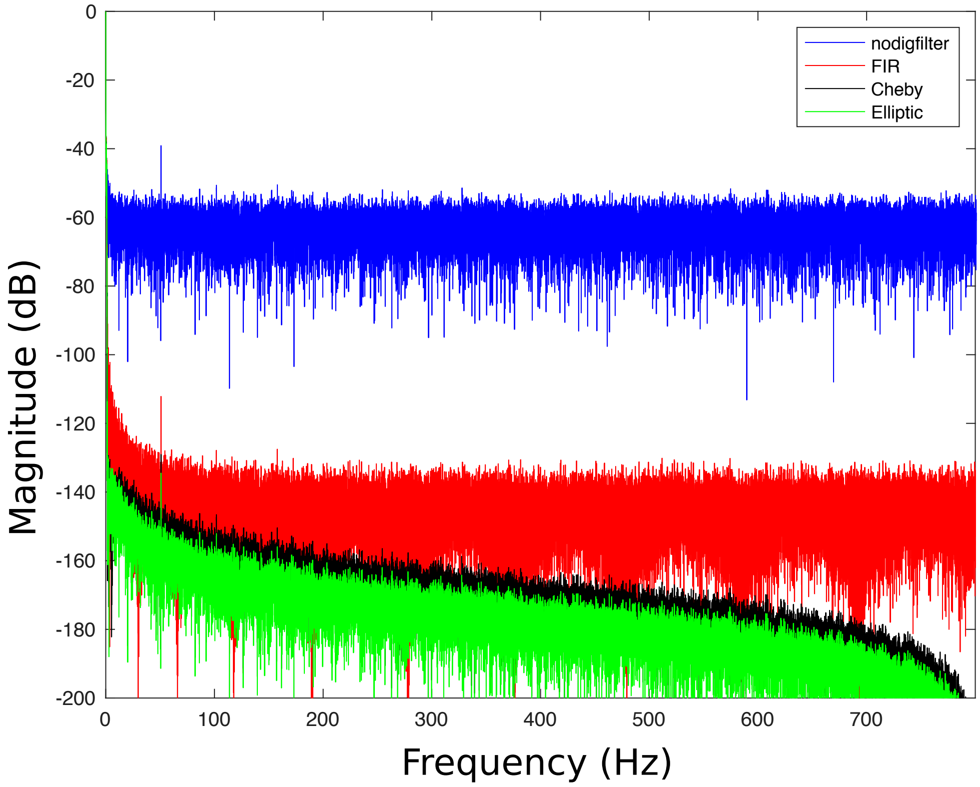

Figure 4.14. FFTs of Cplate values, calculated through oversampling and decimation, with and without 1 Hz low-pass digital filters

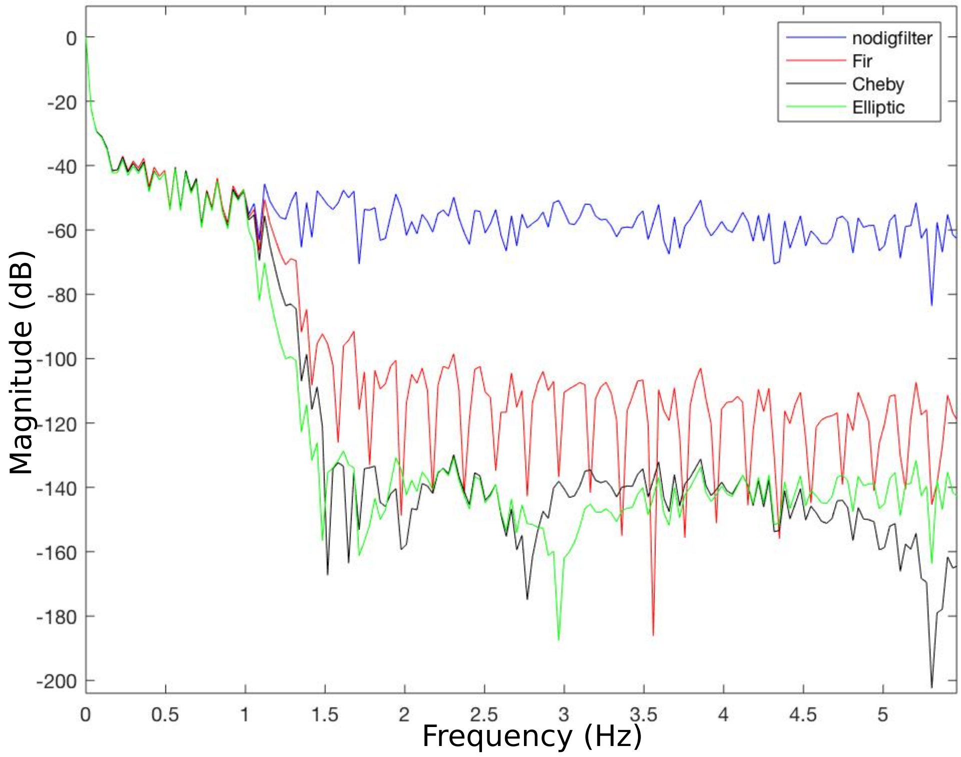

Figure 4.15. Detail of FFTs of Cplate values, calculated through averaging, with and without 1 Hz low-pass digital filters

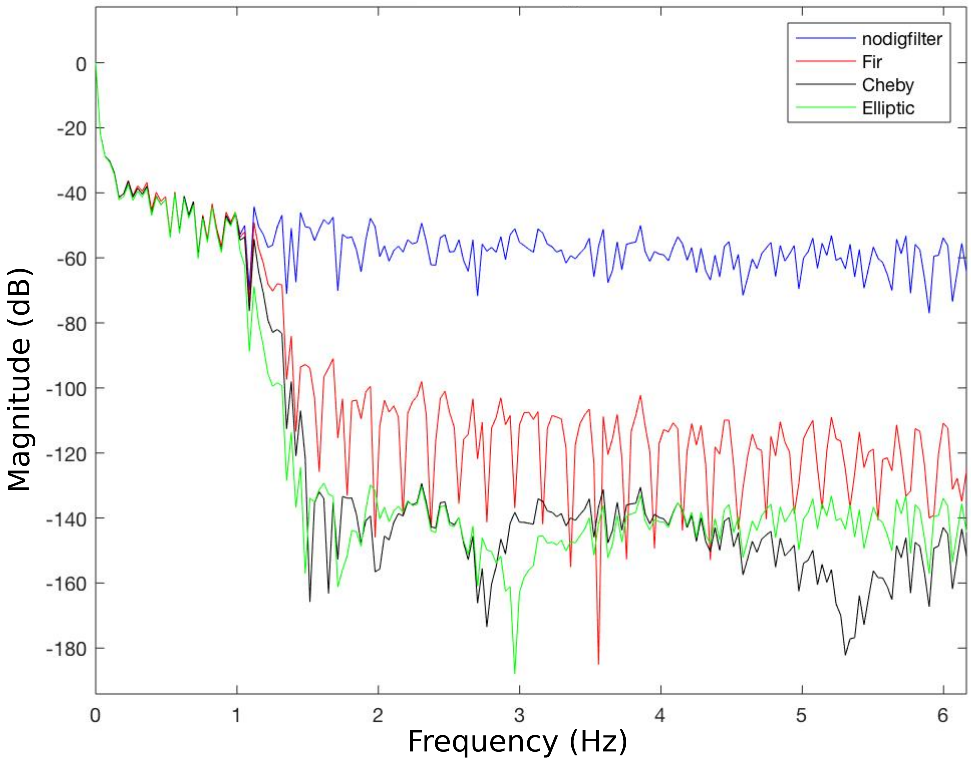

Figure 4.16. Detail of FFTs of Cplate values, calculated through oversampling and decimation, with and without 1 Hz low-pass digital filters

Outside the band of interest (0 Hz-3 Hz), the noise level is rather low, -40 dB or less, and it is further attenuated by the digital filters. The Elliptic filter (green line) starts attenuating earlier than the other filters, and stays below -130 dB after 1.5 Hz like Chebyshev (black line). The slope of the FIR (red line) is slower with lower attenuation (below -100 dB) and it has big ripples in the stopband.

There are no significant differences between averaging and oversampling & decimation methods (Figure 4.13, Figure 4.14), since the discretization noise is less than signal noise. There is a high peak noise (-40 dB) at 50 Hz in the FFT of the signal without filtering, which is reduced by -70 dB with the FIR filter, by -90 dB with the Chebyshev filter and by -95 dB with the Elliptic filter.

Next, I repeat the sequence by including also analog antialias filter of 20 kHz (Figure 4.17, Figure 4.18, Figure 4.19, Figure 4.20).

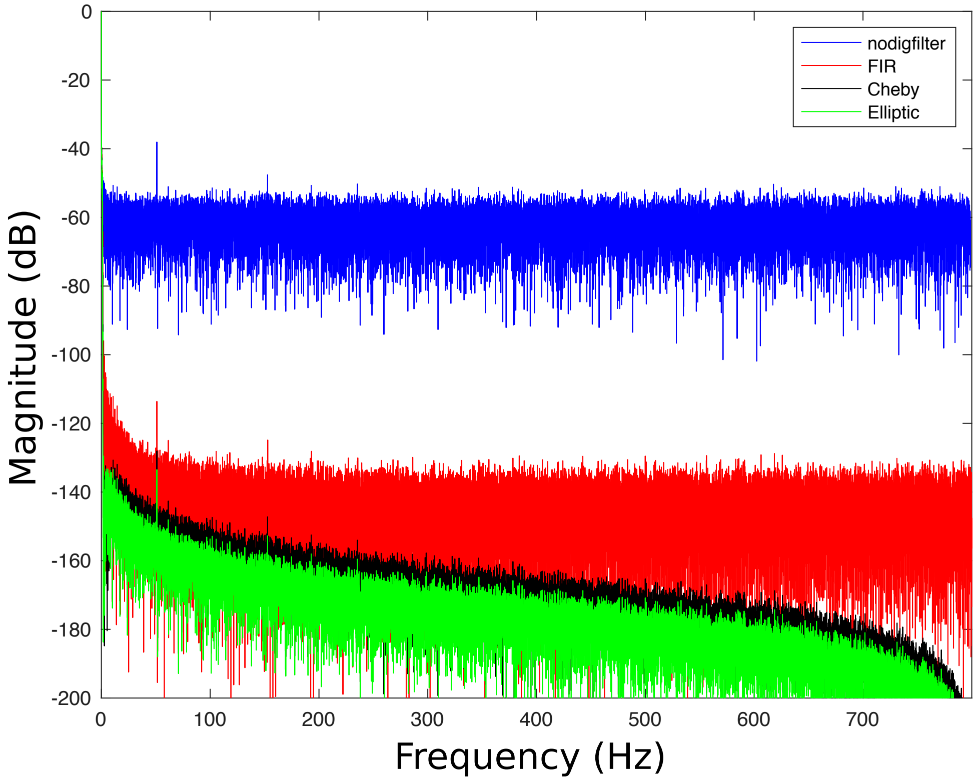

Figure 4.17. FFTs of Cplate values, calculated through averaging, before digital filters and after digital filters, including analog antialias filter with cut off frequency of 20 kHz

