Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Fast Digital Notch Filter

Semester project
Philip Leindecker
15 September, 2020

Advisors: P. Zupancic, A. Baumgärtner and Prof. T. Esslinger
Departement of Physics, ETH Zürich

Abstract

During the semester project a fast digital notch filter on a microcontroller-based feedback device 'LockStar' was implemented for improved frequency stabilization in optical cavities. With this filter algorithm the device is able to cancel or modulate multiple resonant frequencies at once and deliver real-time feedback to the system. Due to the customizable initialization parameters the algorithm can be easily applied to various different situations without any additional development while reaching suppressions up to -60 dB with a sampling rate of up to 100 kHz.

Contents

1 Introduction

Active stabilization of experimental parameters, such as laser power or length of optical resonators, play a crucial role in many applications in modern physics. The 'IMPACT' lab in Tilman Esslinger's group at ETH Zurich studies long-range interacting quantum systems. In order to achieve stable and reproducible results in these kinds of experiments, the optical cavities, in which a Bose-Einstein condensate (BEC) is placed, must be actively stabilized. An acousto-optic modulator (AOM) can be used for controlling the laser power, or piezoelectric actuators attached cavity mirrors for stabilizing the cavity length. The 'IMPACT' lab is currently developing the digital feedback device 'LockStar', which should act as a highly adaptable and 'smart' control device for these parameters, which otherwise are usually controlled by analog devices.

The goal of the semester project is to improve the active frequency stabilization for this feedback device. The main focus is to cancel out mechanical resonances of a given signal in real time. By applying a digital filter, using a low number of selected Fourier coefficients representing a custom version of the Goertzel Algorithm, it was possible to suppress such resonances up to -60 dB. Due to the limited amount of processing time within the real-time response of the microcontroller, other algorithms like calculating the full FFT or performing large real-time convolutions [1] are not possible in this case.

I will demonstrate the results in various different cases such as cancelling out resonances of integer and non-integer multiples of the fundamental frequency or cancelling out nearby resonances. The current implementation of the algorithm also allows selective phase shifting or amplitude modulation for the different resonances and can also adapt to real-time changes of these parameters. Additionally, I will compare the measured computational run time related to the different numbers of cancelled frequencies and other initial parameters.

The approach was originally developed for active stabilization of laser frequencies in optical cavities, but can easily be extended to other fields such as metrology or atomic microscopy [1].

In Section 2 I will give a short introduction to the hardware in use. In Section 3 I will present the filter algorithm, its mathematical derivation for the different steps and a comparison to the known Goertzel algorithm. Finally, in Section 4 I will summarize and compare different cases, such as cancelling out single and multiple integer and non-integer multiples of the fundamental frequency.

2 Hardware

2.1 LockStar

The digital feedback device, also called LockStar (Fig. 2.1), first build and designed by Philip Zupancic and then further improved by Philip Rhyner during his Master Thesis [4], consists mainly of the following four components:

- **ADC** (analog digital converter): Allows to measure the signals from the experiment and sends them to the microcontroller for further operation. Hardware: LTC2353 Dual 16-bit ADC with $>$1000 GΩ input impedance and successive approximation register (SAR) to digitise the analog signal.
- **DAC** (digital analog converter): Two DACs are used to give the electrical feedback from the microcontroller to the system. Hardware: AD5791 from Analog Devices, which is a single channel 20-bit voltage output DAC.
- **Microcontroller**: Manages the other two previously mentioned components and calculates the real time feedback for our applications. Hardware: STM32F427
- **Raspberry Pi**: Enables easy access and programming of the microcontroller and due to its Ethernet connector also allows remote control of the device. Operating System (OS): Linux

With the given 16-bit input and 20-bit output resolution, the device can achieve a bandwidth of more than 200 kHz for closed-loop feedback operation. This makes the device a fast, accurate and versatile solution for amplitude and frequency stabilization.

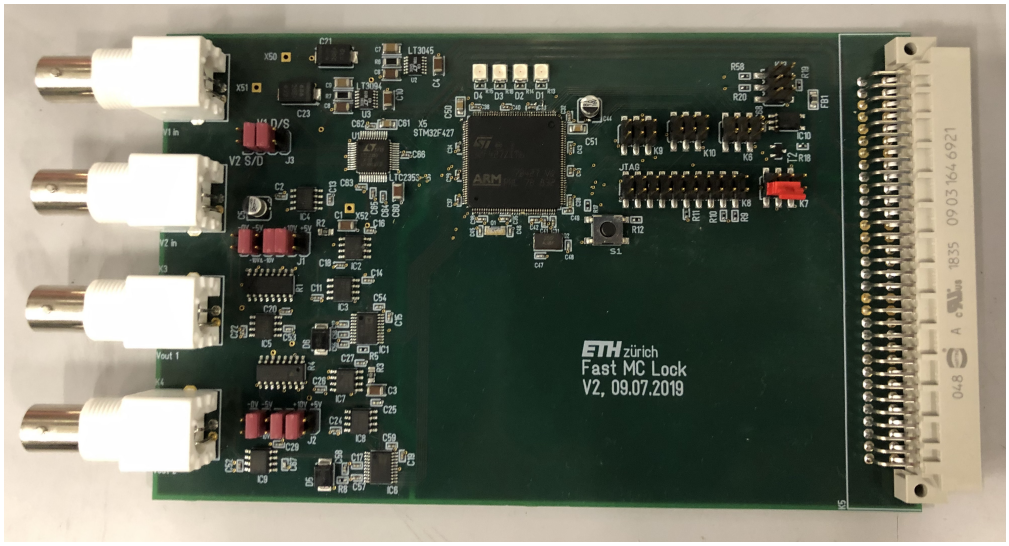


Figure 2.1: Picture of the LockStar (without the Raspberry Pi) printed circuit board (PCB); Left: four BNC connectors connected to the ADCs and DACs; Center: Microcontroller (32-bit ARM Cortex-M4 processor); Right: Power adapter

2.2 Frequency Lock

A frequency lock of the cavity system is achieved by actively adapting to changes in the length of the optical cavity via piezoelectric actuators which are mounted on top of the cavity mirrors (Fig. 2.2). The microcontroller uses a proportional–integral–derivative (PID) control loop to give feedback to the piezoelectric actuators. A PID controller is a conventional mechanism to bring a process variable (laser-intensity) as close as possible to a setpoint.

The problem while using a PID loop for stabilization are mechanical resonances in the system, since they represent non-linearities in the otherwise linear transfer function of a PID controller, and thus would lead the control system to overshoot. After a short period of time, this results into a unstable cavity which is not locked any more. Therefore the goal of this semester project is to filter out these resonances and thus improve this PID controller.

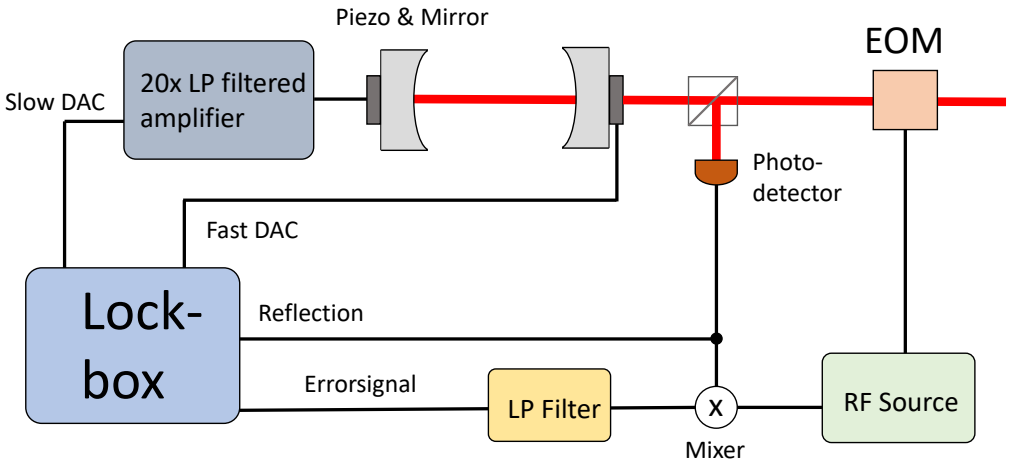


Figure 2.2: Schematic overview of the hardware set-up. The two analog outputs are used to control the mirror piezos. One output is amplified and low-pass filtered and the other goes directly to the piezo. As analog inputs into the Lock-box we have the reflection, or sometimes transmission, signal and the error-signal produced in combination with a electro-optic modulator (EOM).

3 Filter Algorithm

In digital signal processing (DSP), various different filters with distinct capabilities exist, such as Low- and High-pass filters, to suppress certain frequencies above or below a cut-off frequency.

In this case the goal is to cancel out single or multiple frequencies to avoid (mechanical) resonances in the cavity-system, thus to implement a digital band-stop filter (or notch-filter) on the microcontroller. This section presents the necessary steps for implementing such a filter algorithm.

3.1 Recursive DFT

In contrast to the ordinary fast Fourier transform (FFT), which calculates the full spectrum of a discrete signal, the following recursive discrete Fourier transform (recursive DFT) allows one to computationally efficiently determine single Fourier components of defined frequencies [4]. This is necessary for the microcontroller to give appropriate real-time feedback to the system.

