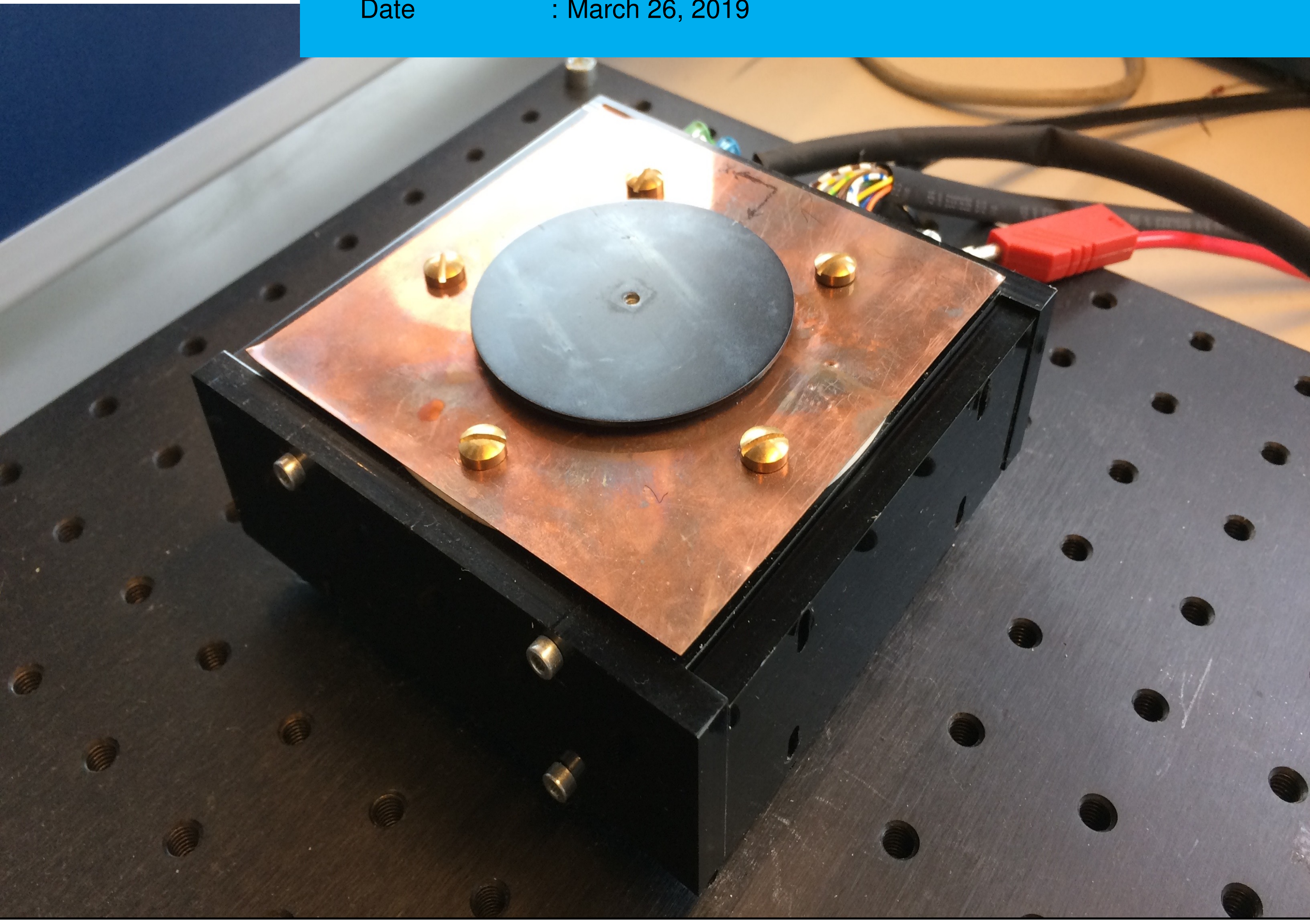
Department of Precision and Microsystems Engineering

A compact low-cost XY360°stage

Wouter Jutte

Report no	: 2019.011
Coach	: Dr. N. Saikumar
Professor	: Ir. J.W. Spronck
Specialisation	: Mechatronic System Design
Type of report	: Master thesis
Date	: March 26, 2019



A compact low-cost XY360°stage

using Arduino and PWM based amplifiers

by

Wouter Jutte

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Tuesday March 26, 2019 at 12:45 PM.

Student number:	4128249	
Thesis committee:	Ir. J.W. Spronck,	TU Delft, supervisor
	Dr. N. Saikumar,	TU Delft
	Prof Dr. Ir. G.V. Vdovine	TU Delft

Glossary

ADC Analog-to-Digital Converter. vii, xi, 2, 3, 11–17, 23, 29–34, 37, 41, 44–46, 61, 65, 67–70

DAC Digital-to-Analog Converter. 2, 19, 22, 38

DMA Direct Memory Acces. vii, 17, 18, 26, 38, 46

DOF Degrees Of Freedom. 37, 43, 47, 53–55

ENOB Effective Number Of Bits. 11–14, 17, 18, 32, 34, 37, 44

FFT Fast Fourier Transform. 12–15, 17

lsb least significant bit. 11, 61, 65, 68, 69, 71

MA Moving Average. 17

MSD Mechatronic System Design. vii, 2, 3, 43, 51

PID Proportional-Integral-Derivative. viii, 7, 25, 29, 31, 39, 46, 75

PSD Position Sensitive Detector. vii, viii, xii, 11, 17, 18, 29, 43, 54, 58, 60, 61

PWM Pulse Width Modulation. vii, viii, 2–5, 19–22, 25, 29, 37, 38, 43, 45, 46, 75

RPi Raspberry Pi. 56

SINAD SIgnal-to-Noise-And-Distortion. 13, 14, 18

SNR Signal-to-Noise-Ratio. 11–14, 17, 18, 26, 41, 65, 67, 71

Preface

I almost can't believe that I'm writing my final words as a student at TU Delft. Working my way through my bachelor's degree, I often questioned where I was going to use all this theory for. In the master everything started to fall into place, with this thesis as the icing on the cake. I couldn't have imagined that so many different fields of engineering come together in what looks like such a simple concept as positioning. On top of that, I've got to experience the practical side of things. It really was the most valuable and enjoyable experience out of all the years at TU Delft (which you tend to forget when you're writing your thesis).

It wouldn't have been such a good experience without the people from the Mechatronic System Design group. First and foremost I want to thank Jo Spronck for being the best supervisor I could have hoped for. When progress is slow or you're just not feeling it, having a meeting with someone as positive and funny as Jo really works magic to get you in the right head space again. Always giving feedback from a positive angle and always willing to help. I also want to thank Niranjan, Hassan and Andres for all the feedback and making the monday morning meetings such an enjoyable experience.

Wouter Jutte
Delft, March 2019

Abstract

The application of actively controlled micrometer precision positioning stages in products that are accessible to start-ups, individual consumers or third world countries is currently limited by their price and size. To solve this problem, the use of low-cost and compact components in stages is investigated. Earlier research done in the MSD group already showed that relatively low-cost sensor concepts and mechanical designs can be applied to significantly reduce the price of stages, while still achieving micrometer precision[22][41][35]. In this research, the applicability of the Arduino Due micro-controller and amplifiers based on Pulse Width Modulation (PWM) in precision stages is investigated.

The Arduino Due and PWM based amplifiers are the new selected components, which are part of a bigger mechatronic system. The performance of the new components can therefore only be evaluated with a whole system in mind. The choice is made to use the mechanical structure and position sensitive detector(PSD) of Haris Habib's design[22], who achieved a precision of 0.2 µm. An exploded view of his stage can be seen in Figure 1 and a more in depth system overview is added in Appendix C.