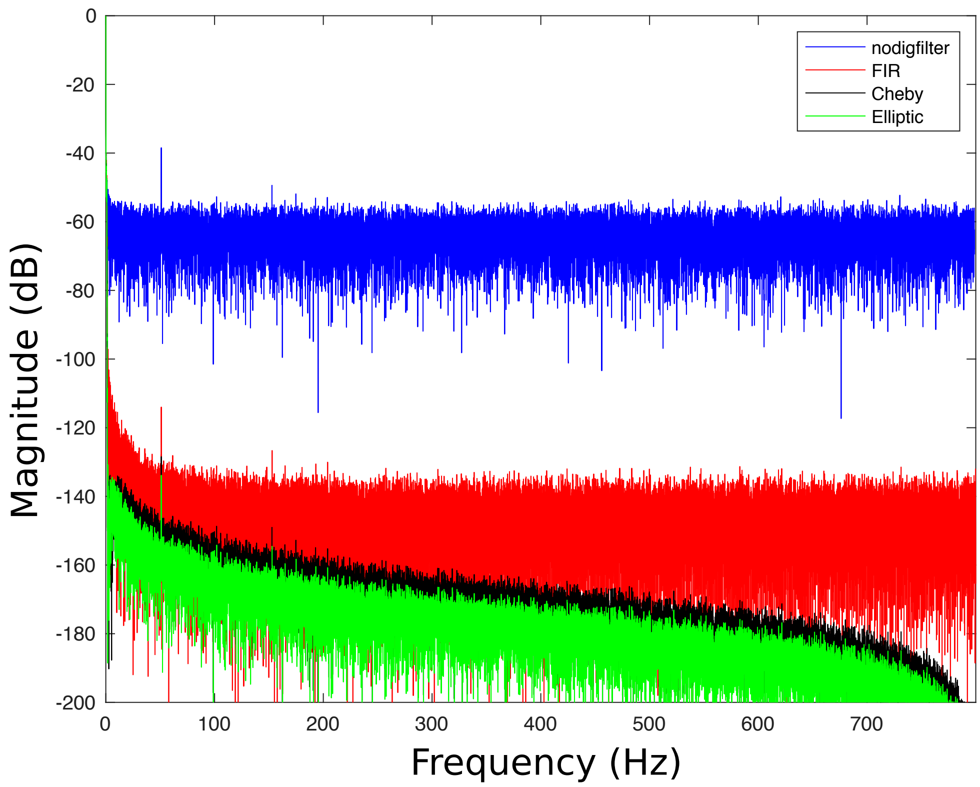

Figure 4.18. FFTs of Cplate values, calculated through oversampling and decimation, with and without 1 Hz low-pass digital filters, including analog antialias filter with cut off frequency of 20 kHz

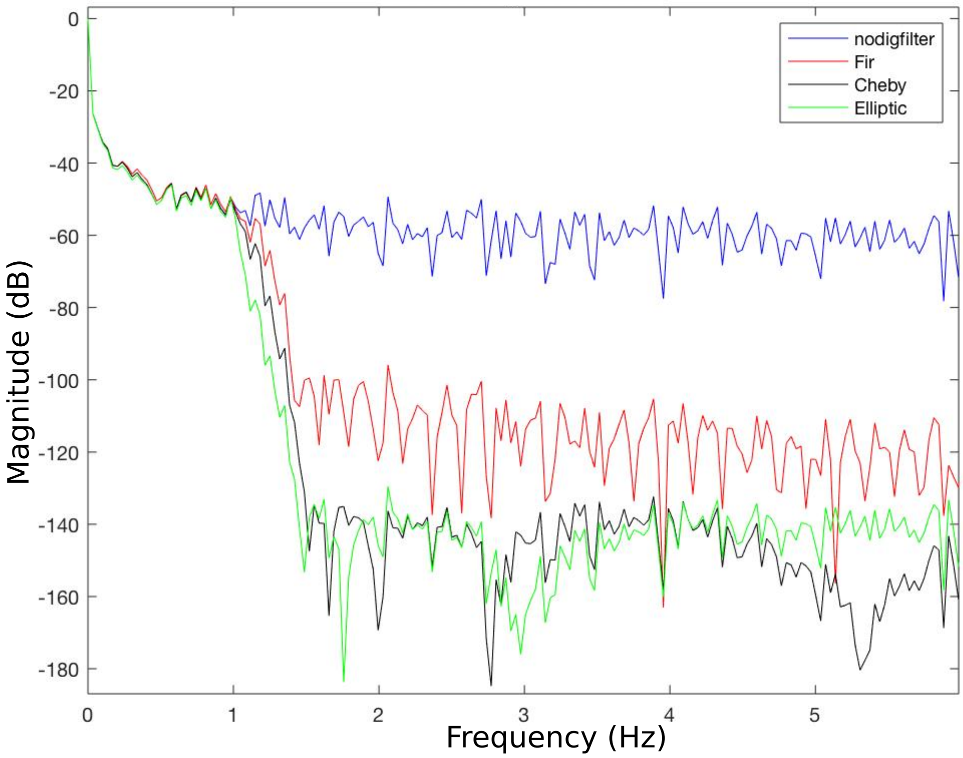

Figure 4.19. Detail of FFTs of Cplate values, calculated through averaging, with and without 1 Hz low-pass digital filters, including analog antialias filter with cut off frequency of 20 kHz

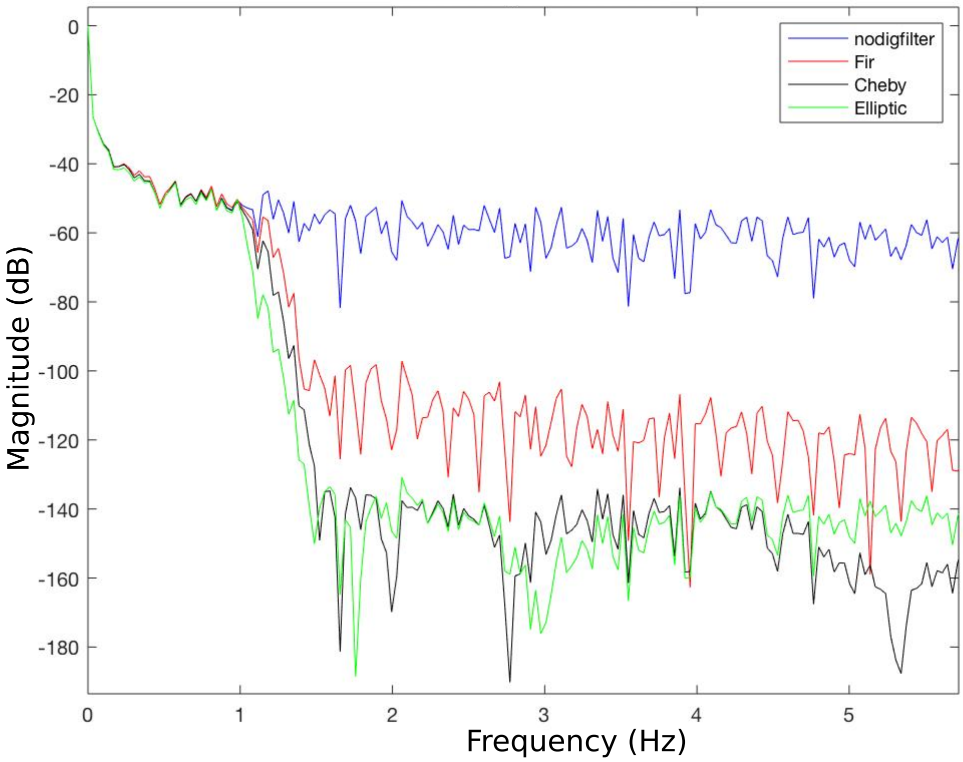

Figure 4.20. Detail of FFTs of Cplate values, calculated through oversampling and decimation, with and without 1 Hz low-pass digital filters, including analog antialias filter with cut off frequency of 20 kHz

Also in this set of results, the Elliptic filter presents the steepest roll-off rate, and the averaging and oversampling & decimation do not have significant differences. Figure 4.17 and Figure 4.18 show the same high peak noise (-40 dB) at 50 Hz in the FFT computed before filtering, and then attenuated respectively by the filters.

Lastly, I repeat the sequence using a 15 kHz analog antialias filter.