Given a discrete-time series signal $\{x_n\}$ in a running window (= batch) of $n = [m, m + N - 1]$, with N being the size of the running window, the k^{th} Fourier component is given by

$$X_k^{(m)} = \sum_{n=0}^{N-1} x_{n+m} \cdot e^{-i2\pi kn/N} \tag{3.1}$$

If we now shift the running window one data point further to $n = [m + 1, m + N]$, we can rewrite the k^{th} Fourier component in the following manner

$$\begin{aligned} X_k^{(m+1)} &= \sum_{n=0}^{N-1} x_{n+m+1} \cdot e^{-i2\pi kn/N} \\ &= \sum_{n=1}^{N} x_{n+m} \cdot e^{-i2\pi k(n-1)/N} \\ &= e^{i2\pi k/N} \sum_{n=1}^{N} x_{n+m} \cdot e^{-i2\pi kn/N} \\ &= e^{i2\pi k/N} \left(\sum_{n=0}^{N-1} x_{n+m} \cdot e^{-i2\pi kn/N} - x_m + x_{N+m} \right) \\ &= e^{i2\pi k/N} \left(X_k^{(m)} - x_m + x_{N+m} \right) \end{aligned} \tag{3.2}$$

This equations shows that by storing the past N data points of the signal, one can calculate the current FFT component by just replacing the old data point x_m with the newest one x_{N+m}. Since at the very beginning no data points are present, the first N points are set to zero. Therefore the algorithm needs to read exactly N data points to be fully initialized (see Fig. 3.1).

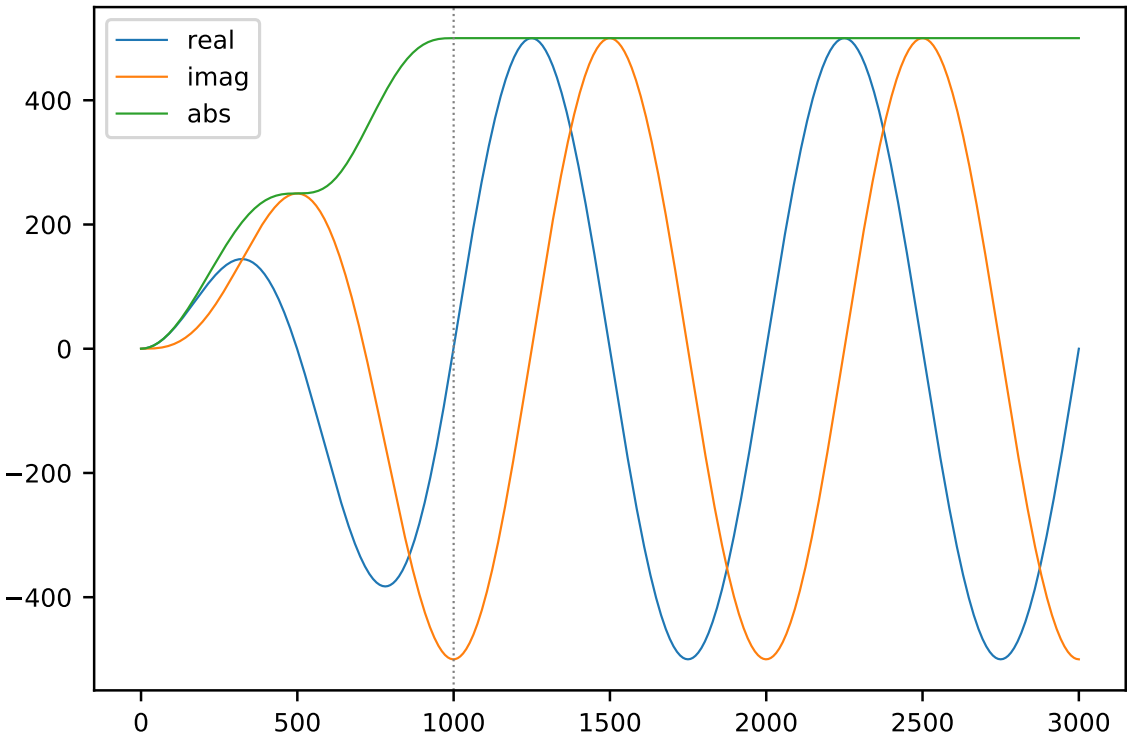


Figure 3.1: Real and imaginary part together with their absolute value of the recursively calculated Fourier coefficients. After a first initialization period of a 1000 data points (up to the dashed vertical line) the coefficients reach their exact values for the given batch size.

3.2 Inverse DFT

With Fourier components given like in Eq. 3.1, the inverse FFT (iFFT) in general looks like the following

$$x_k^{(m)} = \frac{1}{N} \sum_{n=0}^{N-1} X_{n+m} \cdot e^{i2\pi kn/N} \tag{3.3}$$

Another possibility to transform the Fourier coefficients back to the real signal is to re-construct the separate sine-waves of the components like this

$$x_k = A_k \sin(2\pi f_k t + \varphi_k) \tag{3.4}$$

with the corresponding amplitude A_k and phase φ_k given by:

$$A_k = |X_k| = \sqrt{\mathrm{Re}(X_k)^2 + \mathrm{Im}(X_k)^2} \tag{3.5}$$

$$\varphi_k = \mathrm{atan2}\left(\frac{\mathrm{Im}(X_k)}{\mathrm{Re}(X_k)}\right) \tag{3.6}$$

3.3 FFT Interpolation

One major limitation of the DFT approach is the restriction to frequencies, which can be resolved with the chosen Fourier spectrum. The resolution defining each spectrum is given by $\frac{\text{batchsize}}{\text{samplerate}}$.

There are several possibilities to adapt the resolution of the spectrum, which all come with their own advantages and disadvantages:

- **Increase/Decrease batch size**: With larger batch size, the resolution of the spectrum increases. The drawback is, that with larger batch size, the system also reacts slower to changes in amplitude or phase of the frequencies.
- **Increase/Decrease sampling rate**: Smaller sampling rate means higher resolution but slower reaction to changes and also a lower Nyquist frequency, up to which the system can cancel out resonances.
- **Padding**: This approach artificially increases the batch size with zeros, or the mean value of the signal, to increase the resolution. It is suited for short time FFTs, but it is difficult to apply it to long time signals.
- **FFT interpolation**: This approach uses the nearest complex Fourier coefficients around the desired non-integer frequency to interpolate the desired frequency together with a sinc function. The disadvantage in this case is the increased computational cost, which is necessary to determine the extra Fourier componentns around this frequency [5].

FFT Interpolation for a arbitrary frequency $f_r = r/N$, with r being any real number and N =batch size, can best be summarized in the following equation

$$A_r \simeq \sum_{[r]-m/2}^{[r]+m/2} A_k \cdot e^{-i\pi(r-k)} \,\mathrm{sinc}[\pi(r-k)] \tag{3.7}$$

A_k are the complex FFT amplitudes at the integer frequencies k, $[r]$ is the nearest integer to r and m is the number of neighboring FFT coefficients used in the interpolation. The $\mathrm{sinc}(x) = \frac{\sin(x)}{x}$ function (see Fig. 3.2) is the key to computing an accurate interpolated amplitude using a relatively small number of operations. Since $\mathrm{sinc}[\pi(r-k)] \to 0$ as $\pi(r-k) \to \pm\infty$, the expansion of A_r in terms of the A_k is dominated by the local Fourier amplitudes (i.e., where $k \sim r$).

A higher number of neighboring FFT coefficients m results in a more accurate cancelling of the desired frequencies. To make the algorithm more adaptable to different use cases, this number m adds an additional degree of freedom to the final implementation.

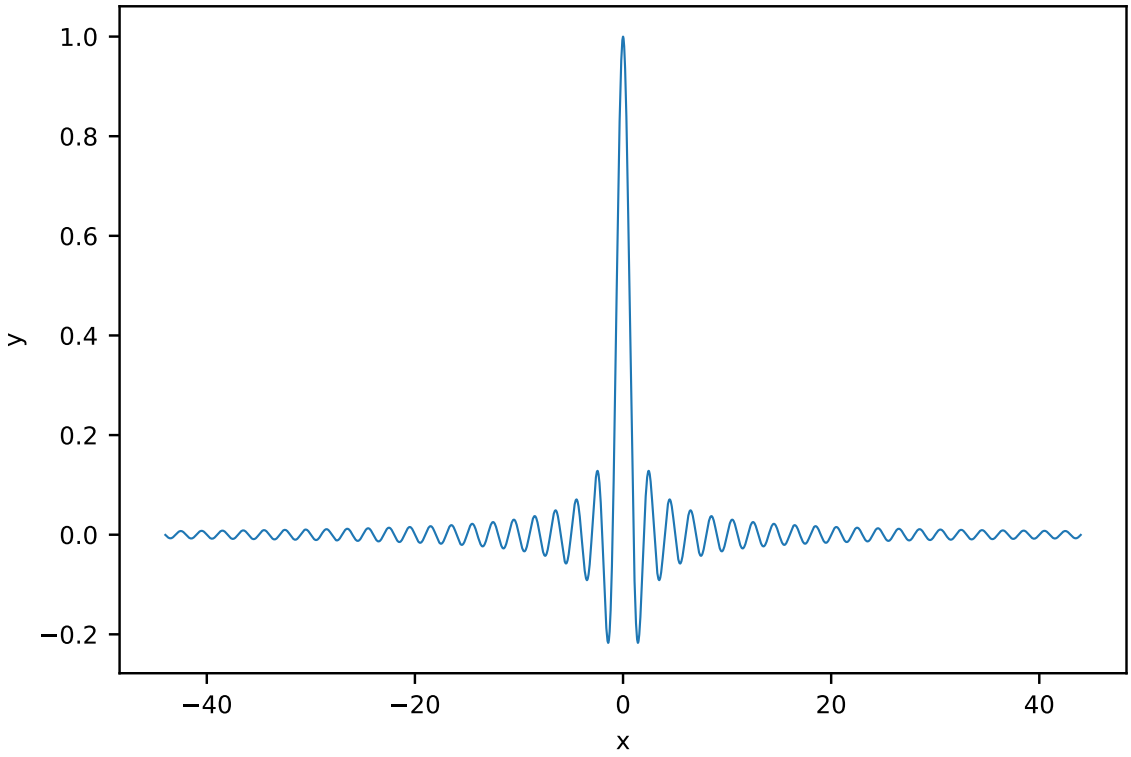


Figure 3.2: $\mathrm{sinc}(x) = \frac{\sin(x)}{x}$ in the interval between $[-14\pi, 14\pi]$.