The Arduino Due micro-controller(€40) and PWM based amplifiers(€6) will replace the dSPACE DS-1103 controller(€20000) and linear current amplifiers(€1000) in the original design, with the goal to maintain as much of the original precision as possible. Even though the comparison in price isn't completely fair, because application specific alternatives for the dSPACE system and linear current amplifiers can most definitely be purchased for less, it will still be significantly more expensive than the Arduino and PWM based amplifiers.

Obviously, using low-cost and compact micro-controllers and amplifiers doesn't come without its drawbacks. To name a few examples, the Analog-to-Digital Converter (ADC) in the Arduino doesn't have the resolution of higher quality components, PWM adds frequency content to the output of the amplifier and the Arduino processor isn't quite as fast. An extensive theoretical analysis is done to reveal how the Arduino and PWM based amplifiers affect the overall performance of the stage, which can then be used to optimize system parameters to circumvent the performance losses that are inevitable with cheap components.

The 12-bit ADC on the Arduino board (which experimentally only showed 8 effective bits) isn't accurate enough to get the desired precision. A technique used in digital signal processing, which involves oversampling and filtering[33], is used to improve the accuracy of the ADC. Experiments of the ADC in isolation showed that every time the number of samples, taken and averaged, is increased by a factor of 4, a gain in accuracy equivalent to almost 1 bit is obtained.

PWM based amplifiers have several advantages over linear current amplifiers. PWM based amplifiers don't require bulky heatsinks, high-power op-amps and digital-to-analog converters, which makes them very cost-effective and compact. However, PWM has several disadvantages as well. The pulse wave used to set the average voltage adds harmonic content to the output of the amplifier. The extra harmonic content can lead to audible noise and a position disturbance. By setting the pulse wave to 42 kHz, extra harmonic content will only occur at 42 kHz and higher harmonics. This is outside of the audible range and audible noise is therefore avoided. Evaluation of the transfer function, that describes how amplifier errors transfer over to the position, showed that the low-pass characteristics attenuate the frequency content to a negligible amount. Other disadvantages of PWM, such as the limited output resolution and the amplifier acting as a voltage source also didn't show a significant effect on the position for the setup used in this project.

The control algorithm is implemented on the Arduino. Algorithm design and the hardware together determine the frequency at which the program can run. To make optimal use of oversampling and filtering, extra samples are taken within a program cycle with the use of the Arduino feature Direct Memory Access(DMA). DMA allows measurements to be written directly to memory without intervention of the CPU and therefore allows for simultaneous sampling and processing. The data size required, and therefore also the processing time, to gain accuracy in the data acquisition grows exponentially. This leads to a trade-off between the benefits of a higher program frequency vs the benefits of having a better data acquisition accuracy.

The Arduino Due and PWM based amplifiers are implemented into a complete system after the components are addressed in isolation, leading to the complete system illustrated in Figure 2. Measurement noise became so dominant over the other disturbance sources, due to the required measurement processing circuits and an already noisy sensor, that optimal PID design for precision meant designing for low control bandwidth. The PID controller is therefore designed for 25 Hz bandwidth. Heavy analog low-pass filtering can't be used to solve the measurement noise issue, due to the sensor concept, which requires a single PSD sensor to take 2 different measurements shortly after each other. A 100 kHz analog low-pass filter is therefore used. In experiments oversampling by a factor of 16 gave the best precision(3σ) of 1.5 µm. The stats of the complete system are combined in Table 1.

Although the precision of 0.2 µm is not fully maintained, it has still a very good performance considering the cost and size. Further optimization, other techniques and/or slightly different hardware, outlined in the recommendations, could easily improve the performance, without a significant increase in cost and size.

An extended abstract is added in Appendix A.

Table 1: Comparison of the performance specifications between the original and the new design

	Original	New
Translational bandwidth	60 Hz	25 Hz
Translational precision(3σ)	0.2 µm	1.5 µm
Program frequency	25 kHz	820 Hz
Total oversampling factor (for 100 Hz bandwidth)	125x	64x
Volume	20 dm^3	1 dm^3
Energy consumption, stationary (10% force)	3 W	0.03 W
Energy consumption, moving (100% force)	30 W	3 W
Cost	€20000	€700

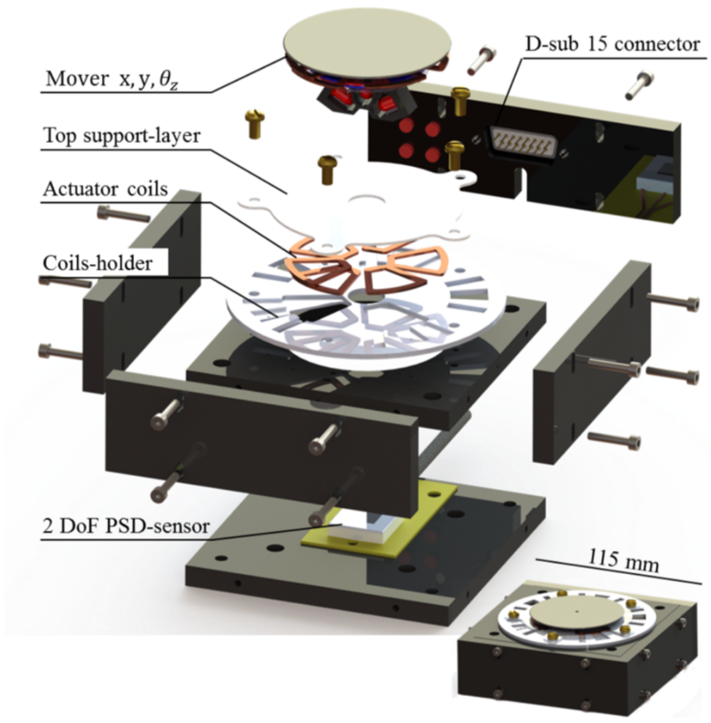


Figure 1: Exploded view demonstrator stage showing the mover, actuator coils, PSD and laser-cut (support) structures[22].

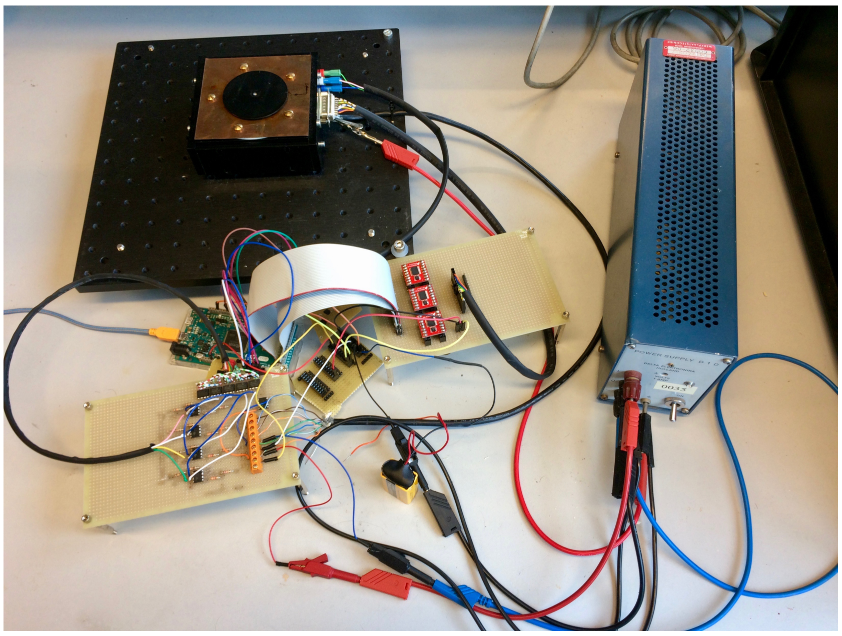


Figure 2: Visualization of the complete demonstrator stage. Printing the signal processing circuits on a PCB with proper shielding will reduce measurement noise and also reduce the size of the stage significantly.