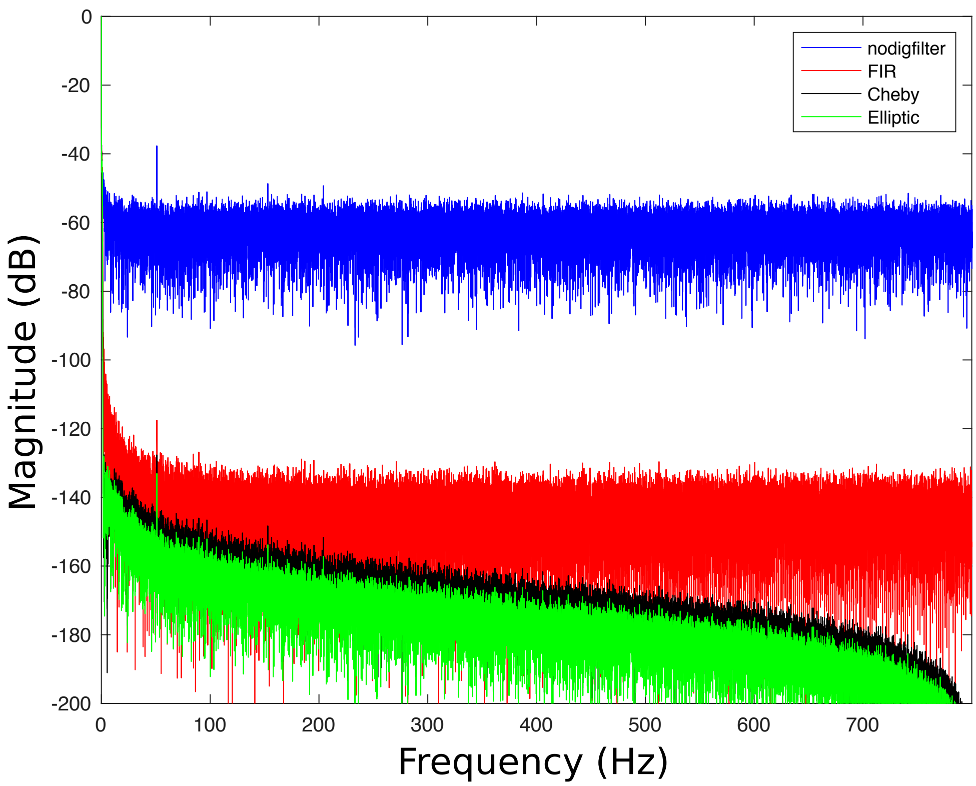

Figure 4.21. FFTs of Cplate values, calculated through averaging, before digital filters and after digital filters, including analog antialias filter with cut off frequency of 15 kHz

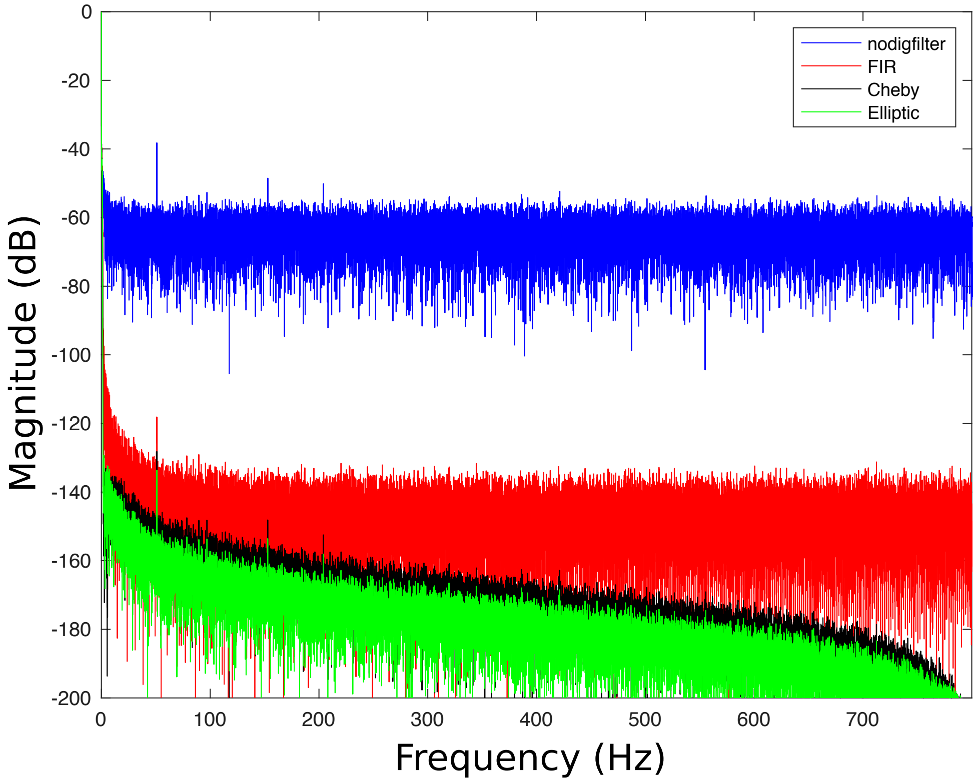

Figure 4.22. FFTs of Cplate values, calculated through oversampling and decimation, with and without 1 Hz low-pass digital filters, including analog antialias filter with cut off frequency of 15 kHz

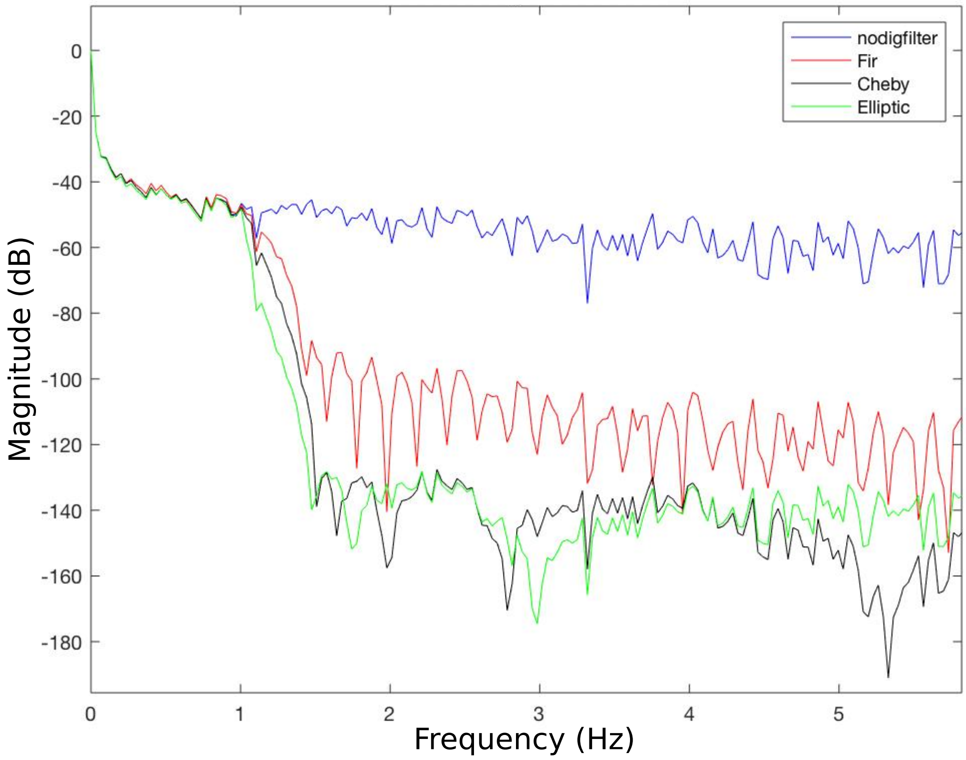

Figure 4.23. Detail of FFTs of Cplate values, calculated through averaging, with and without 1 Hz low-pass digital filters, including analog antialias filter with cut off frequency of 15 kHz

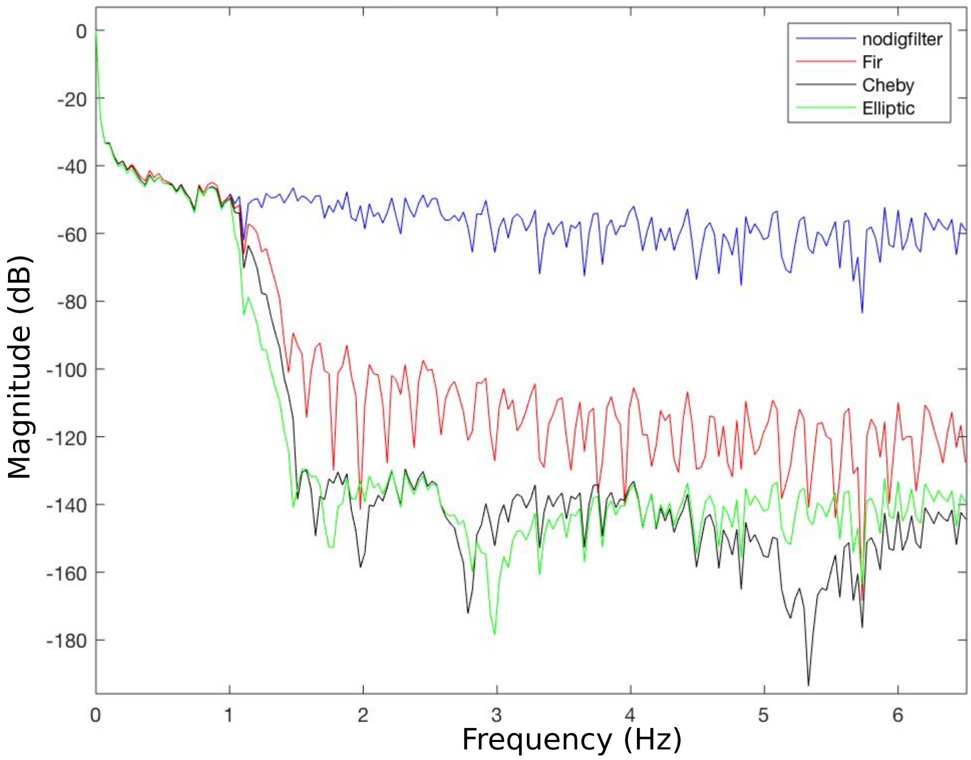

Figure 4.24. Detail of FFTs of Cplate values, calculated through oversampling and decimation, with and without 1 Hz low-pass digital filters, including analog filter with cut off frequency of 15 kHz

From the previous images it's clearly visible the different behaviors of the filters. Elliptic (green line) shows the fastest roll-off rate and it requires lower order (7), FIR (red line) has the poorest roll-off rate and it is computationally expensive while Chebyshev (black line) has a good performance even if it has slower roll-off rate than Elliptic and it requires higher order (11). Another important thing that I can state from detailed pictures is that lines between 0 Hz and 1.5 Hz are smoother and lower in the cases of analog filters, this means that noise in useful band is effectively reduced.

I replicate the analysis by using digital filters with cut off frequency of 3 Hz. Figures 4.25, Figure 4.26, Figure 4.27 and Figure 4.28 show results of test without analog antialias filters.