3.4 Final Algorithm

The final algorithm running on the microcontroller is written in C++ and combines all the previous steps into the *FFTCorrection* class.

3.4.1 Computational Optimizations

Since all the previous described steps rely on relatively complex functions such as: $\sin(x), \tan 2(x), e^x, ...$, the computational processing time is still rather high on the microcontroller although only the recursive FFT is performed. In order to get rid of these functions and to further decrease the number of calculations following optimizations for each step were implemented:

- **Recursive DFT**: In Eq. 6.1 one can pre-calculate the factor $e^{i2\pi k/N}$ for all relevant frequencies, which reduces the final real-time calculation to just one multiplication, one subtraction and one addition, thus three low-level operations in total.
- **Inverse DFT**: Using trigonometric relationships Eq. 3.5 can be expanded in the following

manner,

$$x_k = A_k[\sin(2\pi f_k t)\cos(\varphi_k) + \cos(2\pi f_k t)\sin(\varphi_k)] \tag{3.8}$$

$$\cos(\varphi_k) = \cos\left[\operatorname{atan2}\left(\frac{\mathrm{Im}(X_k)}{\mathrm{Re}(X_k)}\right)\right] = \frac{\mathrm{Re}(X_k)}{A_k} \tag{3.9}$$

$$\sin(\varphi_k) = \sin\left[\operatorname{atan2}\left(\frac{\mathrm{Im}(X_k)}{\mathrm{Re}(X_k)}\right)\right] = \frac{\mathrm{Im}(X_k)}{A_k} \tag{3.10}$$

Performing this expansion allows again for pre-computing the two factors $s_k = \sin(2\pi f_k t)$ and $c_k = \cos(2\pi f_k t)$, which in the end results in the vastly simplified equation for x_k

$$x_k = A_k[s_k\,\mathrm{Re}(X_k) + c_k\,\mathrm{Im}(X_k)] \tag{3.11}$$

Finally, the iDFT for each component consists again only of three multiplications and one addition, so four operations in total.

- **FFT interpolation**: In Eq. 3.7 it is also possible to pre-calculate the relatively complex factors $e^{-i\pi(r-k)}\,\mathrm{sinc}[\pi(r-k)]$ for each A_k and thus reduce the whole FFT correction to only one single multiplication for each component.

3.4.2 Numerical-Error Accumulation

The implementation of the algorithm in C++ comes with both advantages and disadvantages. The main advantages are clearly the speed together with the simplicity of adapting code quickly and remotely on the microcontroller as opposed to other approaches like using field-programmable gate arrays (FPGAs) as described in this paper [1].

The main disadvantages although are the limitations to certain data types given by the native instruction-set of the microcontroller together with their low-precision algebra in C++. The data type *Double* is for example not represented in this native instruction-set and thus requires the usage of the data type *Float*, which can only store up to seven significant decimal digits. This limitation leads to numerical-error accumulation for the recursively calculated Fourier coefficients described in Sec. 3.1.

To overcome this problem, one can accept additional computing time by calculating a parallel running recursive DFT. In the case of only one running DFT the accumulated numerical error gets larger with every new data point and eventually would completely distort the calculated Fourier coefficients. The parallel running FFT allows for limiting this error accumulation to only one batch size. The implementation in C++ can be elegantly done with pointers in the following way:

1. Initialize the first running FFT called '*runningFFT1*' and assign it to a similar called pointer '*activeFFT**'.
2. When the first running FFT is fully initialized, start initializing the second running FFT called '*runningFFT2*' and also assign it to a pointer called '*passiveFFT**'.

3. Let the algorithm run for another full batch size, where the *runningFFT1* is the used for the iFFT in the algorithm. In parallel *runningFFT2* also gets fully initialized.

4. Switch the active and passive FFT pointers to the respectively opposite running FFTs, so that the *activeFFT* Pointer now points to *runningFFT2* and the *passiveFFT* Pointer points to *runningFFT1*. This switching of pointers allows for greater efficiency instead of just having to copy the whole arrays.

5. Reset *runningFFT1*, so all entries become zero again. This eliminates all error accumulation from the previous batch.

6. Let the algorithm run for another full batch size, where now *runningFFT2* is being used for the iFFT and *runningFFT1* is being initialized again in parallel.

7. Repeat steps 4 - 6 every time a full batch size of new data points is reached.

To summarize, every time one recursive FFT is fully initialized, the other one is reset to zero and starts again. This means that the numerical-error is only accumulated over a short time of one batch size (see Fig. 3.3). The pointer to the fully initialized DFT which is used for the iFFT is called '*active FFT**', the other one is called '*passive FFT**'.

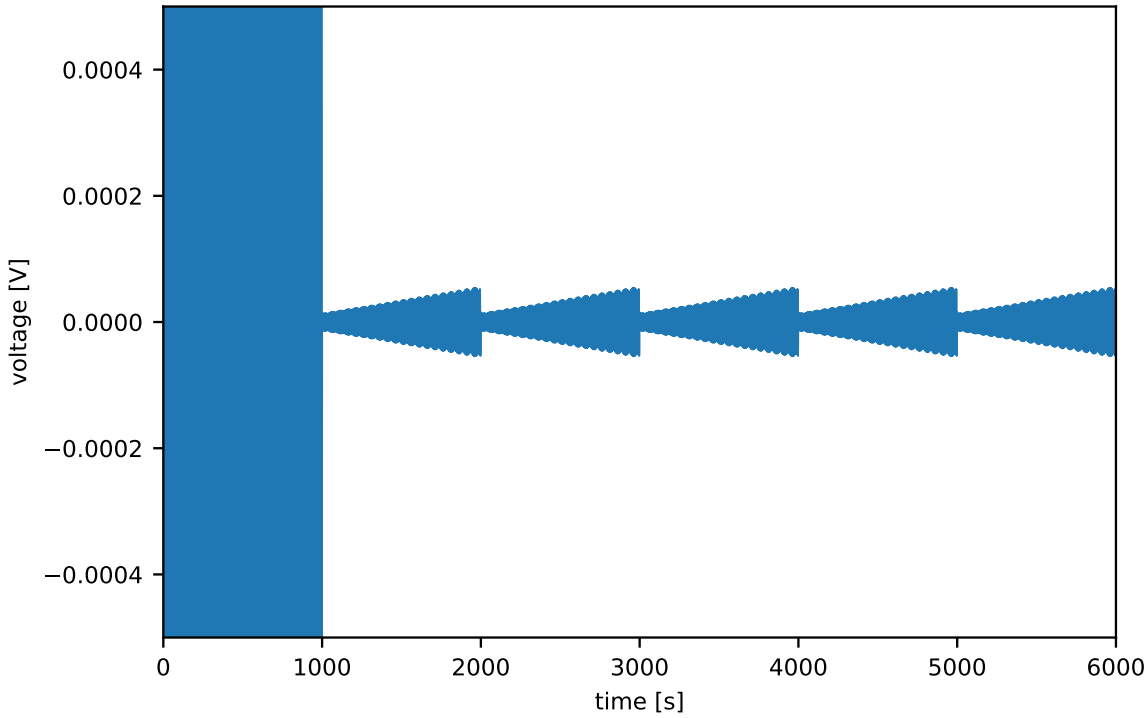


Figure 3.3: This plot is a simulation to demonstrate the numerical-error accumulation during each new batch of the filter algorithm. It represents the remaining amplitude of a single frequency in an artificial signal (a simple sine-wave), which is being cancelled out with the filter algorithm. The batch size in this case is a 1000 data points. During the first 1000 data points the filter algorithm is being initialized, thus the amplitude of the cancelled signal remains still relatively high. Then, in every subsequent set of a 1000 data points, the algorithm is fully initialized and the effect of numerical error-accumulation can be inferred. At the beginning of each batch, the remaining amplitude is noticeable smaller than at the end of each batch. This effect originates in using two parallel running FFTs, where every time a new batch is started, one FFT is being reset to zero and initialized again and the other one is being used to cancel out the signal.