Contents

1 Introduction

Fabrication and inspection in the micro/nano-scale are rapidly becoming more important in our lives. The miniaturization of electromechanical systems is the reason that portable devices, such as smartphones, can have so much functionality today and the inspection of biological samples helps with understanding and diagnosing diseases. Industrial fabrication and inspection in this scale is currently done by expensive and bulky systems, optimized for large quantities. Due to their cost and size, these systems are inaccessible for start-ups, individual consumers and 3rd world countries. Reducing the cost and size would open all sorts of possibilities in prototyping, research and home health monitoring.

A typical example is the blood smear test used to look for abnormalities in blood cells. Traditionally, trained laboratorians have examined blood smears manually using a microscope. More recently, automated systems have become available to help analyze blood smears more efficiently, see Figure 1.1[8]. However, these systems are still very expensive and big, which makes them only suitable for laboratories. Results of the test could be available much quicker if the tests could be done at a doctor's practice or at home, without the involvement of a laboratory. However, this requires the system to be much cheaper and smaller.

The positioning stage is part of what makes fabrication and inspection in the micro-scale so expensive. This thesis investigates possibilities to reduce the price and size of micrometer precision stages. This has to be done while maintaining as much precision as possible, while criteria as speed can be relaxed upon.

Figure 1.1: Automatic Hematology Imaging Analyzer[8]. Effective solution for automated differentiation of peripheral blood cells in large-sized laboratories by Vision Hema Ultimate

1.1. Literature study

A literature study is done to determine the essential parts used in state-of-the-art positioning stages. A decision can then be made about what can be reused and what has to be improved upon. This section is based on the more extensive literature study added in Appendix B.

1.1.1. The essential components of a stage of a positioning stage

Examination of many feedback-controlled positioning systems showed that all systems are build from the same essential components. All of the components are part of the motion control feedback loop, illustrated in the diagram in Figure 1.2. There is the mechanical structure that has to be positioned, the sensor to measure the position, the controller to calculate the appropriate control signal, an amplifier to amplify the control signal to a signal of sufficient power to drive the actuator, which then creates the forces on the mechanical structure. To interface between the digital controller and the analog physical world, analog-to-digital converters(ADC) and digital-to-analog converters (DAC) are required.

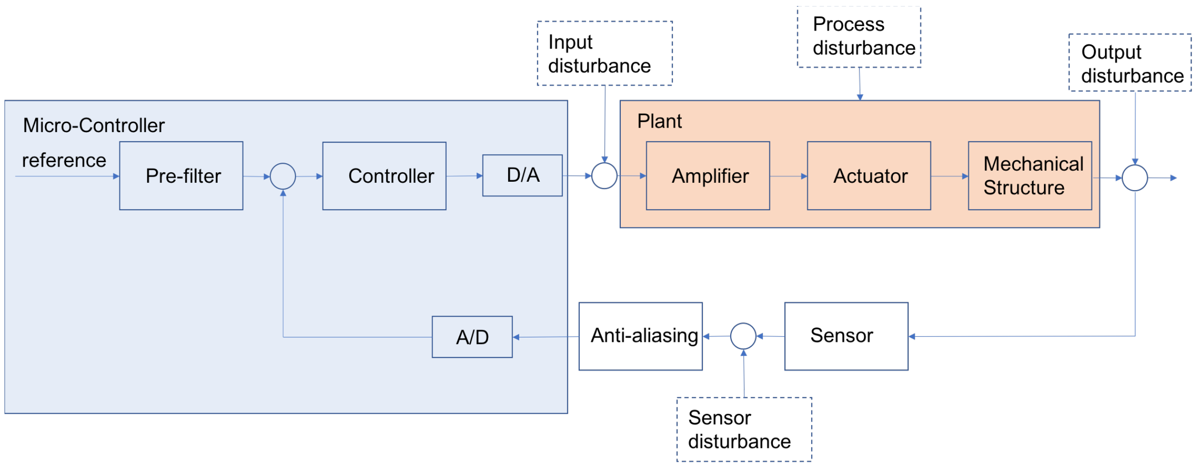


Figure 1.2: The feedback control loop of a positioning stage, showing the essential components and disturbances.

1.1.2. State of the art

The extensive literature study showed that most of the research focused primarily on performance. However, TU Delft's Mechatronic System Design (MSD) department, of which this research project is a part, already made a lot of progress in low-cost and compact precision stages. Four of the stages designed at the department together with the design of this project are broken down into their essential components in Figure 1.3.

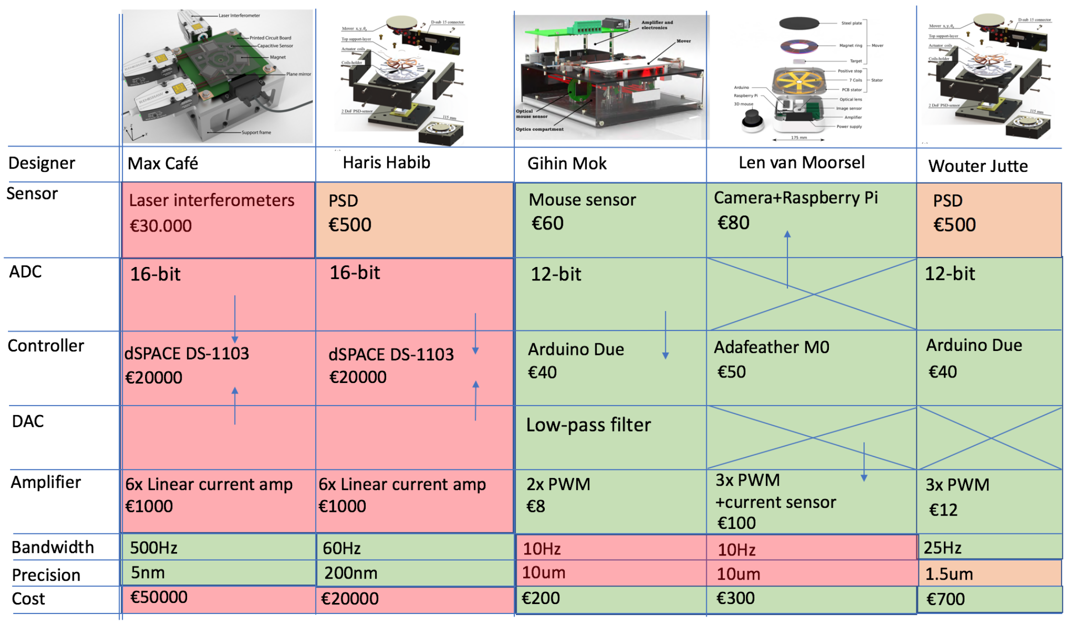
The stage designed by Max Café[19] shows the capabilities of a high cost, high performance stage. Haris Habib designed a stage based on a Position Sensitive Detector (PSD)[22], reducing the cost of the sensor system to only €500 and achieving a very good precision of 0.2 µm. For the other parts of the system, very high-quality components were used, leading to a still very expensive and bulky design. The stages designed by Gihin Mok[35] and Len van Moorsel[41] replaced not only the sensor systems, but all components by much cheaper alternatives leading to complete standalone systems of very low cost, both achieving 10 µm precision.