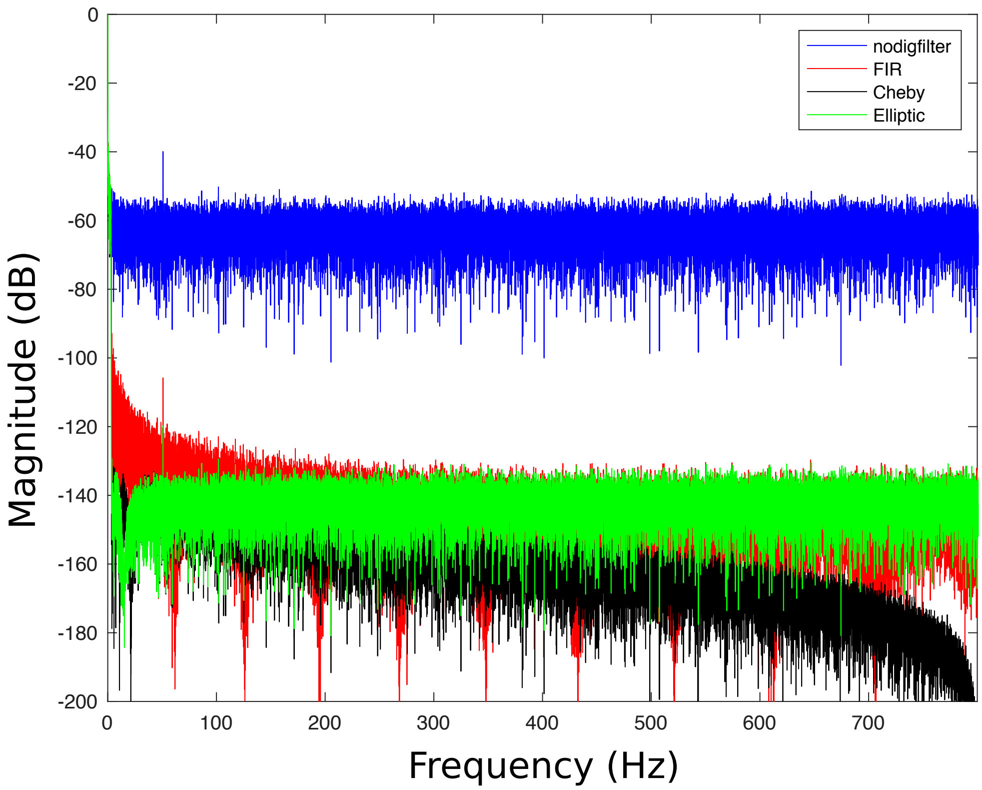

Figure 4.25. FFTs of Cplate values, calculated through averaging, with and without 3 Hz low-pass digital filters

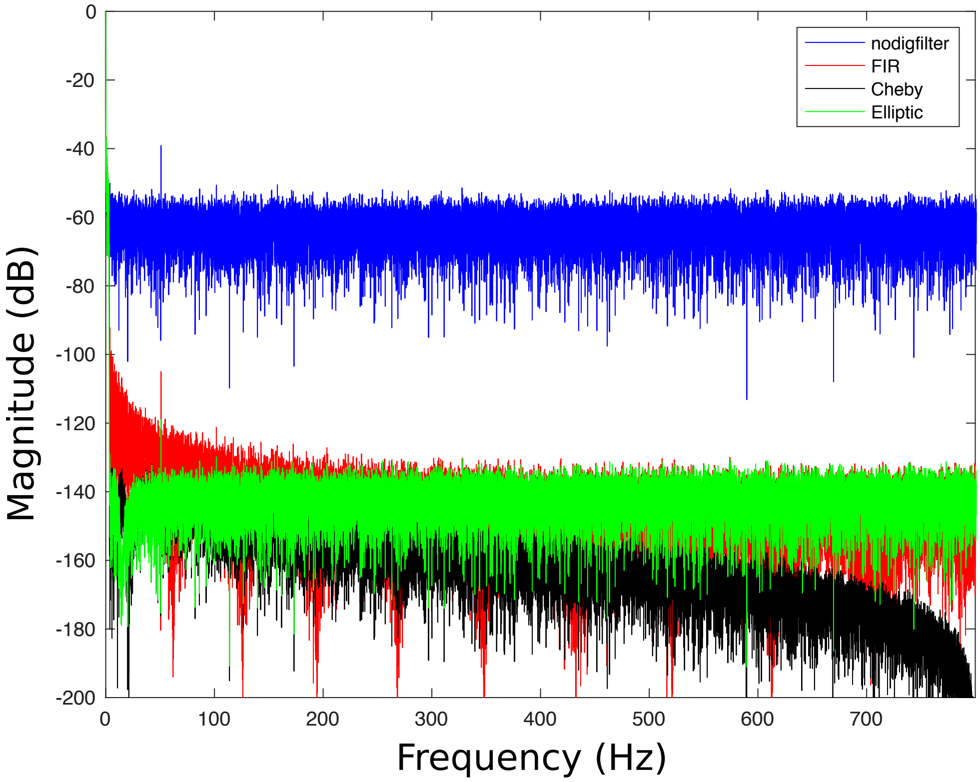

Figure 4.26. FFTs of Cplate values, calculated through oversampling and decimation, with and without 3 Hz low-pass digital filters

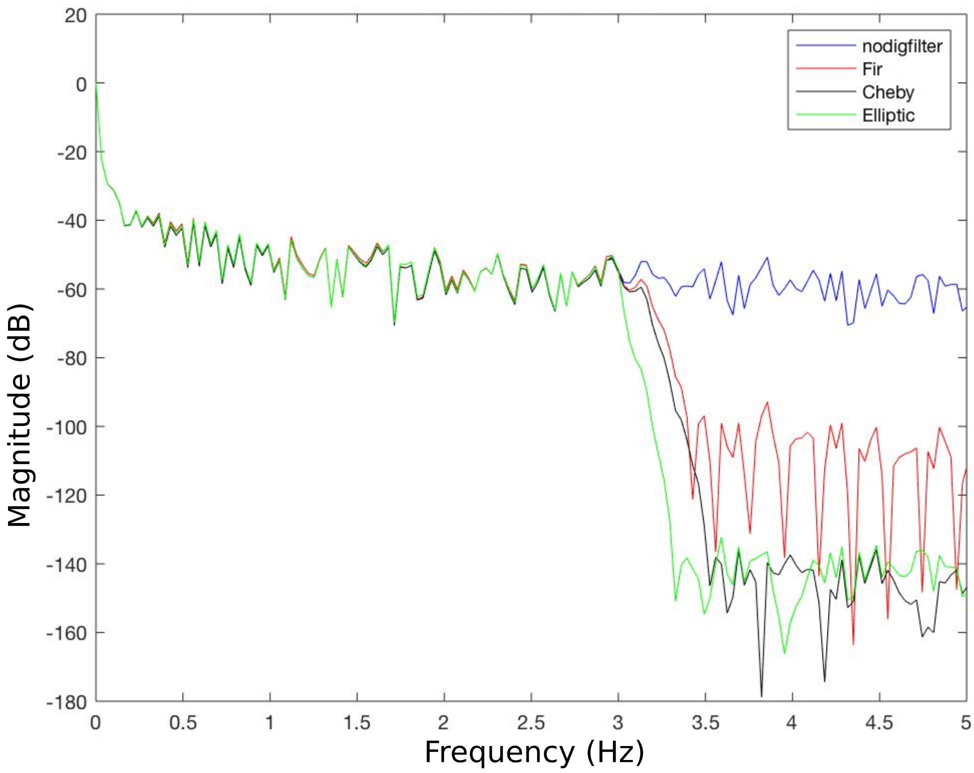

Figure 4.27. Detail of FFTs of Cplate values, calculated through averaging, with and without 3 Hz low-pass digital filters

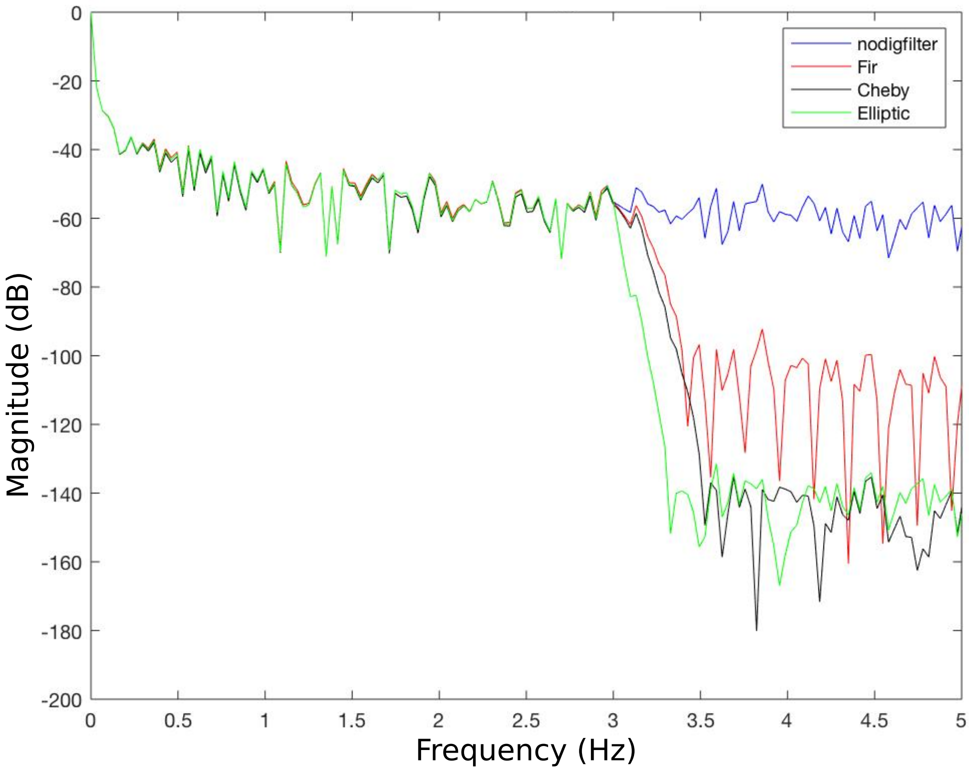

Figure 4.28. Detail of FFTs of Cplate values, calculated through oversampling and decimation, with and without 3 Hz low-pass digital filters

Elliptic starts attenuating before the others. FIR still has big ripples in stopband and it achieves a reduction of -140 dB at 300 Hz. In addition Elliptic at high frequency stuck at -140 dB while Chebyshev goes even lower. There is, as in the previous results, a high peak noise at 50 Hz, which is then reduced by -65 dB with the FIR filter, by -80 dB with the Chebyshev and Elliptic filter. There are not significant differences between averaging and oversampling and decimation.

Figures 4.29, Figure 4.30, Figure 4.31, and Figure 4.32 show results of test including analog antialias filter with cut off frequency of 20 kHz.