3.4.3 Implementation

The implementation of the *FFTCorrection* class (see Class-Diagram 3.4) can be divided into two main parts:

1. Pre-calculations and initializations before the actual filtering starts.
2. The primary running filter function called *CalculateCorrection*.

The initialization of the pre-calculated factors such as amplitude correction or Fourier coefficients are

only performed once at the very beginning, before the first correction is made. Input for each initialization are the following parameters:

Initial parameters	**Sample rate**	Hz
	Batch size	Data points
	Resonant peaks	Frequencies which should be cancelled or corrected, given as $\{Peak\}$
Peak	**Frequency**	Hz
	Fourier coeffs.	Number of neighboring FFT coefficients for the FFT interpolation. Must be 1, for a frequency which is exactly in the spectrum, or even for the interpolation.
	Amplitude scaling	Factor which scales the amplitude of the correction. Set to 0 for just cancelling the peak.
	Phase shift	Factor $\in [0, 2\pi]$ which shifts the phase of the correction.

The primary *CalculateCorrection* function (see code snippet in Fig. 3.5) is used for the actual active cancelling of the desired frequencies and has the following properties:

CalculateCorrection	**Input**	Current ADC value pv
	Output	Frequency corrected pv value, which is then further corrected in the PID loop

FFTCorrection	
+ sampleRate	*float*
+ batchSize	*float*
+ resonantPeaks	*vector* <peak>
- runningIndex	*int*
- runningFFT1, runningFFT2	*vector* <fftEntry>
- *activeFFT, *passiveFFT	*vector* <fftEntry>
- fourierCoeffs	*vector* <*complex* <*float*>>
- amplitudeCorr	*vector* <*complex*<*float*>>
...	
+ CalculateCorrection(pv: float)	*float*
+ Setup()	*void*
- SwapFFTs()	*void*
- ResetActiveFFT()	*void*
...	

(a) FFTCorrection-Class

fftEntry	
+ fft	*complex* <*float*>
+ supportSin	*float*
+ supportCos	*float*
+ shiftSin	*float*
+ shiftCos	*float*

(b) fftEntry-Struct

peak	
+ freq	*float*
+ fftCoeffs	*float*
+ amplitudeShift	*float*
+ phaseShift	*float*

(c) peak-Struct

Figure 3.4: FFTCorrection-Class-Diagram; Contains only the most relevant attributes and functions.

3.5 Goertzel Algorithmus

The above algorithm can bee seen as a custom version of the Goertzel algorithm.
The Goertzel algorithm was first introduced in 1958 by G. Goertzel [2] and enables one to (computationally) efficiently determine the kth DFT component of a discrete-time signal $\{x_n\}$ of length N. With the frequency $\omega_0 = \frac{2\pi k}{N}$ to be analysed, it is represented by the following two equations, where only the output y_n is of interest in the end.

$$s_n = x_n + 2\cos(\omega_0)s_{n-1} - s_{n-2} \tag{3.12}$$

$$y_n = s_n - e^{-i\omega_0}s_{n-1} \tag{3.13}$$

At the beginning s_{-1} and s_{-2} are set to zero. For each iteration, it applies a single real-valued coefficient, using real-valued arithmetic for real-valued input sequences.
Covering a full spectrum is computationally less efficient with using the Goertzel algorithm and thus it should only be applied when analysing less than $\log_2 N$ frequencies.

Additionally, the Goertzel algorithm is also vulnerabe to numerical-error accumulation using low-precision arithmetic and long input sequences. For a more detailed description, derivation of the algorithm and generalization to non-integer multiples of fundamental frequencies see [3].

```
float FFTCorrection::CalculateCorrection(float pv) {

    pvOut = pv;
    diffSignalX = pv - runningSignal[runningIndex];

    for (int i = 0; i < runningFFTSize; i++) {
        // Running FFT
        (*activeFFT)[i].fft = fourierCoeffs[i] * ((*activeFFT)[i].fft +
            diffSignalX);
        (*passiveFFT)[i].fft = fourierCoeffs[i] * ((*passiveFFT)[i].fft + pv);

        // Calculate Amplitude Correction
        tempResult = (*activeFFT)[i].fft * amplitudeCorr[i];
        // Generate output signal
        pvOut -= amplitudeNorm * ((*activeFFT)[i].supportSin * tempResult.real
            () +
        (*activeFFT)[i].supportCos * tempResult.imag());
        pvOut += amplitudeNorm * ((*activeFFT)[i].shiftSin * tempResult.real()
             +
         (*activeFFT)[i].shiftCos * tempResult.imag());
    }

    runningSignal[runningIndex] = pv;
    runningIndex += 1;
    if (runningIndex == batchSize) {
        runningIndex = 0;
        // reset active FFT to zero
        ResetActiveFFT();
        // swap active and passiveFFT
        SwapFFTs();
    }

    return pvOut;
}
```

Figure 3.5: C++ implementation of the *CalculateCorrection* function. **Input**: Current ADC value pv; **Output**: frequency corrected pv value, which is then further corrected in the PID loop.

4 Results

This section summarizes several different cases in which the *FFTCorrection* algorithm can be applied. They range from cancelling out single frequencies in the spectrum with different input parameters, to phase shifting and dealing with multiple frequencies. Two quantities, filter suppression in [dB] and run time in [μs], are mainly used to compare these cases. Additionally, the full width at half maximum (FWHM) in Hz of the peaks are added for a better understanding of the quality of the filter.

The results are illustrated by the transfer function of the system, simulated with a Bode Analyser. If not stated otherwise, the following plots are all performed with a sample rate of 10 kHz and a batch size of 1000 data points. The FFT resolution in these cases is 10 Hz with a Nyquist frequency of 5 kHz.

4.1 Test Setup

The test setup for the LockStar during the project consisted mainly of two additional devices (Fig. 4.1). First a Bode Analyser (*OMICRON LAB*, Bode 100), to measure the transfer function of the system, and secondly a PicoScope (*pico Technology*, PicoScope 4824) to measure the run time of the algorithm. Both devices can be operated via a remote connection.

Additionally, a simulated Bode Analyser in Python was implemented, which also uses the *FFTCorrection* algorithm written in C++ in order to compare the measured results from the Bode Analyser with their theoretical limit.

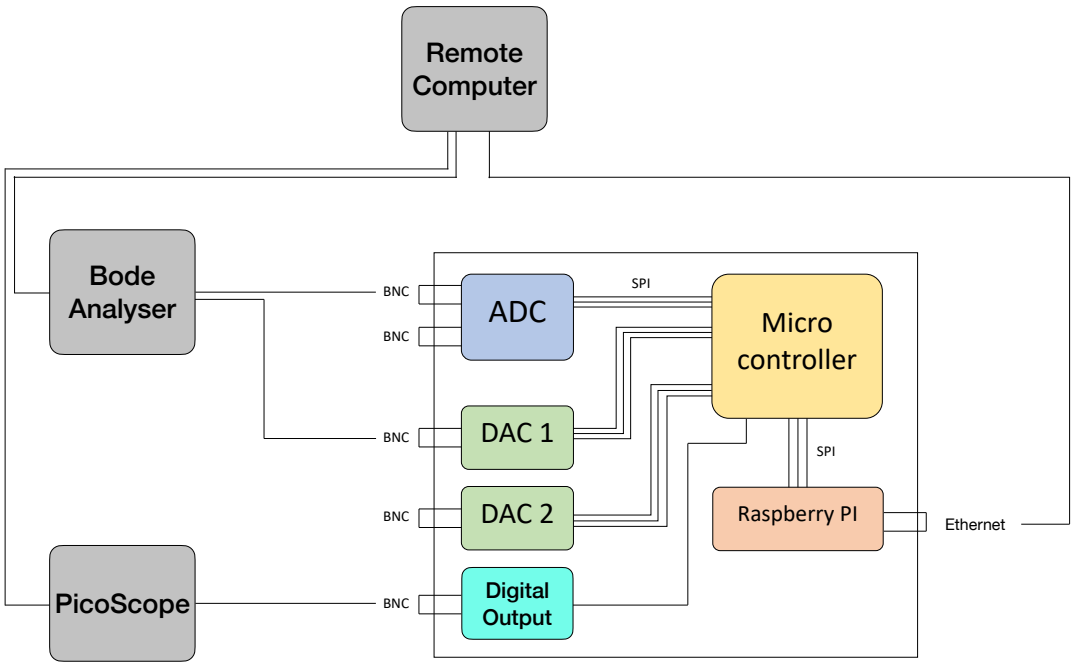


Figure 4.1: Test setup for the LockStar frequency response; Left: External test devices (Bode Analyser, PicoScope). The Bode Analyser gives a sinusoidal input signal, and measures the corresponding output signal from the microcontroller. The Picoscope measures the overall run time of microcontroller to process the signal and apply the *FFTCorrection* algorithm; Top: External remote computer to control the setup; Right: LockStar with ADCs, DACs, Microcontroller and Raspberry Pi

4.1.1 Bode Analyser

The Bode Analyser Suite (Fig. 4.2) allows one to measure the magnitude of the frequency response in [dB] and the phase shift of the system in [°]. Depending on the initial sweep parameters, the Bode Analyser has different sweep velocities, which has to be taken into account while testing the time-depended filter algorithm presented in the next chapter. Since the filter algorithm needs a certain time to be fully initialized, the time for the Bode Analyser for going from one measured data point to the next one should be less than the initialization time of the filter algorithm in order to achieve comparable results.