1.1.3. Research Focus and project objectives

Although very promising sensor systems have been proposed, none of them did an in-depth investigation into the capabilities and problems of low-cost alternatives for the micro-controller and the amplifier, which lead to the focus of this research:

Alternative micro-controllers and amplifiers for micrometer positioning stages, that are low-cost and compact.

The Arduino Due micro-controller(€40) and Pulse Width Modulatin (PWM) based amplifier(€6) specifically have been chosen to be investigated as replacements for the dSPACE DS-1103 system (€20000) and self-built linear current amplifiers (components only ~€1000) in Haris' design. This design is used as a starting point, because even though it is not the cheapest design, it is the only low-cost design that managed sub-micrometer precision. General insight into the capabilities and problems of low-cost micro-controllers and PWM is most important, so that better design choices can be made in future designs. However, an additional goal is to maintain as much of the 0.2 µm precision as possible, while replacing the expensive components of the original design.



Designer	Max Café	Haris Habib	Gihin Mok	Len van Moorsel	Wouter Jutte
Sensor	Laser interferometers €30.000	PSD €500	Mouse sensor €60	Camera+Raspberry Pi €80	PSD €500
ADC	16-bit	16-bit	12-bit		12-bit
Controller	dSPACE DS-1103 €20000	dSPACE DS-1103 €20000	Arduino Due €40	Adafeather M0 €50	Arduino Due €40
DAC			Low-pass filter		
Amplifier	6x Linear current amp €1000	6x Linear current amp €1000	2x PWM €8	3x PWM +current sensor €100	3x PWM €12
Bandwidth	500Hz	60Hz	10Hz	10Hz	25Hz
Precision	5nm	200nm	10um	10um	1.5um
Cost	€50000	€20000	€200	€300	€700

Figure 1.3: Five stages built by TU Delft's MSD group broken down into their essential components, cost and performance.

1.2. General Approach to analysing low-cost components

The approach of the analysis is done for all of the components in a very similar manner. Low-cost components usually have in common that some performance is lost, which introduces disturbances into the system, which transfer over to position errors. Examples are position measurements that deviate from the true position or output voltages/forces that are different from the intended output. Their effect on the position can in general be reduced in 3 ways. Either (1) the disturbance itself is reduced, (2) the system dynamics are designed to attenuate the disturbance or (3) the frequency range of the disturbance is changed to a frequency range that is attenuated by the dynamics of the system. Frequency spectrum analysis is a valuable tool in determining the viability of a component. Once the frequency characteristics of the disturbances are determined, system parameters can be optimized to attenuate their effect. Although the method is the same for all components, the characteristics of the disturbances and their transfer functions are very different and should be addressed separately.

1.3. Thesis overview

The Arduino Due and PWM based amplifiers will be analyzed and experimentally verified individually at first, after which they will be implemented and tested in a complete system. The topics will be presented in the following order:

- Chapter 2 – Preliminary work required to evaluate the Arduino and amplifiers will be addressed, such as the required bandwidth, transfer functions and system dynamics, all of them important for the analysis of the components done in later chapters.

- Chapter 3 – The Analog-to-Digital Converter (ADC). The importance of the ADC in data acquisition will be discussed, the accuracy of the ADC will be tested and techniques will be presented to improve the performance.

- Chapter 4 - The PWM based amplifier. The concept of the PWM amplifier will be described, the disadvantages that come with it and how to deal with them. The disturbances in signal reconstruction that are not specific to the PWM based amplifier will also be addressed.

- Chapter 5 – The micro-controller. The choice to take the Arduino Due is substantiated. The implementation of the control algorithm will be discussed and the processing speed of the algorithm will be tested.

- Chapter 6 – Implementation. The Arduino and the PWM based amplifier will be implemented into a complete system and the precision will be tested.
- Chapter 7 – Conclusions will be summarized.
- Chapter 8 - Recommendations will given for improvement of the stage and future work.

2 Preliminary work for performance analysis

The micro-controller and amplifiers are part of a whole system, the stage. To evaluate the applicability of the cost-effective Arduino Due and PWM based amplifiers in this mechatronic system, their performance has to conform with the precision goal of 0.2 µm. However, before the system and its components can be evaluated, another measure of performance is required. Frequency spectrum analysis is often used to evaluate dynamic systems, in which the bandwidth is the measure of performance. This chapter first derives the relation between the precision and the bandwidth for a simplified mass-spring system without feedback control and floor vibrations. The concept is then expanded to a feedback controlled system with control, where feedback control takes the place of the physical spring. The components used for feedback control introduce new disturbance sources, due to imperfections of the components. The effect of the imperfections on the performance of the stage can be quantified with transfer functions. The transfer functions are derived and contain the dynamics of all the individual components, from which the mechanical structure and the actuator are already known and can be expressed in the frequency domain. The bandwidth requirement and the transfer functions will be used in subsequent chapters to evaluated the Arduino Due and PWM based amplifier. In summary, these are the subjects addressed in this chapter:

- The required stiffness to meet the precision criterion without control.
- The relation between the stiffness and the bandwidth.
- An estimation of the required bandwidth.
- How the feedback control loop takes over the role of the spring.
- The transfer functions of the disturbances in the system.
- The system dynamics from the original design expressed in the frequency domain.

2.1. The required stiffness for a maximum allowable error in a precision system without control

The control bandwidth of a feedback-controlled precision system is closely related to it's precision performance. This can be much better explained with a much simpler mass-spring system, without any feedback control. A stage used for microscopy is used as an example, displayed in the schematic in Figure 2.1.

The microscope stands on a table that has floor vibrations n and the sample that has to be inspected lies on the mass m. If the mass is physically attached to the floor, depicted as the spring, the mass will follow the movement of the floor. Whenever the floor accelerates at a constant rate, the mass will also accelerate due to the spring force. The relative distance between the floor and the mass increases until the spring force accelerates the mass at an equal rate. The stiffer the spring, the smaller the distance between the microscope and the mass will be. The stiffness required to meet the 0.2 µm precision criterion can easily be calculated in this equilibrium scenario, where the acceleration of both the floor and the mass are equal. The equation of motion is:

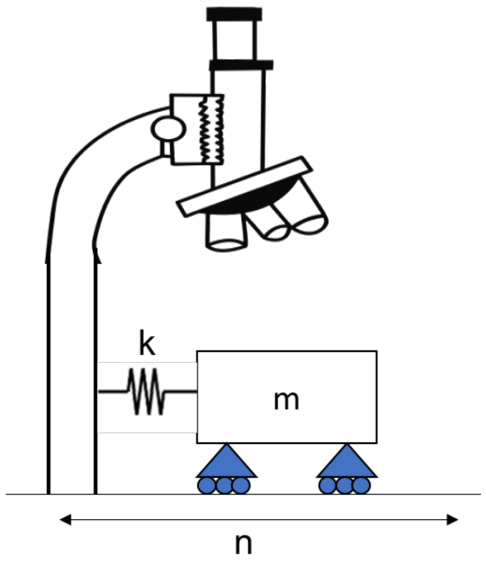


Figure 2.1: Schematic of a simplified precision system without control.

$$m\ddot{x} + kx = ku \tag{2.1}$$

$$m\ddot{x} = k(u - x) \tag{2.2}$$

, where $\ddot{x}$ is the acceleration of the mass, x the position of the mass, k the spring stiffness and u the position of the floor. The relative distance $\epsilon := u - x$, will not change when the acceleration of the mass is equal to the acceleration of the floor. The stiffness can then be calculated with:

$$k = \frac{ma}{\epsilon} \tag{2.3}$$

, where a is the acceleration of both the floor and the mass. An estimation of the maximum floor accelerations is obtained based on an extensive study[38] into quasi-steady floor vibrations in manufacturing plants in section G.1 and resulted in floor accelerations of:

$$a(3\sigma) = 58\,\mathrm{mm/s^2}$$

This result can be substituted together with the desired precision as the maximum allowable relative distance $\epsilon_{max} = 200\,\mathrm{nm}$, and the mass of the mover $m = 65\,\mathrm{g}$ to calculate the required stiffness.