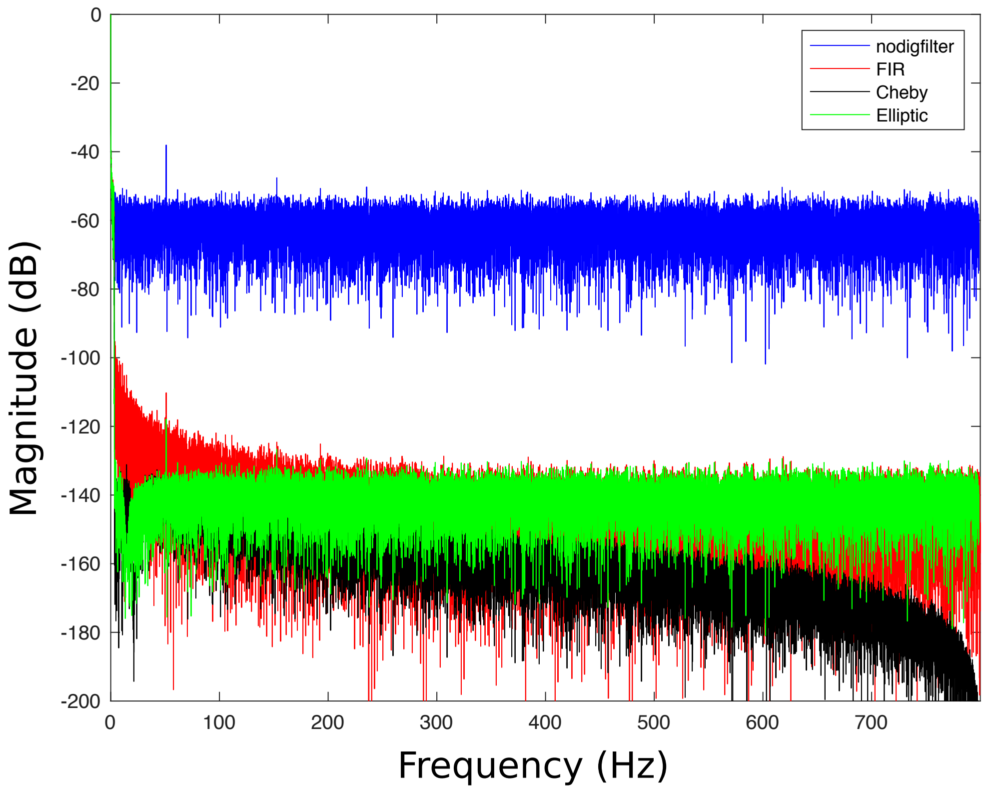

Figure 4.29. FFTs of Cplate values, calculated through averaging, with and without 3 Hz low-pass digital filters, including analog antialias filter with cut off frequency of 20 kHz

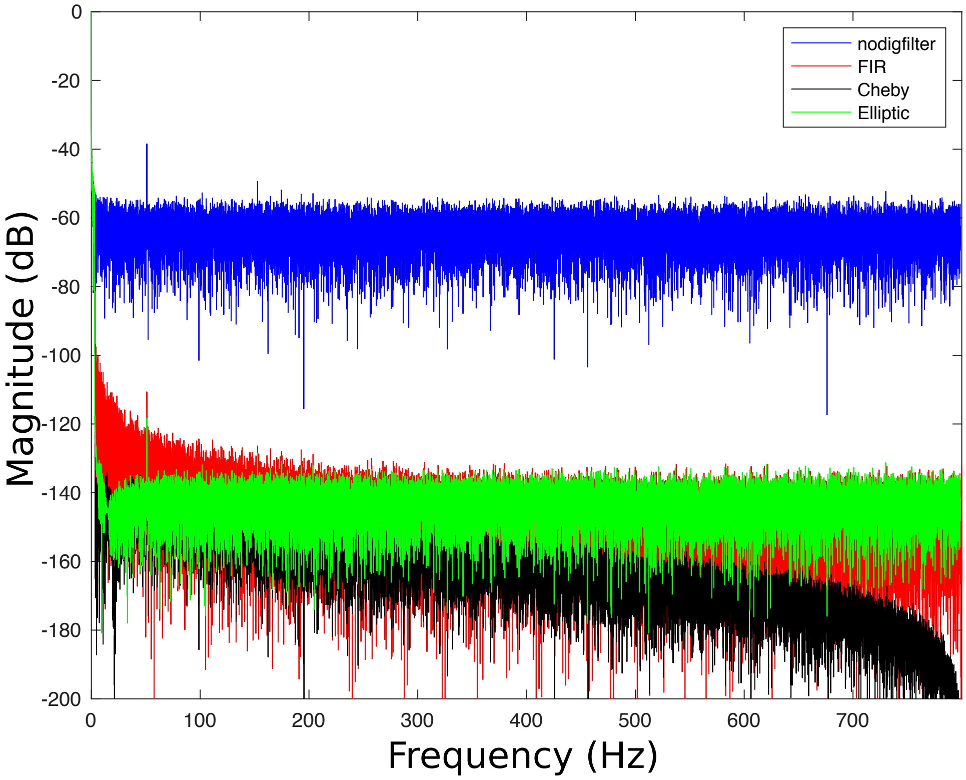

Figure 4.30. FFTs of Cplate values, calculated through oversampling and decimation, with and without 3 Hz low-pass digital filters, including analog antialias filter with cut off frequency of 20 kHz

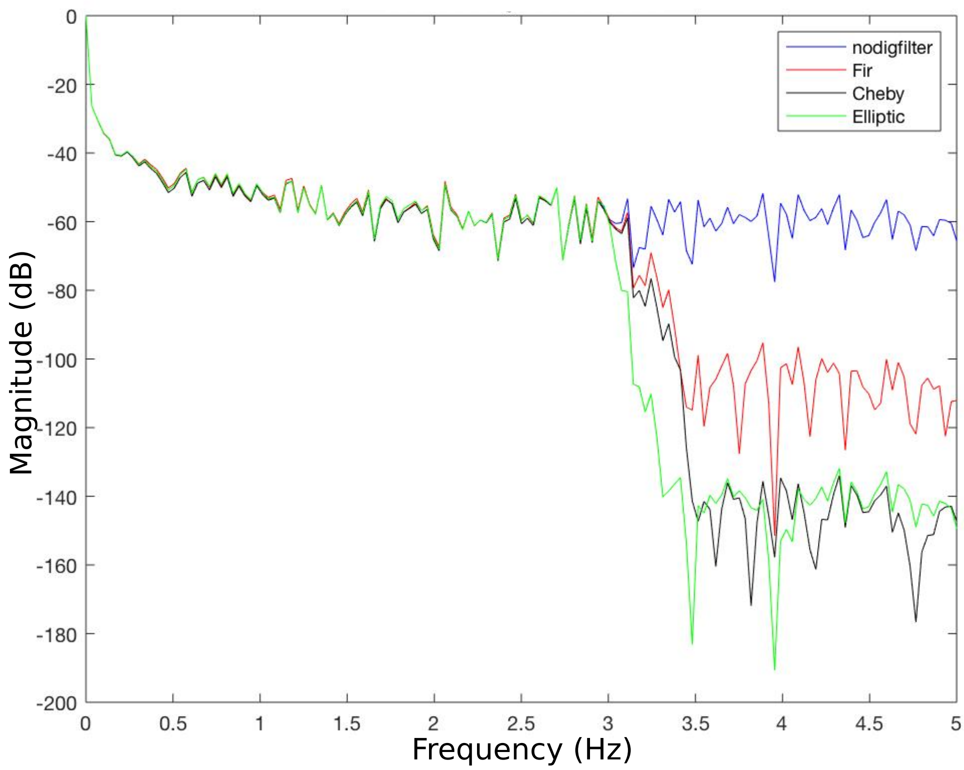

Figure 4.31. Detail of FFTs of Cplate values, calculated through averaging, with and without 3 Hz low-pass digital filters, including analog antialias filter with cut off frequency of 20 kHz

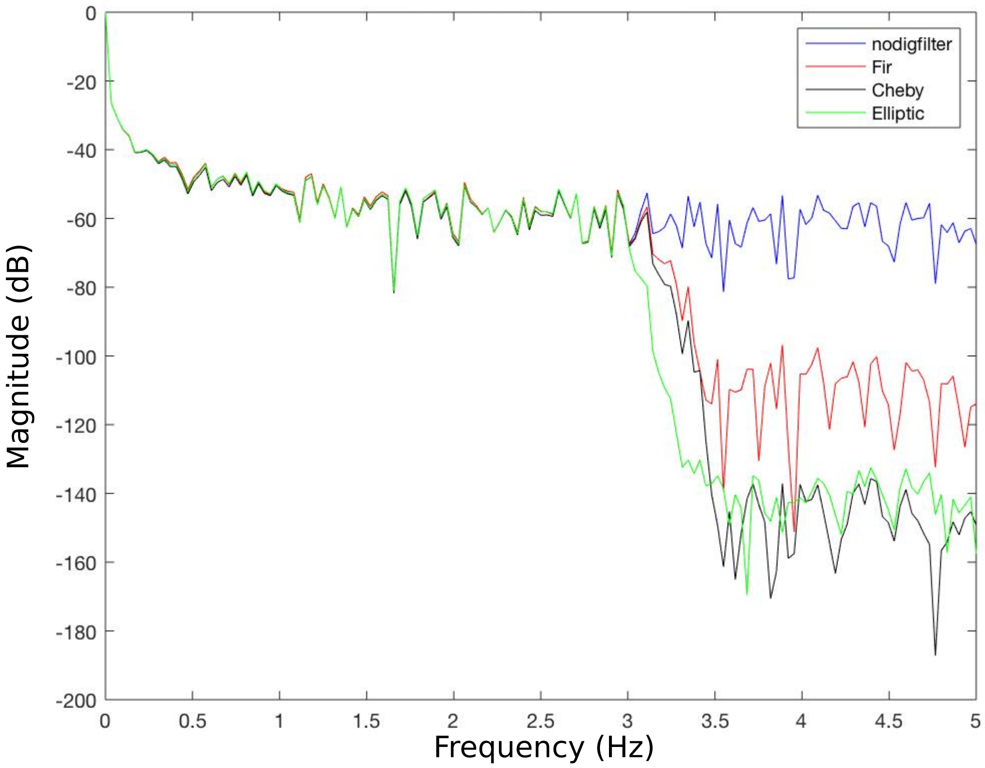

Figure 4.32. Detail of FFTs of Cplate values, calculated through oversampling and decimation, with and without 3 Hz low-pass digital filters, including analog antialias filter with cut off frequency of 20 kHz

Elliptic has a faster roll-off rste than FIR and Chebyshev, however at high frequencies it has the same behavior argued in the previous set of the results. The line between 0 Hz and 3 Hz is smoother and lower than the case without analog filter.