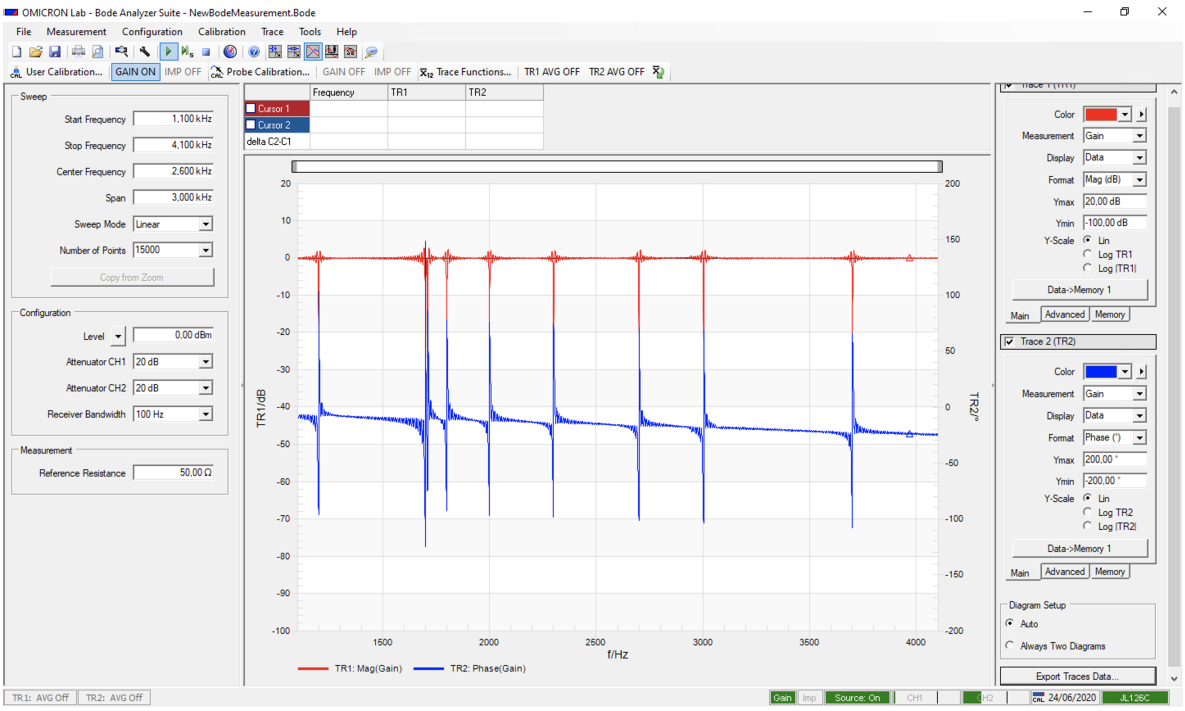


Figure 4.2: Bode-Analyser-Suite; Red: magnitude of the frequency response in [dB]; Blue: phase shift of the system in [°]; Left: Initial Sweep parameters

4.1.2 PicoScope

The computational run time for the final filter algorithm can be measured by sending from one Digital-Output of the LockStar a TTL signal (in our case 5 V) directly before the filter algorithm is performed and then setting it back to zero voltage, after the filtering has finished. This 'step' function can then be measured with a PicoScope, which is acting as a digital oscilloscope.

Fig. 4.3 presents the plot of such a run time step-function of the filter algorithm while cancelling out a single frequency. The duration in this case is approximately 2.5 μs and can be divided into the microcontroller back-log consisting mainly of reading the ADC input (0.2 μs) and writing to the DAC output (1.1 μs) and the actual filtering-time (in the case of a single frequency: 1.2 μs).

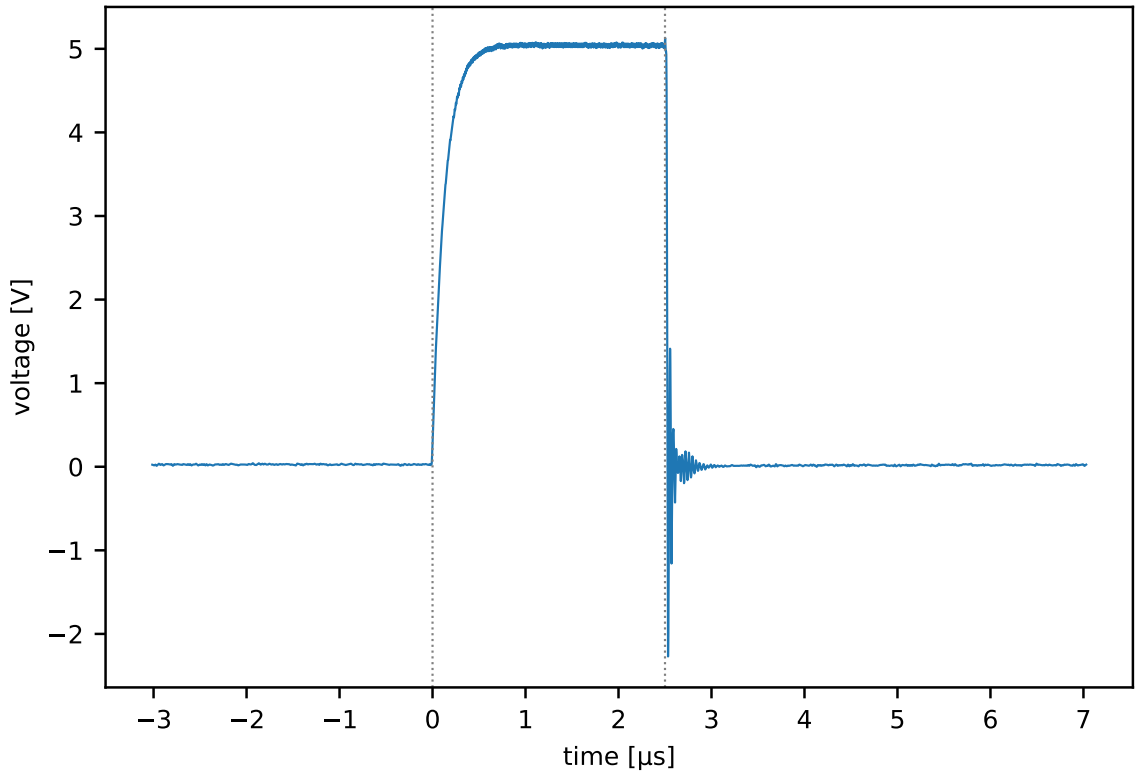


Figure 4.3: This step function measured with the PicoScope represents the total time for cancelling out a single frequency on the microcontroller. The duration of the run time in this case is approximately 2.5 μs, which can be divided into the microcontroller back-log, such as reading and returning the analog inputs (1.3 μs), and the actual filtering-time (1.2 μs)

4.2 Transfer Function

The transfer function T_f is a mathematical function, that theoretically models the system's output for each possible input and is mainly used in this section to illustrate the results. It is calculated via the following equations (N = batch size):

$$r_f = \frac{2}{N} \sum_{i=0}^{N} s_{\text{in}}(f, x_i) \cdot s_{\text{out}}(f, x_i) \tag{4.1}$$

$$T_f = 10 \log_{10}(|r_f|) \tag{4.2}$$

r_f represents a correlation of the original signal s_{in} (a simple sine-wave with the given frequency f) and the processed signal from the filter algorithm s_{out}. Finally, for T_f this correlation is transformed into the logarithmic scale since suppression is measured in [dB].

A pole in the transfer function represents a frequency which is cancelled out in the original signal. The two main properties of such a pole are:

- Depth in [dB], which determines the quality of the filter suppression
- Line width in [Hz], which determines how accurate the filter is confined to the desired frequency

The logarithmic scale of the transfer function distorts the results for the line with due to non-linear scaling. Thus for measuring the correct line width the $|r_f|$ value is used. In this thesis the line with is determined by the size of the pole at half maximum of the pole (see Fig. 4.4) also called full-width at half maximum (FWHM).

Additionally, each pole also has some 'ripples' on both sides which can be modelled as a sinc function (see Fig. 3.2) and depends on several factors such as sampling rate or batch size. This 'ringing' effect near the edges of the cancelled frequency is called Gibbs phenomenon and arises due to the discontinuity created by the filter algorithm. In the case of cancelling out resonances in our cavity system this effect is still small enough, but it might has an impact on other applications.