2.2. The eigenfrequency and the bandwidth

The floor accelerations are not constant in reality, but much more dynamic than assumed in the last section. Frequency analysis is commonly used to understand and design dynamic systems. In frequency analysis the bandwidth is used to as a measure of performance, which is closely related to the the eigenfrequency of a damped mass-spring system. This can be shown easily, using the Fourier transform of the equations of motion (Equation 2.1), which gives:

$$\frac{X(j\omega)}{U(j\omega)} = \frac{k}{-m\omega^2 + k} \tag{2.4}$$

, where ω is the angular velocity. This transfer function allows evaluation of the compliance to floor vibrations on a frequency basis. The frequency response of a typical mass-spring system is shown in Figure 2.2. The frequency response is split into 2 regions by the eigenfrequency. The frequency range with a good response, which is about equal to the bandwidth and the rest of the frequency range which has an attenuated response. The bandwidth of the system can therefore be estimated with the eigenfrequency:

$$f_{bw} = \sqrt{\frac{k}{m}} = \frac{a}{\epsilon}\frac{\mathrm{rad}}{\mathrm{s}} = \frac{1}{2\pi}\sqrt{\frac{a}{\epsilon}}\mathrm{Hz} \tag{2.5}$$

Substitution of $a(3\sigma) = 58\,\text{mm/s}^2$, $m = 65\,\text{g}$ and $\epsilon = 0.2\,\mu\text{m}$ in Equation 2.5 gives an eigenfrequency $f_{bw} = 85\,\text{Hz}$. For both convenience and a safety margin, a bandwidth of 100 Hz will be used.

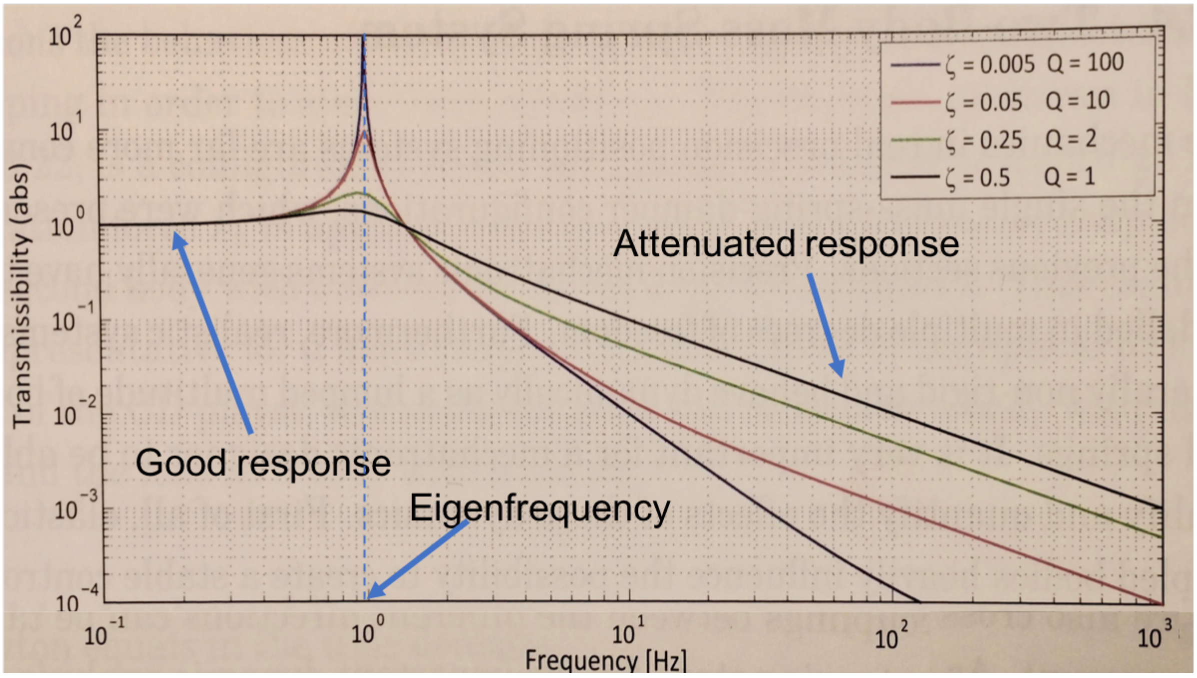


Figure 2.2: Magnitude plot of the transmissibility of a damped mass-spring system in response to external vibrations.[37]

2.3. Virtual stiffness and damping

So far, the example used a physical spring. In positioning stages that have to move, a physical attachment obviously doesn't work. A feedback loop with PID control can therefore be used to emulate the stiffness (and damping), which is depicted in Figure 2.3. The appropriate force is calculated by measuring the position of the stage relative to a reference. For the precision requirement to be met, the controller has to match the stiffness of the physical spring, leading to a system with a similar bandwidth.

Although a similar bandwidth puts the precision of the stage in the right order of magnitude, a similar bandwidth does not necessarily lead to the exact same precision. The reasons are that more disturbances are in the system and that the extra components can be used to change the dynamics of the system. For example, the controller can be much more complex than just emulating a physical system. Integrators can be used to reject low frequency disturbances and even though the bandwidth might be the same, the dynamics within the bandwidth are different. The 100 Hz requirement is also based on a scenario with constant acceleration, which in reality is not the case. The real precision/bandwidth relation is much more complex. It is however impossible to calculate the exact position error beforehand. Evaluating the system and components for 100 Hz will put the precision in the right order of magnitude, after which it can be iterated on.

2.4. The disturbance transfer functions

The disturbances due to the extra components enter the system at different places in the feedback control loop and therefore have their own transfer functions, which can be seen in Figure 2.4. Errors in the data acquisition enter as measurement disturbances m. This includes any disturbance due to quantization or other measurement noise. Process disturbances d are the combined disturbances in data processing, amplification and actuation. Floor vibrations enter the system as output disturbances n. The significance of the disturbances on the position y depend on their transfer functions.

The transfer functions of the different system inputs can be acquired using the diagram in Figure 2.4. The position y can be expressed in all their individual contributions combined (Equation 2.6). The sensitivity function (Equation 2.7) describes the system response to output disturbances. The complementary sensitivity (Equation 2.8) function describes the system response to both the reference signal and the measurement disturbances, assuming that no pre-filter is used. The transfer function for the process disturbance is the sensitivity function multiplied by the plant dynamics (Equation 2.9).