Figures 4.33, Figures 4.34, Figures 4.35, and Figures 4.36 show results of test including analog filter with cut off frequency of 15 kHz.

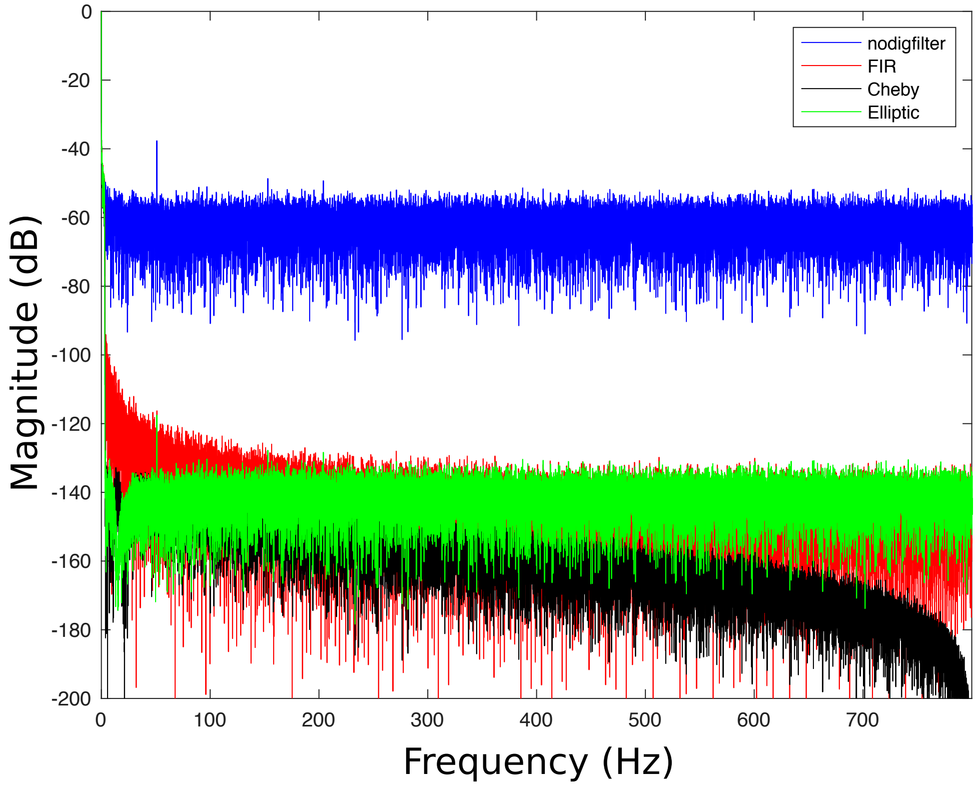

Figure 4.33. FFTs of Cplate values, calculated through averaging, with and without 3 Hz low-pass digital filters, including analog antialias filter with cut off frequency of 15 kHz

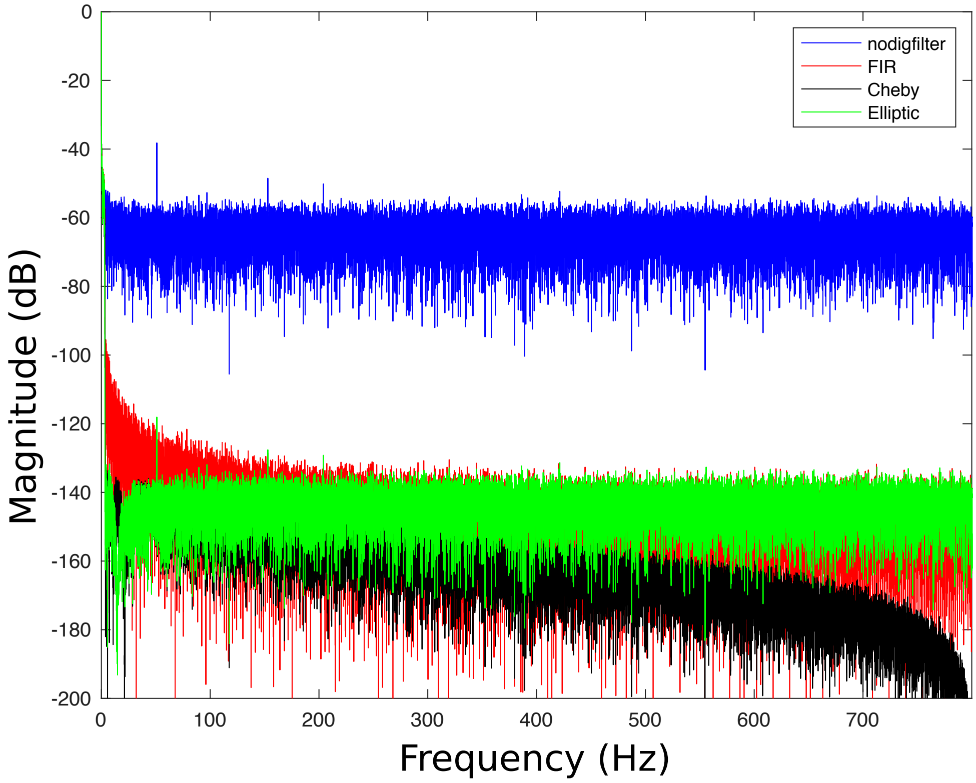

Figure 4.34. FFTs of Cplate values, calculated through oversampling and decimation, with and without 3 Hz low-pass digital filters, including analog antialias filter with cut off frequency of 15 kHz

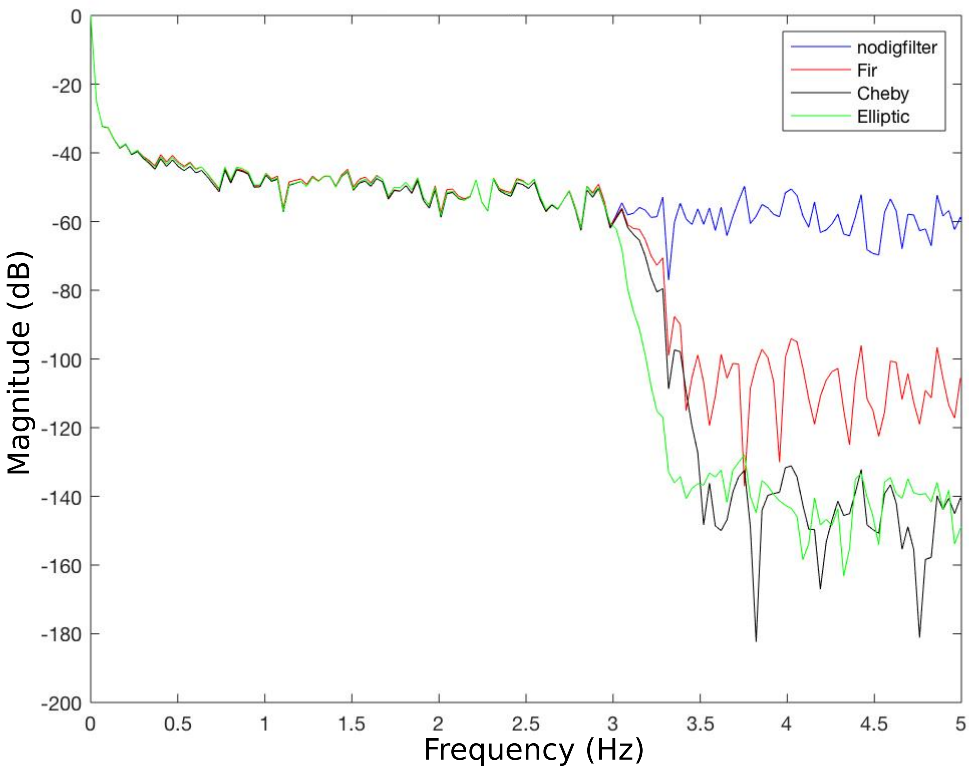

Figure 4.35. Detail of FFTs of Cplate values, calculated through averaging, with and without 3 Hz low-pass digital filters, including analog antialias filter with cut off frequency of 15 kHz

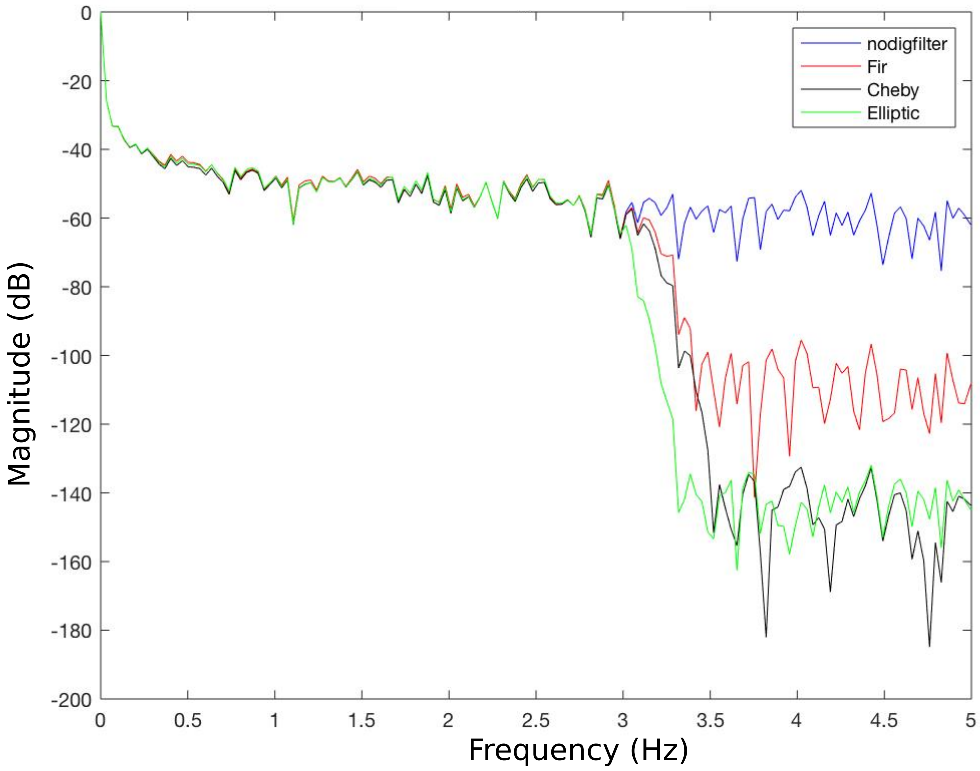

Figure 4.36. Detail of FFTs of Cplate values, calculated through oversampling and decimation, with and without 3 Hz low-pass digital filters, including analog antialias filter with cut off frequency of 15 kHz

From this set of results, I can arise the same conclusions explained before. I highlight again the fact that the line between 0 Hz and 3 Hz is even smoother and lower than the case with analog antialias filter with cut off frequency of 20 kHz. Figure 4.33 and Figure 4.34 shows the high peak noise at 50 Hz, which is reduced by -90 dB with FIR and by -80 dB with Chebyshev and Elliptic.