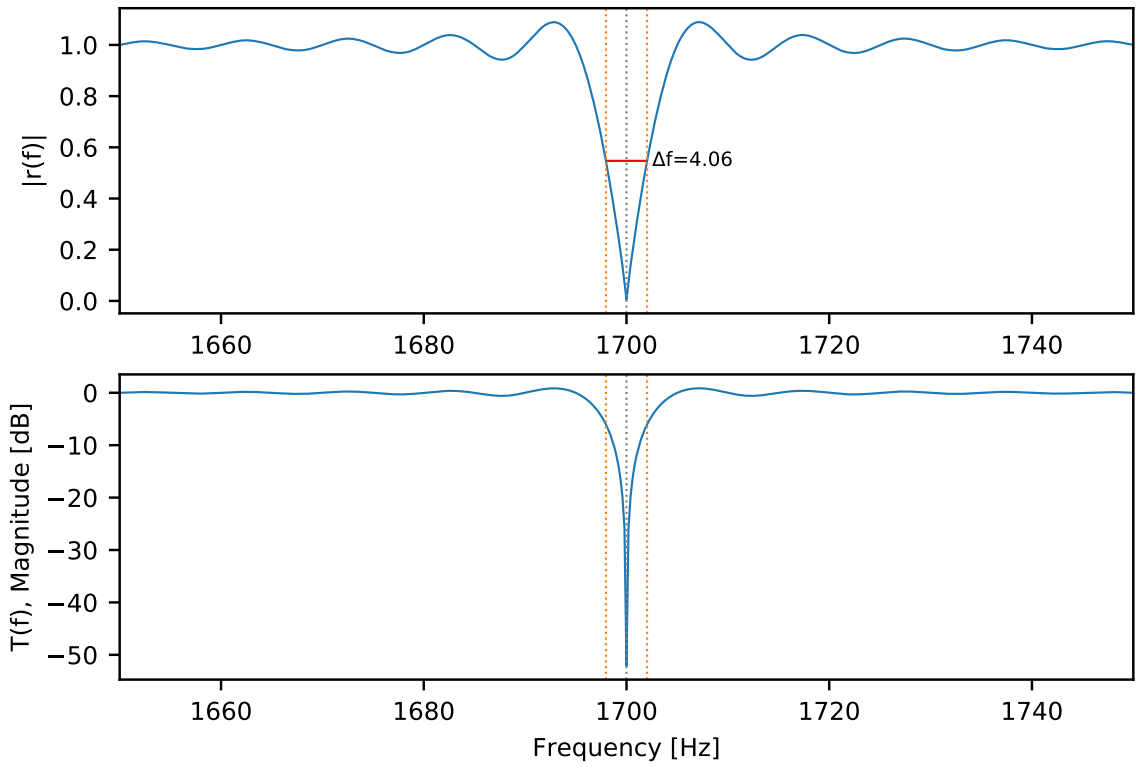


Figure 4.4: The first graph represents the simulated $|r_f|$ value, the second graph illustrates the corresponding simulated Transfer function T_f in the case of cancelling out the frequency f=1.7kHz (FWHM = 4.06 Hz, Filter suppression =-52.08 dB).

4.3 Single Frequency

4.3.1 Cancel Frequency in the Spectrum

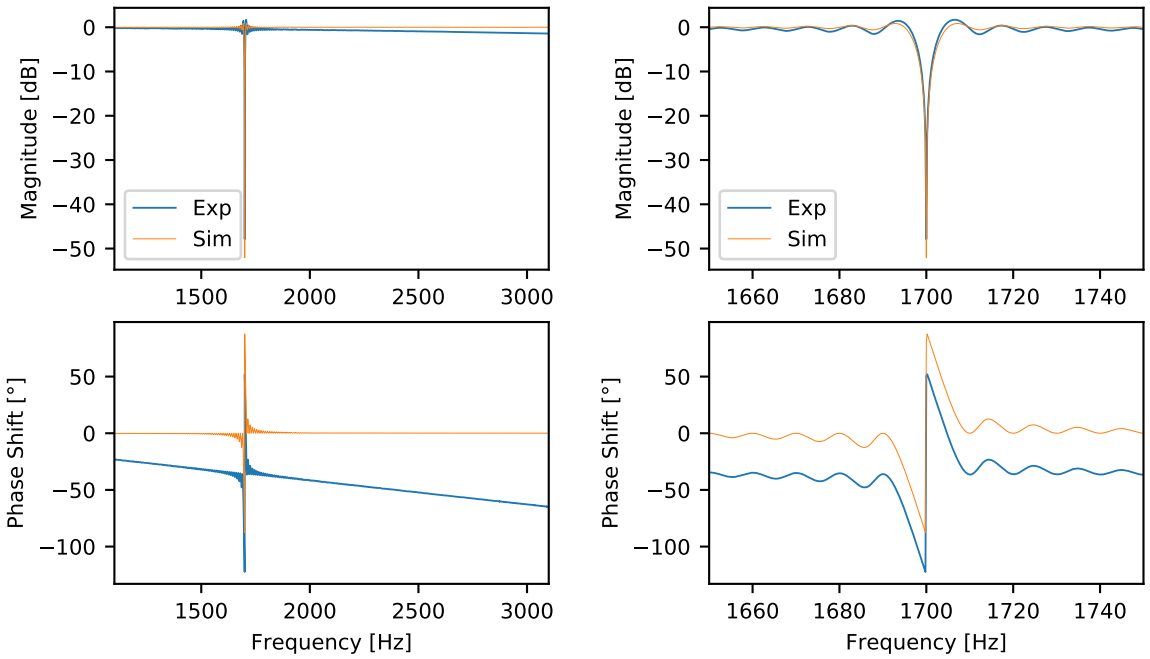
Tab. 1 summarizes the filter quality and run time for frequencies, which are integer-multiples of the fundamental frequency. Note that for every preceding run time, one has to subtract 1.3 μs, that is the time the program takes without applying the *FFTCorrection* algorithm.

Parameters		Results		
Frequency [kHz]	Fourier coeffs.	Filter quality [dB] (theory)	FWHM [Hz] (theory)	Run time [μs]
1	1	-47.91 (-52.08)	3.73 (4.06)	2.5
2	1	-42.07 (-47.56)	3.77 (4.07)	2.5
2	1	-40.47 (-45.56)	3.78 (4.08)	2.5

Table 1: Comparison of the filter quality for different frequencies between 1.7 - 2.7 kHz, thus integer multiples of the fundamental frequency. Both experimental and (theory) results are displayed for filter quality and FWHM. The (theory) results are generated with the simulated Bode Analyser.

Fig. 4.5 illustrates what effect cancelling out a single frequency which is represented in the spectrum, has on the transfer function of the system (Frequency: 1.7 kHz). The first row represents the magnitude of the signal in [dB], with a wide scan (in the interval [1.1 kHz, 3.1 kHz]) on the left hand side and a narrow scan (in the interval [1650 Hz, 1750 Hz]) on the right hand side. The second row presents the corresponding phase in [$^\circ$] for both wide and narrow scan.

The downwards trend in the phase-plot for the experimental data in Fig. 4.5 originates in the internal time delay on the microcontroller. Thus this trend is not visible in the simulated data.



(a) Wide-scan

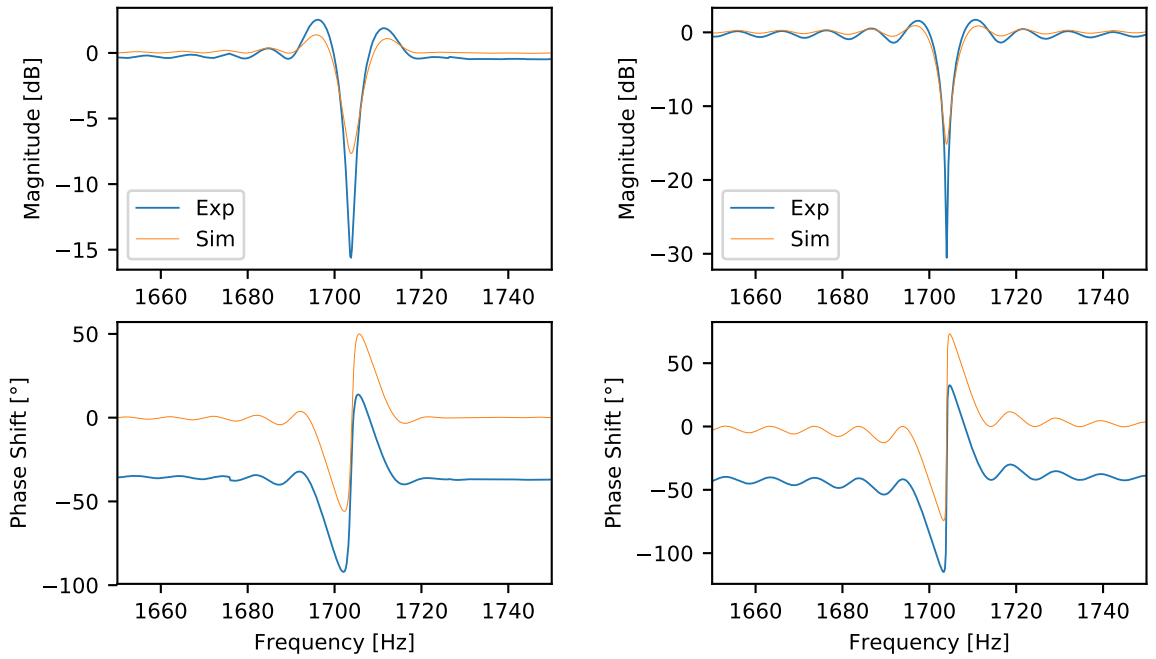
(b) Narrow-scan

Figure 4.5: Effect of cancelling out a single frequency (Frequency: 1.7 kHz).