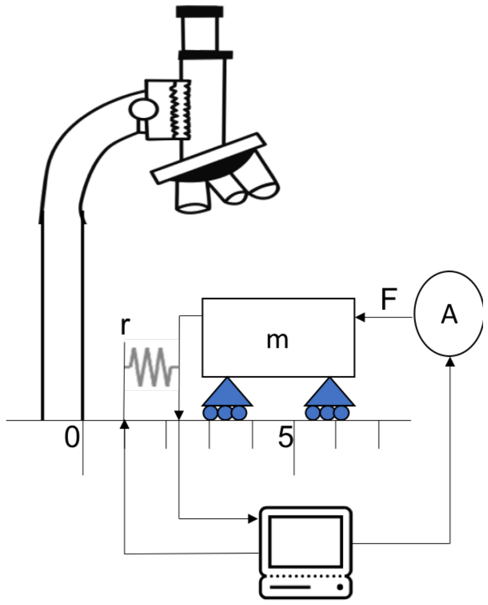


Figure 2.3: A schematic of a simplified precision system using virtual stiffness. The force is calculated by a micro controller, measuring the position of the stage relative to the reference r. The force is then generated by the actuator A.

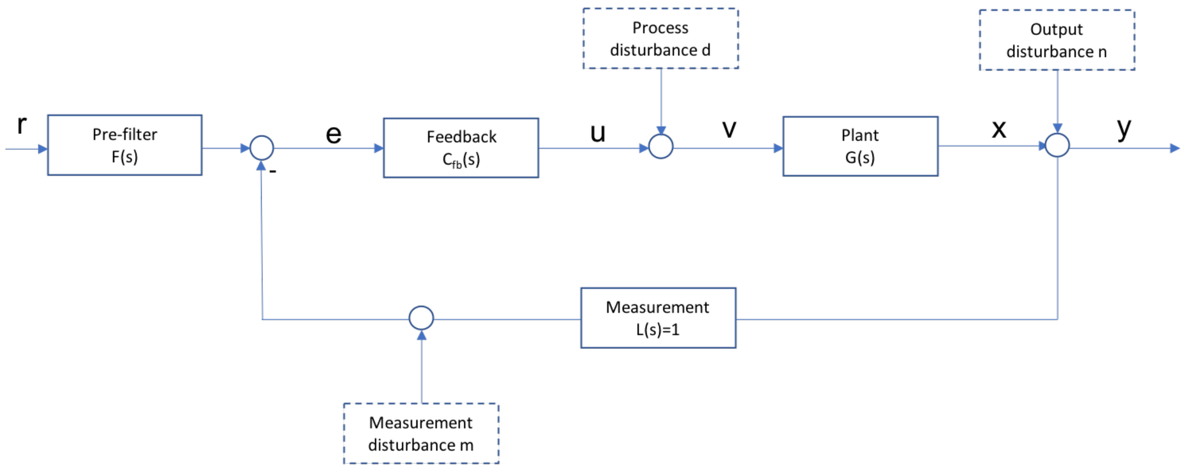


Figure 2.4: The feedback control loop of a positioning stage, containing the dynamics and the disturbances of the system.

$$Y(s) = \frac{GCF}{1+GC}r + \frac{G}{1+GC}d + \frac{1}{1+GC}n - \frac{GC}{1+GC}m \tag{2.6}$$

$$\text{Sensitivity function } S(s) = \frac{1}{1+GC} \tag{2.7}$$

$$\text{Complementary sensitivity function } T(s) = \frac{GC}{1+GC} \tag{2.8}$$

$$\text{Process disturbance transfer function } H(s) = \frac{G}{1+GC} \tag{2.9}$$

2.5. System dynamics

The transfer functions (Equation 2.7-2.9) are all functions of the plant dynamics G(s) and the controller dynamics C(s). Not all the dynamics are known from the start of the design process. The mechanical structure and actuator are the only components that are known from the start, because they are taken from the original design. The dynamics of these parts can be expressed in the frequency domain and will be used to evaluate design decisions in subsequent chapters.

2.5.1. Mechanical dynamics

The mechanical structure consists of the mover and the ferrofluid bearing (see system overview in Appendix C). Ferrofluid bearings are absent of stick-slip behavior[43], resulting in negligible stiffness. The mechanical structure is therefore a mass-damper system. Ferrofluid will, however, result in a nonlinear damping coefficient that depends on the thickness of the fluid layer[29]. Movement of the stage will leave behind some fluid on the surface over which it's moving. This will change the thickness of the fluid and therefore change the damping coefficient. Exact modeling is therefore difficult and a simplified linear estimation is used. The transfer functions of mass-damper systems for translation and rotation are given below:
Translation:

$$G_x = \frac{x}{F} = \frac{1}{ms^2 + cs} \tag{2.10}$$

Rotation:

$$G_z = \frac{\omega}{T} = \frac{1}{Is^2 + c_{rot}s} \tag{2.11}$$

The mass of the mover is weighed to be 65 g with a scale. The Inertia is estimated, with the assumption that all the mass is in the magnets, because the magnets are by far the heaviest part. This assumption gives an Inertia of $I = 2.19 \times 10^{-5}\,\mathrm{kgm}^2$. The translational and rotational damping coefficient are calculated in Appendix E. The translational and rotational damping coefficients are $c = 18\,\mathrm{N\,s\,m^{-1}}$ and $c_{rot} = 1.7 \times 10^{-3}\,\mathrm{N\,m\,s\,rad^{-1}}$ respectively.

2.5.2. Actuator dynamics

The Lorenz actuator can be modelled as a simple resistor-inductor circuit, displayed in the diagram in Figure 2.5.

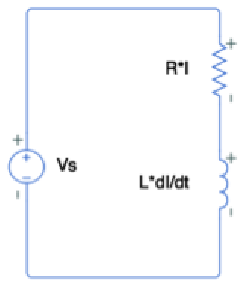


Figure 2.5: Electric circuit diagram of a Lorenz actuator.

The transfer function can be obtained by setting up the differential equation and using the La place transform. The differential equation for such a simple LR-circuit can be set up using Kirchhoff's circuit laws:

$$V_s(t) = I(t)R + L\frac{\mathrm{d}I(t)}{\mathrm{d}t} \tag{2.12}$$

Using the Laplace transform and rewriting gives the transfer function:

$$\frac{I(s)}{V_s(s)} = \frac{1}{sL + R} = \frac{\frac{1}{R}}{s\frac{L}{R} + 1} \tag{2.13}$$

Note that this is identical to the standard form of a first order low-pass filter with a cut-off frequency of:

$$\omega_0 = \frac{1}{\tau_\epsilon} = \frac{R}{L} \tag{2.14}$$

The resistance and inductance of the coils are measured with an LCR meter and are $R = 3.2\,\Omega$ and $L = 8080\,\mu\text{H}$ respectively, which results in a cutoff frequency of $\omega_0 = 40\,000\,\text{rad}\,\text{s}^{-1}$ or $f = 6400\,\text{Hz}$.