Chapter 5

Conclusion

This research focuses on implementing a new front end for capacitive sensors, based on single plate differential measurements.
The implementation proposed and analyzed is composed of an analog antialiasing low-pass filter in order to reduce high frequency noise, an Analog to Digital Converter and a microprocessor to implement a processing method for measuring capacitance that significantly reduces extern environmental noise, and a digital filter to reduce noise outside the frequency bands of interest.
The new processing method uses differential single plate measurements, and it provides:

- Cancellation of measurement drift of electric capacitance of single-plate sensors due to environmental sources.
- Reduction of measurement jitter because the measurement of the electrical capacitance of sensor plate does not rely on voltage thresholds.
- Use of oversampling techniques to lower the performance requirements for analog-to-digital convertor (ADC) and signal filters.
- Differential capacitance measurement can be used on most capacitive sensors without requiring physical changes.

From the tests performed and the results collected I find that ADC with oversampling and antialiasing filter with cut off frequency of 15 kHz, which is ten times higher than signal frequency, give the best high frequency noise reduction. Instead, regarding digital filters I conclude that Elliptic gives the best result with the less effort.

This research improves long-range traceability for indoor safety and security applications, such as detection of persons or wares, indoor monitoring of person position, movements and behavior.

The new front-end has to be tested with the real analog interface composed of one single plate which is charged/discharged by a constant current driven by a square wave generated through a signal generator. Then the stability of the analog interface has to be checked.

An alternative implementation can be based on analog circuits to process Vc as follows:

- analog differentiator circuits calculate the slopes of the ramps,S_L and S_R
- an analog inverter to calculate the modulus of the negative slope;
- an analog average circuit (low-pass filter) to calculate the average of the modulus of the two slopes, which reduces also high-pitch noise present in VC.

Other implementations can use combinations of analog and digital processing for the various operations mentioned above.

References

[1] Ramezani Akhmareh, Alireza, Mihai Teodor Lazarescu, Osama Bin Tariq, and Luciano Lavagno. "A tagless indoor localization system based on capacitive sensing technology." Sensors 16, no. 9 (2016): 1448.

[2] Iqbal, Javed, Mihai Teodor Lazarescu, Osama Bin Tariq, and Luciano Lavagno. "Long range, high sensitivity, low noise capacitive sensor for tagless indoor human localization." In Advances in Sensors and Interfaces (IWASI), 2017 7th IEEE International Workshop on, pp. 189-194. IEEE, 2017.

[3] Tariq, Osama Bin, Mihai Teodor Lazarescu, Javed Iqbal, and Luciano Lavagno. "Performance of Machine Learning Classifiers for Indoor Person Localization With Capacitive Sensors." IEEE Access 5 (2017): 12913-12926.

[4] Iqbal, Javed, Arslan Arif, Osama Bin Tariq, Mihai Teodor Lazarescu, and Luciano Lavagno. "A contactless sensor for human body identification using RF absorption signatures." In Sensors Applications Symposium (SAS), 2017 IEEE, pp. 1-6. IEEE, 2017.

[5] Iqbal, Javed, Mihai Teodor Lazarescu, Arslan Arif, and Luciano Lavagno. "High sensitivity, low noise front-end for long range capacitive sensors for tagless indoor human localization." In Research and Technologies for Society and Industry (RTSI), 2017 IEEE 3rd International Forum on, pp. 1-6. IEEE, 2017.

[6] Mihai T. Lazarescu. "Drift Rejection Differential Frontend for Single Plate Capacitive Sensors".

[7] STMicroelectronics. "Ultra-low-power Arm® Cortex®-M4 32-bit MCU+FPU, 100DMIPS, up to 128KB Flash, 40KB SRAM, analog, ext. SMPS". STM32L412xx datasheet. DS12469 Rev 7, September 2019.

[8] STMicroelectronics. "STM32 Nucleo-32 boards (MB1180) ". UM1956 Rev 5, 2018.

[9] STMicroelectronics. "Getting started with STM32CubeL4 MCU Package for STM32L4 Series and STM32L4+ Series". UM1860 Rev 13, December 2019.

[10] https://en.wikipedia.org/wiki/MATLAB. "MATLAB".Wikipedia.

[11] M. Sami Fadali, Antonio Visioli. "Antialiasing filters". In Digital Control Engineering (Second Edition), 2013.

[12] https://en.wikipedia.org/wiki/Digital_filter."Digital filter". Wikipedia.

[13] Actel. "Improving ADC Results.Through Oversampling and Post-Processing of Data". January 2007.

[14] T. Dear, J. Chang, W. Chow, D. Wai, J. Wang. "First-Order RC and RL Transient Circuits". Electrical Engineering 42/100 Summer 2012. Department of Electrical Engineering and Computer Sciences University of California, Berkeley.

Appendix A

MATLAB code

A.1 Functions generated after the design of Cheby filters

```
function y = ChebyFilter1(x)
%DOFILTER Filters input x and returns output y.

% MATLAB Code
% Generated by MATLAB(R) 9.7 and DSP System Toolbox 9.9.
% Generated on: 10-Jun-2020 10:18:13

%#codegen

persistent Hd;

if isempty(Hd)

    % The following code was used to design the filter coefficients:
    %
    % Fpass = 1;     % Passband Frequency
    % Fstop = 1.5;   % Stopband Frequency
    % Apass = 1;     % Passband Ripple (dB)
    % Astop = 80;    % Stopband Attenuation (dB)
    % Fs    = 1600;  % Sampling Frequency
    %
    % h = fdesign.lowpass('fp,fst,ap,ast', Fpass, Fstop, Apass, Astop, Fs);
    %
    % Hd = design(h, 'cheby2', ...
    %     'MatchExactly', 'stopband', ...
    %     'SystemObject', true);

    Hd = dsp.BiquadFilter( ...
        'Structure', 'Direct form II', ...
        'SOSMatrix', [1 -1.99996458500042 1 1 -1.99913696140169 ...
        0.99915400963576; 1 -1.99995806575845 1 1 -1.9973147386843 ...
        0.997333149805012; 1 -1.99993925054548 1 1 -1.9951173703122 ...
        0.995138660093771; 1 -1.99988129356832 1 1 -1.992446475964 ...
        0.99247213904942; 1 -1.99956289752332 1 1 -1.98978522939044 ...
        0.989815670103883; 1 1 0 1 -0.994280600521733 0], ...
        'ScaleValues', [0.48138456232159; 0.439047801989281; ...
        0.350452225044885; 0.216189503413461; 0.0696420520701446; ...
        0.00285969973913346; 1]);
end

s = double(x);
```

```
y = step(Hd,s);
```

```
function y = ChebyFilter2(x)
%DOFILTER Filters input x and returns output y.

% MATLAB Code
% Generated by MATLAB(R) 9.7 and DSP System Toolbox 9.9.
% Generated on: 10-Jun-2020 12:02:02

%#codegen
persistent Hd;

if isempty(Hd)

    % The following code was used to design the filter coefficients:
    %
    % Fpass = 3;     % Passband Frequency
    % Fstop = 3.5;   % Stopband Frequency
    % Apass = 1;     % Passband Ripple (dB)
    % Astop = 80;    % Stopband Attenuation (dB)
    % Fs    = 1600;  % Sampling Frequency
    %
    % h = fdesign.lowpass('fp,fst,ap,ast', Fpass, Fstop, Apass, Astop, Fs);
    %
    % Hd = design(h, 'cheby2', ...
    %     'MatchExactly', 'stopband', ...
    %     'SystemObject', true);

    Hd = dsp.BiquadFilter( ...
        'Structure', 'Direct form II', ...
        'SOSMatrix', [1 -1.99980979556816 1 1 -1.99889511289001 ...
        0.999041440875156; 1 -1.99979897899548 1 1 -1.9968778500709 ...
        0.997030346568633; 1 -1.99977474777104 1 1 -1.99454843649983 ...
        0.994714312858045; 1 -1.99973046410583 1 1 -1.99163280182952 ...
        0.991821304623568; 1 -1.99965101890501 1 1 -1.98774466997648 ...
        0.987968607342168; 1 -1.99949930002953 1 1 -1.98234162147593 ...
        0.982619268885779; 1 -1.99916620153476 1 1 -1.97479309497699 ...
        0.975149296465124; 1 -1.99820893284197 1 1 -1.96494802558787 ...
        0.965409061799504; 1 -1.99303883444895 1 1 -1.95487096449132 ...
        0.955440452610676; 1 1 0 1 -0.975101717518861 0], ...
        'ScaleValues', [0.769319535454047; 0.758609768661389; ...
        0.736402738293566; 0.699360634773507; 0.641689102647641; ...
        0.554518526506645; 0.427203338676199; 0.257408668102648; ...
        0.0818093055214536; 0.0124491412405693; 1]);
end

s = double(x);
y = step(Hd,s);
```

A.2 Functions generated after the design of Elliptic filters

```
function y = EllipticFilter1(x)
%DOFILTER Filters input x and returns output y.

% MATLAB Code
% Generated by MATLAB(R) 9.7 and DSP System Toolbox 9.9.
% Generated on: 10-Jun-2020 10:20:47

%#codegen