4.3.2 Cancel Arbitrary Frequency

To demonstrate the effect of FFT interpolation, Fig. 4.6 illustrates the case of cancelling out a frequency not present in the spectrum (Frequency: 1704 Hz). One can see, that with a smaller number of just 2 neighboring Fourier coefficients taken into account, as presented on the left hand side, the quality of the filter is about a factor of two worse (in dB) than the filter on the right hand side with 12 neighboring Fourier coefficients.

The results from the simulated Bode Analyser in this section lack the same quality of suppression as the experimental measured data, although the results for the simulated suppression should always exceed the experimental one. This discrepancy (see Fig. 4.6, Tab. 2, 3, 4) might originate in the finite precession algebra of C++ and needs some further investigation.



(a) Coefficients: 2

(b) Coefficients: 12

Figure 4.6: Effect of cancelling out a single Frequency, which is not represented in the spectrum (Frequency: 1704 Hz). Each plot with a different number of Fourier coefficients taken into account.

Tab. 2 summarizes the increasing filter quality and run time for a increasing number of neighboring Fourier coefficients (Frequency: 1703 Hz). With every new pair of Fourier coefficients taken into account for the FFT interpolation, the run time increases about 1.7 μs.

Parameters		Results		
Frequency [Hz]	Fourier coeffs.	Filter quality [dB] (theory)	FWHM [Hz] (theory)	Run time [μs]
1703	2	-18.49 (-9.05)	4.53 (5.79)	3.3
1703	4	-24.25 (-11.88)	4.21 (5.34)	5.0
1703	6	-27.7 (-13.6)	4.1 (5.14)	6.7
1703	8	-30.1 (-14.85)	4.03 (5.02)	8.3
1703	10	-31.96 (-15.83)	3.96 (4.93)	10.0
1703	12	-33.6 (-16.63)	3.93 (4.87)	11.7

Table 2: Comparison of the filter quality for 1703 Hz with different number of neighboring FFT coefficients.

In Tab. 3 different filter qualities for the frequencies between 1700 and 1705 Hz are represented, while using FFT interpolation. One can infer from that table, that by considering 12 neighboring

Fourier coefficients, the filter for the non-integer multiples of the fundamental frequency can deliver approximately the same order of magnitude as for integer multiples using only one Fourier coefficient.

Parameters		Results		
Frequency [Hz]	Fourier coeffs.	Filter quality [dB] (theory)	FWHM [Hz] (theory)	Run time [μs]
1700	1	-47.91 (-52.08)	3.73 (4.06)	2.5
1701	12	-46.41 (-25.3)	3.77 (4.39)	11.7
1702	12	-38.85 (-19.51)	3.84 (4.66)	11.7
1703	12	-33.6 (-16.63)	3.93 (4.87)	11.7
1704	12	-30.54 (-15.17)	3.98 (5.0)	11.7
1705	12	-29.71 (-14.73)	3.99 (5.06)	11.7

Table 3: Comparison of the filter quality for different frequencies between 1700 and 1705 Hz, thus for non-integer multiples of the fundamental frequency.

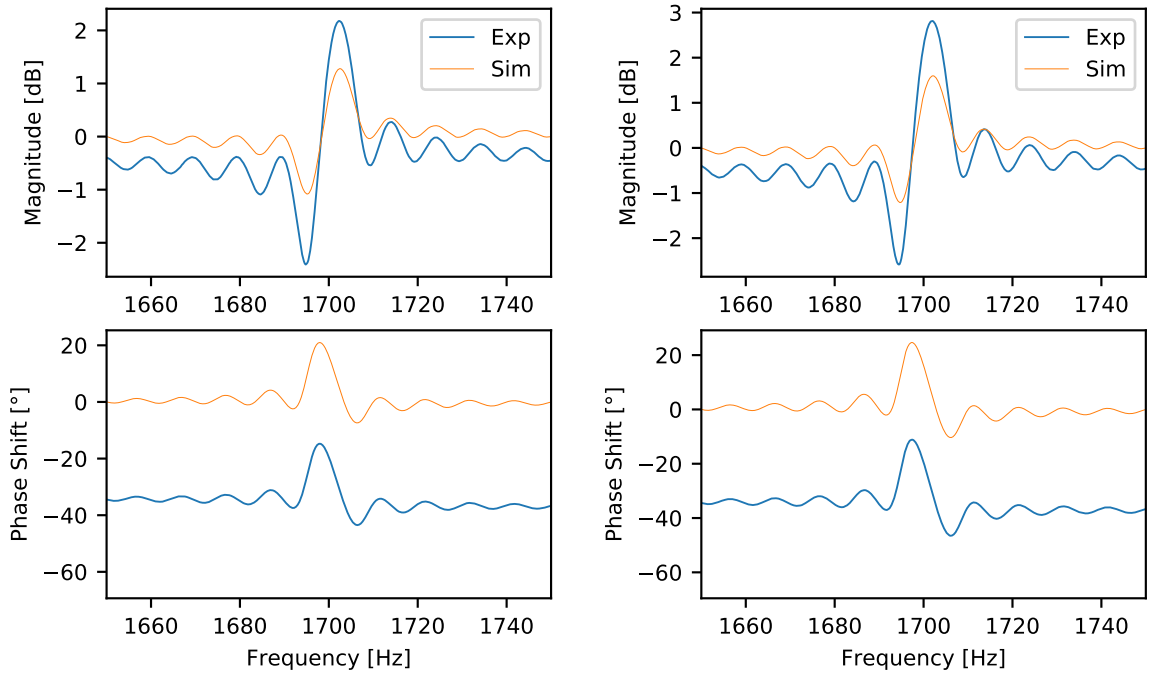
Finally, Tab. 4 compares the results for different sample rates and batch sizes. In the case of a sample rate of 170 kHz and a batch size of 1000, the run time of the algorithm exceeds the available bandwidth of the microcontroller and thus no cancelling is visible (marked by '-'). Additionally, this table indicates the inverse relation ship between batch size and line width of the frequencies.

Parameters			Results		
Sample rate [kHz]	Batch size	Frequency [kHz]	Fourier coeffs.	Filter quality [dB] (theory)	FWHM [Hz] (theory)
10	100	1.7	1	-61.09 (-66.85)	37.96 (40.44)
10	500	1.7	1	-40.59 (-54.59)	7.62 (8.1)
10	1000	1.7	1	-47.91 (-52.08)	3.73 (4.06)
10	2000	1.7	1	-37.48 (-49.05)	1.9 (2.03)
20	1000	1.7	1	-39.14 (-47.66)	7.63 (8.14)
50	1000	1.7	1	-59.29 (-46.75)	18.79 (20.35)
100	1000	1.7	1	-51.36 (-52.51)	37.55 (40.46)
170	1000	1.7	1	-	-

Table 4: Comparison of the results for different sample rates and batch sizes (Frequency: 1.7 kHz). If the run time of the algorithm exceeds the available bandwidth of the microcontroller and no cancelling is visible it is marked by '-'.

4.3.3 Phase-Shift/Amplitude-Modulate Frequency

Instead of cancelling a frequency, the algorithm also allows phase shifting and amplitude modulation, represented in Fig. 4.7 (Frequency: 1.7 kHz; Phase shift: $\frac{\pi}{2}$; Amplitude modulation: 2).



(a) Phase shift: $\frac{\pi}{2}$

(b) Amplitude modulation: 2

Figure 4.7: Effect of phase shifting (a) and amplitude modulating (b) a single Frequency (Frequency: 1.7 kHz)

4.4 Two Frequencies

In Fourier transformations and interpolations, neighboring Fourier coefficients can influence each other and effect the overall quality of the filter. Tab. 5 presents the results of two frequencies being filtered out while having different distances to each other. The results indicate a quite evenly distributed filter quality, which marginally decreases by a small factor when the two frequencies are directly next to each other in the spectrum.

Parameters		Results		
Frequency [Hz]	Fourier coeffs.	Filter quality [dB] (theory)	FWHM [Hz] (theory)	Run time [μs]
1700, 2700	1, 1	-47.64, -40.61 (-52.07, -45.57)	3.74, 3.9 (4.06, 4.09)	3.3
1700, 2000	1, 1	-47.62, -45.81 (-52.05, -47.86)	3.75, 3.89 (4.07, 4.07)	3.3
1700, 1800	1, 1	-47.77, -48.99 (-52.0, -48.24)	3.77, 3.82 (4.09, 4.04)	3.3
1700, 1710	1, 1	-40.27, -44.14 (-50.83, -49.36)	2.88, 4.06 (4.03, 3.55)	3.3

Table 5: Comparison of the filter quality for cancelling two frequencies.

Fig. 4.8 illustrates the transfer function with two directly neighboring FFT frequencies being filtered out.

Similar to Sec. 4.3.2 the simulated data differs from the experimental data in an unexpected way. Instead of only two single peaks for each cancelled frequency the peaks have a smaller symmetrical aligned smaller side peak (see Fig. 4.8).