3 The Analog-to-Digital Converter

Although all components are equally important in a well designed high-precision stage, the data acquisition is a special case. The transfer function of both the reference and the measurement disturbance is the same complementary sensitivity function (Equation 2.8). Precision stages are required to have a high gain (*GC* in Equation 2.8), for good disturbance rejection and reference tracking. The problem with a high gain is that the compliance to measurement errors also increases. The contribution of the measurement disturbance to the position is the only disturbance that gets worse if the gain is increased. Having a good data acquisition system is therefore particularly important. Haris' work already showed good performance of the PSD sensor achieving a precision of 0.2 µm. However, just as important in the data acquistion as the sensor is the Analog-to-Digital Converter (ADC), which is the topic of this chapter. This chapter investigates the viability of the 12-bit ADC on the Arduino Due board in sub-micrometre positioning stages. The following topics will be discussed regarding the ADC:

- The required number of ADC bits to meet the precision criterion.
- The relation between the Signal-to-Noise Ratio(SNR) and the effective number of bits (ENOB).
- A test for the SNR and the equivalent ENOB.
- A technique to increase the SNR of the Arduino ADC, using oversampling and filtering.
- The limitations of this technique and how this can be improved upon.
- The testing results of oversampling and filtering, using the Arduino ADC

3.1. The required number of effective ADC bits

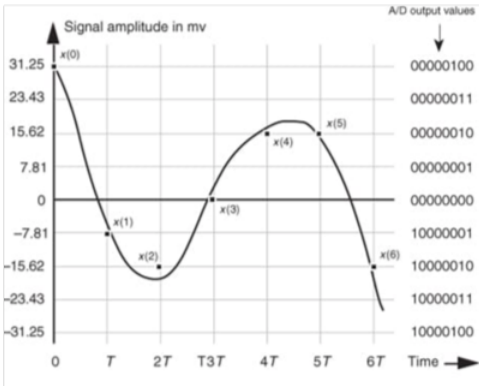
The accuracy of ADC's is often expressed in the length of the output word or the number of bits. It will become obvious later in the chapter that the SNR is the proper representation for the real accuracy and not the number of bits. However, the SNR of an ADC can be converted to the Effective Number Of Bits(ENOB), which is the number of bits that an ideal ADC with an equivalent SNR would have. An approximation of the required number of bits of an ideal ADC is done first, so that later we can later relate this to the SNR of the real ADC. The required number of bits of an ideal ADC can be obtained with a simple derivation, with the use of a few assumptions. In the approximation the following assumptions are made:

1. The ADC is ideal, which means no temperature drift, perfect linearity and the maximum error is a rounding error of half the least significant bit (lsb).
2. The disturbance due to quantization is the only disturbance in the system.
3. The complementary sensitivity function has an 'ideal' value of 1, which means that the measurement error transfers over perfectly, resulting in a position error of the same magnitude.

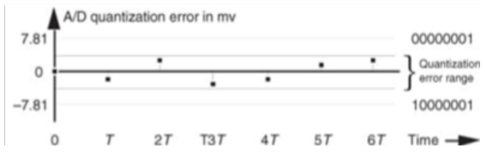
The required number of bits can be obtained by deriving an expression that shows how the quantization error propagates to a measurement error in position. This is done by partial differentiation of the equation that calculates the position from the measured sensor output voltages. This approximation shows a requirement of 16 bits for a maximum position error of 0.2 μm. The full derivation is added as Appendix D. The 12-bit Arduino ADC obviously doesn't meet the 16 bit requirement. Techniques to increase the performance of an ADC will be presented later in this chapter, but first the relation between the SNR and the number of bits of an ideal ADC is discussed.

3.2. The relation between the SNR and the effective number of bits

The real accuracy of an ADC is represented by the SNR or the effective number of bits (ENOB), because the number of bits is insufficient to quantify the real accuracy of an ADC. Although the number of bits is sufficient to quantify the performance of an ideal ADC, it only says something about the number of possible output words and this doesn't include errors due to nonlinearities in real ADC's, like broken bits or imperfect rounding. The number of bits and the SNR are related however, which can be easily understood when a converted digital signal is viewed at as the sampled version of the actual signal superposed with an error signal due to quantization. This is illustrated in Figure 3.1, where (a) shows the analog signal and the resulting samples and (b) the resulting rounding errors.



(a) Digitized x(n) values of a continuous signal



(b) Quantization error between the actual analog signal values and the digitized signal values

Figure 3.1: Quantization errors[33]

This quantization error signal can also be expressed in the frequency domain, just like any other signal. Reduction of the error signal in the time domain leads to a reduction of the quantization noise in the frequency domain and therefore increases the SNR. An equation (Equation 3.1) for the number of bits of an ideal b-bit ADC can be derived in terms of the SNR. The complete derivation of Equation 3.1 is quite extensive and is part of Appendix F. Testing the SNR of an ADC and substituting it into the equation gives the number of bits that an ideal ADC with equivalent SNR would have, represented by the ENOB.

$$ENOB = \frac{SNR - 1.76}{6.02} \tag{3.1}$$

The previous section showed a requirement of a 16-bit ideal ADC. The ADC should therefore have an ENOB of 16, which is equivalent to a SNR of ~ 100dB.

3.3. Testing the SNR of the Arduino ADC

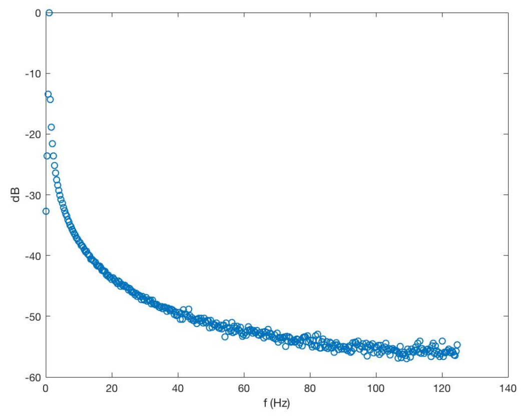
The SNR of an ADC can be tested, by applying a sinusoidal analog voltage to the input of the ADC and analyzing the spectrum of the ADC's output samples. This voltage signal is generated by a signal generator which gives a very clean signal with minimal noise. The signal is then sampled with the Arduino Due after which the data is processed on a PC. The Fast Fourier Transform(FFT) is used on the data after which the SNR can be obtained. The following subsection discusses the choice of the FFT size, sampling frequency and signal frequency so that spectral leakage can be avoided. The subsection after that discusses 2 methods to obtain the SNR from the FFT.

3.3.1. Avoiding spectral leakage

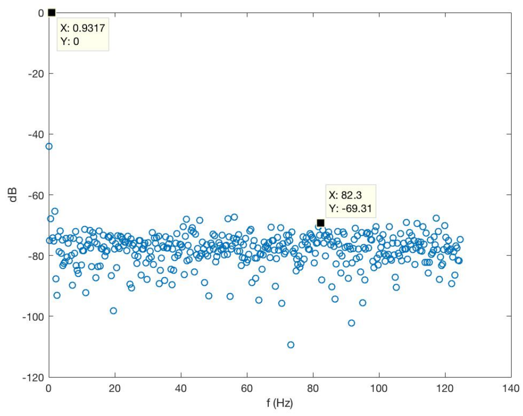
Spectral leakage is a phenomenon that occurs when the number of signal periods included in the Fast Fourier Transform is not an integer number. This has to be avoided to improve the efficiency of the spectral measurements, so that a finite sample size suffices[33]. Spectral leakage can be easily avoided by satisfying the following equation, which makes sure an integer number of signal periods is included in the FFT:

$$f_{signal} = \frac{m * f_s}{N} \tag{3.2}$$

With f_{signal} the input frequency, m the integer number of sinusoidal periods included in the FFT, f_s the sampling frequency and N the FFT size. Spectral leakage is easily identified in the FFT, due to its curved characteristic, shown in Figure 3.2a. The frequency spectrum of a signal sampled by the Arduino Due, where spectral leakage is avoided is shown in Figure 3.2b.



(a) The Fourier transform of a 1 Hz input signal sampled by the Arduino Due with spectral leakage



(b) The Fourier transform of a 1Hz input signal sampled by the Arduino Due with minimal spectral leakage.

Figure 3.2: The Fourier transform of a 1 Hz input signal sampled by the Arduino Due.