```

```
persistent Hd;

if isempty(Hd)

    % The following code was used to design the filter coefficients:
    %
    % Fpass = 1;     % Passband Frequency
    % Fstop = 1.5;   % Stopband Frequency
    % Apass = 1;     % Passband Ripple (dB)
    % Astop = 80;    % Stopband Attenuation (dB)
    % Fs    = 1600;  % Sampling Frequency
    %
    % h = fdesign.lowpass('fp,fst,ap,ast', Fpass, Fstop, Apass, Astop, Fs);
    %
    % Hd = design(h, 'ellip', ...
    %     'MatchExactly', 'both', ...
    %     'SystemObject', true);

    Hd = dsp.BiquadFilter( ...
        'Structure', 'Direct form II', ...
        'SOSMatrix', [1 -1.99996616585537 1 1 -1.99972617881914 ...
        0.999741532372801; 1 -1.99995236844128 1 1 -1.99913046783303 ...
        0.999141707899633; 1 -1.99986446400695 1 1 -1.99845255091517 ...
        0.998457125311466; 1 1 0 1 -0.999053113667417 0], ...
        'ScaleValues', [0.109470417991286; 0.235979398999456; ...
        0.0337504170897813; 0.00196256832711693; 1]);
end

s = double(x);
y = step(Hd,s);
```

```
function y = EllipticFilter2(x)
%DOFILTER Filters input x and returns output y.

% MATLAB Code
% Generated by MATLAB(R) 9.7 and DSP System Toolbox 9.9.
% Generated on: 10-Jun-2020 12:03:58

%#codegen
persistent Hd;

if isempty(Hd)

    % The following code was used to design the filter coefficients:
    %
    % Fpass = 3;     % Passband Frequency
    % Fstop = 3.5;   % Stopband Frequency
    % Apass = 1;     % Passband Ripple (dB)
    % Astop = 80;    % Stopband Attenuation (dB)
    % Fs    = 1600;  % Sampling Frequency
    %
    % h = fdesign.lowpass('fp,fst,ap,ast', Fpass, Fstop, Apass, Astop, Fs);
    %
    % Hd = design(h, 'ellip', ...
    %     'MatchExactly', 'both', ...
    %     'SystemObject', true);

    Hd = dsp.BiquadFilter( ...
        'Structure', 'Direct form II', ...
        'SOSMatrix', [1 -1.99981250164446 1 1 -1.99911313680107 ...
        0.99924067025711; 1 -1.99975795751691 1 1 -1.99816203175828 ...
        0.998262866068294; 1 -1.9995365865229 1 1 -1.99666591122318 ...
        0.996722106639223; 1 -1.99669253368451 1 1 -1.99524631781953 ...
```

```
        0.995259608746892; 1 -1.99982866539889 1 1 -1.99965420784749 ...
        0.999792821735849], ...
        'ScaleValues', [0.720849656531487; 0.679865530313023; ...
        0.41630403752161; 0.121262806519511; 0.00402430783523777; 1]);
end

s = double(x);
y = step(Hd,s);
```

A.3 Functions generated after the design of FIR filters

```
function y = FirFilter1(x)
%FIRFILTER1 Filters input x and returns output y.

% MATLAB Code
% Generated by MATLAB(R) 9.7 and DSP System Toolbox 9.9.
% Generated on: 10-Jun-2020 10:55:23

%#codegen
persistent Hd;

if isempty(Hd)

    % The following code was used to design the filter coefficients:
    % % Equiripple Lowpass filter designed using the FIRPM function.
    %
    % % All frequency values are in Hz.
    % Fs = 1600;  % Sampling Frequency
    %
    % Fpass = 1;                % Passband Frequency
    % Fstop = 1.5;              % Stopband Frequency
    % Dpass = 0.057501127785;   % Passband Ripple
    % Dstop = 0.0001;           % Stopband Attenuation
    % dens  = 20;               % Density Factor
    %
    % % Calculate the order from the parameters using FIRPMORD.
    % [N, Fo, Ao, W] = firpmord([Fpass, Fstop]/(Fs/2), [1 0], [Dpass, Dstop]);
    %
    % % Calculate the coefficients using the FIRPM function.
    % b  = firpm(N, Fo, Ao, W, {dens});

    Hd = dsp.FIRFilter( ...
        'Numerator', [7.00619879875417e-05 1.73478607968007e-05 ...
        1.72789583086127e-05 1.73870310067519e-05 1.73184940224234e-05 ...
        1.74268805565349e-05 1.73582580923961e-05 1.74666120962446e-05 ...
        1.73972140032942e-05 1.75051129076567e-05 1.74340989340607e-05 ...
        1.75412442843242e-05 1.74677766112779e-05 1.75744589379368e-05 ...
        1.74980635615851e-05 1.76063535400176e-05 1.75276381304525e-05 ...
        1.76454252335782e-05 1.7567761830651e-05 1.77429088834945e-05 ...
        1.76390613978608e-05 1.7703996673789e-05 1.76728743826781e-05 ...
        1.77592663568883e-05 1.77082140045091e-05 1.78014244130081e-05 ...
        1.77461464609755e-05 1.78421445581116e-05 1.7784861603725e-05 ...
        1.78819947649748e-05 1.78230930761331e-05 1.79202286087439e-05 ...
        1.78596621757071e-05 1.79561511495473e-05 1.78944731581168e-05 ...
        1.79909960574468e-05 1.79309638549401e-05 1.80307316672074e-05 ...
        1.79792422194769e-05 1.80815376605328e-05 1.79986926614662e-05 ...
        1.81125130437816e-05 1.80456524820486e-05 1.81487526956243e-05 ...
        1.80845245492134e-05 1.81871460215035e-05 1.8123740470022e-05 ...
        1.82263980304429e-05 1.81633298567428e-05 1.82658507831638e-05 ...
                      %truncated because of too much coefficients
```

```
                              ]);
end

y = step(Hd,double(x));


% [EOF]
```

```
function y = FirFilter2(x)
%FIRFILTER2 Filters input x and returns output y.

% MATLAB Code
% Generated by MATLAB(R) 9.7 and DSP System Toolbox 9.9.
% Generated on: 10-Jun-2020 12:07:53

%#codegen

persistent Hd;

if isempty(Hd)

    % The following code was used to design the filter coefficients:
    % % Equiripple Lowpass filter designed using the FIRPM function.
    %
    % % All frequency values are in Hz.
    % Fs = 1600;  % Sampling Frequency
    %
    % Fpass = 3;               % Passband Frequency
    % Fstop = 3.5;             % Stopband Frequency
    % Dpass = 0.057501127785;  % Passband Ripple
    % Dstop = 0.0001;          % Stopband Attenuation
    % dens  = 20;              % Density Factor
    %
    % % Calculate the order from the parameters using FIRPMORD.
    % [N, Fo, Ao, W] = firpmord([Fpass, Fstop]/(Fs/2), [1 0], [Dpass, Dstop]);
    %
    % % Calculate the coefficients using the FIRPM function.
    % b  = firpm(N, Fo, Ao, W, {dens});

    Hd = dsp.FIRFilter( ...
        'Numerator', [8.60694574398233e-05 5.71237239930419e-05 ...
        5.70786953276382e-05 5.7181318242396e-05 5.71350199404078e-05 ...
        5.72358699889941e-05 5.71886162587477e-05 5.72878013087933e-05 ...
        5.72402053970038e-05 5.73381966439206e-05 5.72914072484539e-05 ...
        5.7389150475521e-05 5.73450841318962e-05 5.74439818418856e-05 ...
        5.74054298355619e-05 5.75065418263109e-05 5.74766803402434e-05 ...
        5.75765360086393e-05 5.75561283462276e-05 5.7607715535572e-05 ...
        5.75341473993673e-05 5.7695107041683e-05 5.76160860903625e-05 ...
        5.77322598076737e-05 5.76694517366086e-05 5.77782841120743e-05 ...
        5.77187676973492e-05 5.78242258841574e-05 5.77664060066618e-05 ...
        5.78702903686687e-05 5.78139744546279e-05 5.79171867488163e-05 ...
        5.78618550216397e-05 5.79638692297977e-05 5.79073791213498e-05 ...
        5.80057711449521e-05 5.79445313187625e-05 5.8038222569523 4e-05 ...
        5.79745555666997e-05 5.80898522937488e-05 5.80213983675039e-05 ...
        5.81109168888309e-05 5.80559424542975e-05 5.814452868259e-05 ...
        5.80870295599269e-05 5.81755966285127e-05 5.81172817554043e-05 ...
        5.8205555491353e-05 5.81465966746848e-05 5.82340737371533e-05 ...
        5.81738157192961e-05 5.82595678991653e-05 5.81971513695786e-05 ...
        5.82808705057872e-05 5.8216902704739 9e-05 5.83006490049763e-05 ...
        5.8237698067762e-05 5.83231665293195e-05 5.82554742522263e-05 ...
        5.83287814625952e-05 5.82695093997513e-05 5.83440747932906e-05 ...
        5.8277801442681e-05 5.83536458459249e-05 5.82854821243469e-05 ...
        5.83601159389341e-05 5.82900769490588e-05 5.83628472041363e-05 ...
```

```
        5.82906172195879e-05 5.83612939163457e-05 5.82871110649456e-05 ...
        5.8356265129037e-05 5.8280915836545e-05 5.83490027313029e-05 ...
                      %truncated because too much coefficients
       ]);
end

y = step(Hd,double(x));


% [EOF]
```

A.4 Code used to apply filters and evaluate FFTs

```
%read values
f=fopen("no_filter.txt","r");
avg_and_dec=textscan(f,"%d");
avg_and_dec=cell2mat(avg_and_dec);

%set parameters
sampling_frequency = 1600.0;
N = length(avg_and_dec);
delta_f = sampling_frequency / N;
xfft=( delta_f .* (0:1:N/2-1) );
% retrieve min value and subtract it
M=min(avg_and_dec);
for i=1:1:N
    avg_and_dec(i)=avg_and_dec(i)-M;
end

%FFT original
fft_avg_and_dec1 = abs(fft(avg_and_dec));
A = mean(avg_and_dec, 1);
fft_avg_and_dec2 = (fft_avg_and_dec1(1:N/2))./A/N;

%apply FIR filter and compute FFT
output2=FirFilter1(avg_and_dec);
output2_1 = abs(fft(output2));
Abutt = mean(output2, 1);
output2_2 = (output2_1(1:N/2))./Abutt/N;

%apply Cheby filter and compute FFT
output3=ChebyFilter1(avg_and_dec);
output3_1 = abs(fft(output3));
Acheb = mean(output3, 1);
output3_2 = (output3_1(1:N/2))./Acheb/N;

%apply Elliptic filter and compute FFT
output4=EllipticFilter1(avg_and_dec);
output4_1 = abs(fft(output4));
Aell = mean(output4, 1);
output4_2 = (output4_1(1:N/2))./Aell/N;

%plot section
plot(xfft,20*log10(fft_avg_and_dec2),'b'),title('FFT no analog filter')
hold on
plot(xfft,20*log10(output2_2),'r')
hold on
plot(xfft,20*log10(output3_2),'black')
hold on
plot(xfft,20*log10(output4_2),'g'),legend('nodigfilter','Fir','Cheby','Elliptic')
```