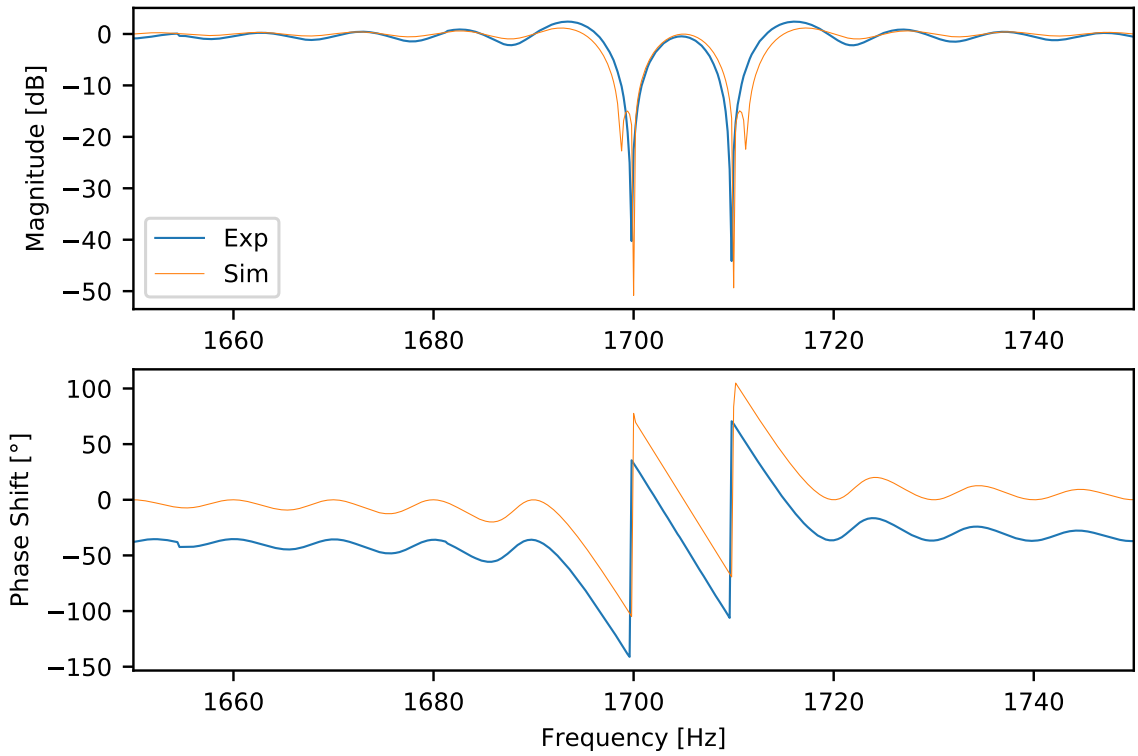


Figure 4.8: Effect of cancelling two neighboring frequencies in the FFT spectrum (Frequencies: 1700 Hz, 1710 Hz)

4.5 Multiple Frequencies

Finally, to demonstrate the capabilities of the filter algorithm for multiple frequencies, Fig. 4.9 illustrates the transfer function with ten different frequencies (Tab. 6) being cancelled out. The suppression in this case ranges from -30 dB to -52 dB and thus presents a valid filter for all the selected resonances.

The suppression in the simulated data mostly outperforms the suppression in the measured data, but still lacks some precession in four of the ten cancelled frequencies.

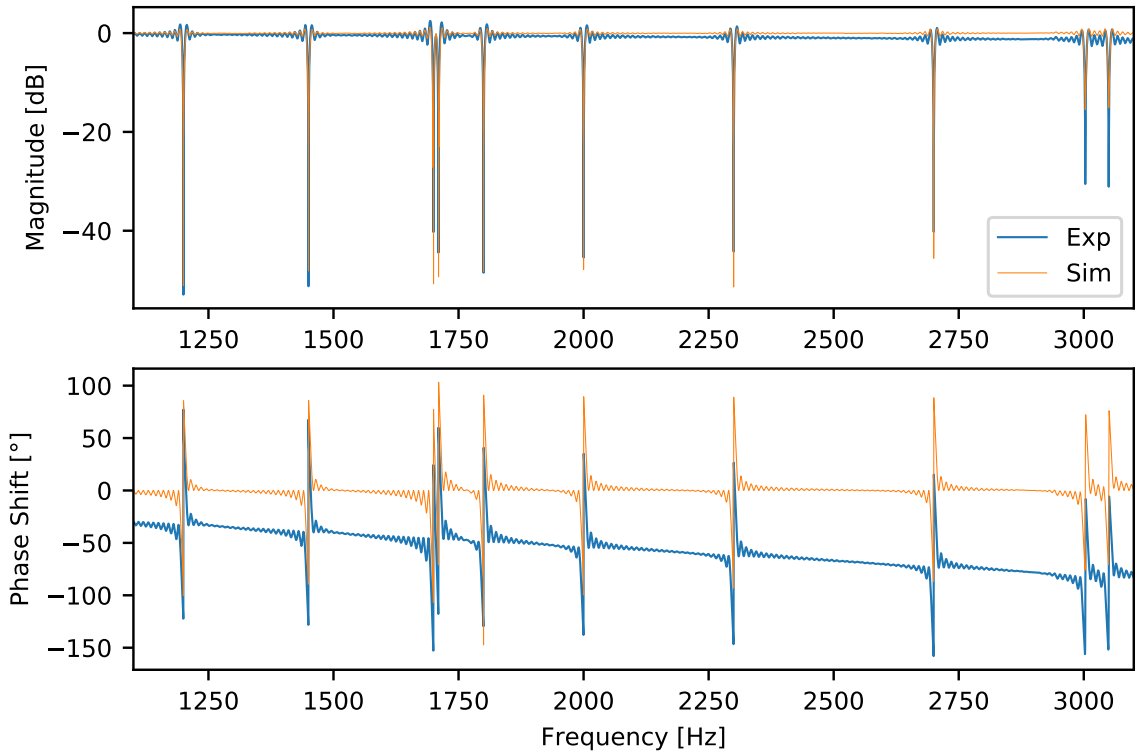


Figure 4.9: Effect of cancelling ten frequencies.

Parameters		Results	
Frequency [Hz]	Fourier coeffs.	Filter quality [dB] (theory)	FWHM [Hz] (theory)
1200	1	-52.94 (-51.12)	3.82 (4.12)
1450	1	-51.24 (-48.15)	3.83 (4.13)
1700	1	-40.28 (-50.77)	2.85 (4.04)
1710	1	-44.45 (-49.32)	4.04 (3.53)
1800	1	-48.5 (-22.93)	3.99 (0.33)
2000	1	-45.44 (-47.84)	3.96 (4.13)
2300	1	-44.22 (-51.4)	3.95 (4.12)
2700	1	-40.24 (-45.58)	3.94 (4.12)
3003	12	-30.54 (-15.46)	4.14 (5.07)
3050	1	-31.09 (-15.11)	4.16 (5.07)

Table 6: Summary of all the frequencies being cancelled in Fig. 4.9. Run time for this configuration is 19.1 μs

5 Conclusion

During the semester project a fast digital notch filter on a microcontroller-based feedback device was implemented for improved frequency stabilization in optical cavities. With this filter algorithm the device is able to cancel or modulate multiple resonant frequencies at once and deliver real-time feedback to the system. Due to the customizable initialization parameters, the algorithm can be easily applied to various different situations without any additional development.

Comparing this algorithm with the approach presented in a paper from J. Simon et al. [1], one can infer that the main difference is their ability to indirectly calculate the whole Fourier spectrum with a bandwidth of 100 kHz due to their higher processing power of a state-of-the-art field-programmable gate array (FPGA). Whereas in our filter algorithm only a few selected Fourier coefficients are calculated in order to stay within the bandwidth of the mircocontroller. The main advantage of our approach is the relatively easy, quick and even remote programmable and adaptable microcontroller to new and different systems in comparison to the low-level programming of the FPGAs.

The next steps for this project are

- Apply the filter algorithm to a real physical cavity, for which the resonances have been identified beforehand.
- Add further filters like a high- or low-pass filter to have even more control over the transfer function of the system.

6 Acknowledgements

I want to thank Philip Zupancic and Alexander Baumgärtner for their support and helpful discussions during this semester project. I really enjoyed working in the 'IMPACT' lab under their supervision, even with the sometimes difficult situations due to the COVID-19 pandemic.

Finally, I also want to thank Prof. Tilman Esslinger for his captivating lecture in Quantum Optics and the opportunity for being part of the 'IMPACT' lab at ETHZ.

References

[1] J. Simon A. Ryou, *Active cancellation of acoustical resonances with an fpga fir filter*, Review of Scientific Instruments **88** (2017).

[2] G. Goertzel, *An algorithm for the evaluation of finite trigonometric series*, American Mathematical Monthly **65** (1958), 34–35.

[3] P. Rajmic P. Sysel, *Goertzel algorithm generalized to non-integer multiples of fundamental frequency*, EURASIP Journal on Advances in Signal Processing **56** (2012).

[4] P. Rhyner, *Building microcontroller-based smart locks for quantum optics experiments*, Master Thesis, ETHZ (2019).

[5] J. Middleditch S.M. Ransom, S. S. Eikenberry, *Fourier techniques for very long astrophysical time-series analysis*, The Astronomical Journal **124** (2002), 1788–1809.

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

FAST DIGITAL NOTCH FILTER

Authored by (in block letters):

For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**
LEINDECKER	PHILIP

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**
Zürich, 15.09.2020	Philip Leindecker

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.