3.3.2. The SNR of the Arduino ADC

There are 2 methods to obtain the SNR from the FFT. The more accurate signal-to-noise-and-distortion (SINAD) and a method based on visual an approximation, which are both described below in their own paragraph. The SINAD method showed that the Arduino ADC has a SNR of 50 dB, which is equivalent to only 8 ENOB. The visual approximation method shows an even worse SNR of 44 dB or 7 ENOB. Although the visual approximation is less accurate, it is a very convenient tool to compare performances, which will prove useful in later sections. Both methods showed that the true accuracy is much lower than the 12-bit specification. To meet the 16-bit requirement, the ADC requires an 50 dB increase in SNR. A technique to increase the SNR of an ADC is described in the next section. How the SNR can be obtained from both methods is described below.

The SNR with the SINAD method The first method is the most accurate technique that characterizes an ADC's true SNR (including nonlinearities) and measures what is commonly called the signal-to-noise-and-distortion (SINAD) and does. The SINAD value is calculated from the spectral power samples with Equation 3.3.

$$SINAD = 10\log_{10}\left(\frac{\text{Total signal power}}{\text{Total noise power}}\right), \text{ in dB} \tag{3.3}$$

The total signal power is obtained by summing all signal-only spectral power samples in the positive-frequency range. The total noise power is obtained by summing all noise-only spectral power samples in the positive frequency range. The 0 Hz sample is ignored. The result is a SINAD of 50 dB for the Arduino ADC.

The SNR with the visual approximation The second is a visual approximation, which uses the difference between the signal and the noise floor from Figure 3.2b and then compensates for the processing gain. The processing gain is the result of the difference between coherent addition of the input signal and incoherent addition of the noise[4]. The SNR increases therefore with the FFT size. The processing gain can be calculated with Equation 3.4.

$$\text{FFT processing gain} = 10\log_{10}(\frac{N}{2}), \text{ in dB} \tag{3.4}$$

The processing gain for a FFT size of $N = 805$, which is used in Figure 3.2b, is 26 dB. Compensating the 70 dB SNR obtained from Figure 3.2b, results in an actual ADC SNR of 44 dB. This underestimates the accuracy of the ADC compared to the SINAD method.

3.4. Increasing the SNR with oversampling and filtering

The quantization noise of the ADC can be reduced by oversampling and filtering the analog signal. The process is pretty straightforward. The analog signal is sampled at a sample rate f_s, that is higher than the minimum rate needed to satisfy the Nyquist criterion and is then digitally low-pass filtered. It has already been established that the error signal, as a consequence of quantization, can be expressed in its spectral components, just like any other signal. Spectral content outside of the frequency range of interest can then also be filtered. The theory behind oversampling assumes that the quantization noise is white, which from Figure 3.2b appears to be true. However, some criteria have to be met for this to be true, which directly affect the limitations of oversampling and filtering. The limitations will be dealt with later, but for now quantization noise is assumed to be white. If the noise is white, its variance (statistical equivalent to power) can be calculated with Equation 3.5, see Appendix F for derivation.

$$\text{total quantization noise power} = \sigma^2 = \frac{lsb^2}{12} \tag{3.5}$$

Noise that is truly white, has an equal amplitude across all frequencies, reaching far beyond the Nyquist frequency. However, all the noise power will end up in the spectrum within the Nyquist frequency, due to aliasing. This leads to the full white noise power being evenly distributed in the region of $\pm\frac{f_s}{2}$, depicted in Figure 3.3.

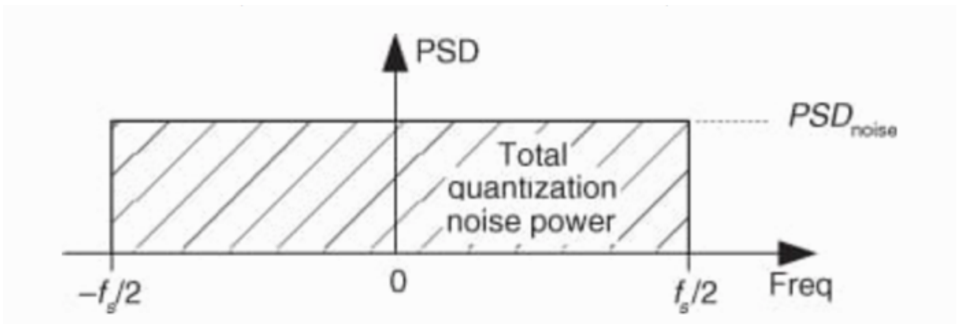


Figure 3.3: The power spectral density of the quantization noise power, if the quantization noise is white. The noise power will be evenly distributed between ± the Nyquist frequency[33].

By increasing the sampling frequency and therefore the Nyquist frequency, the total quantization noise will be spread over a larger frequency range. This in itself will not reduce the quantization noise, it will only spread it out over a larger frequency range, shown in Figure 3.4(a) and (b). However, when the quantization noise is spread over a frequency range that reaches beyond the range of the signals of interest (bandwidth), the noise can be lowpass filtered in such a way that the signals of interest are not affected, while the noise in the frequency range beyond the bandwidth will be attenuated. The quantization noise outside of the dashed line in Figure 3.4b will therefore be attenuated and thereby reduces the total quantization noise. After filtering, the digital signal can be downsampled back to twice the Nyquist frequency, if required. It's important that the downsampled signal uses more bits to represent the samples, otherwise another quantization error will be introduced, which negates the whole process.

Increasing the sampling frequency by a factor 2 and then filtering everything beyond the old Nyquist frequency perfectly will reduce the quantization noise power by half, which equals −3 dB. Equation 3.1 states that increasing the ENOB by 1, requires a noise reduction of 6 dB and thus requires oversampling by a factor

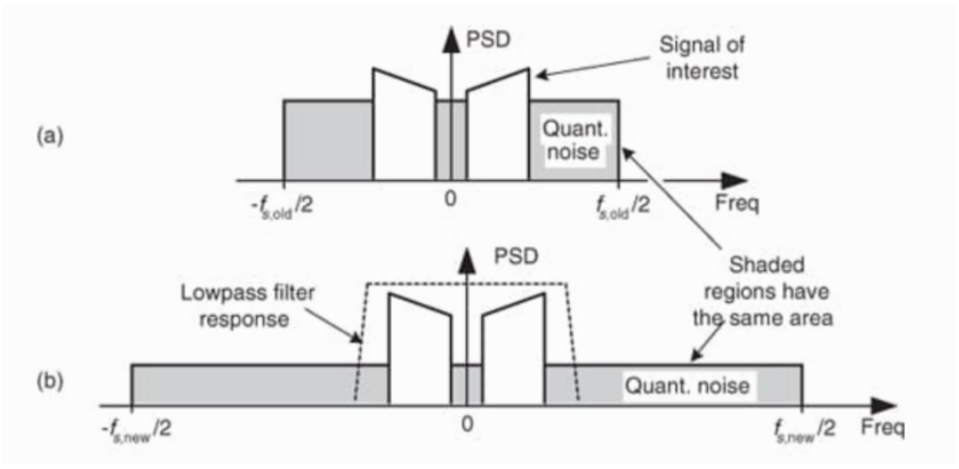


Figure 3.4: The effect of oversampling and filtering[33]. (a) Noise PSD at an $f_{s,old}$ sample frequency. (b) Noise PSD at a higher $f_{s,new}$ sample frequency. Increasing the sampling frequency increases the amount of quantization noise that can be low-pass filtered.

of 4. Oversampling by another factor of 4 would reduce the noise by another 6 dB. This relation between the number of extra bits (w) and the required oversampling frequency (f_{os}) is expressed in Equation 3.6.

$$f_{os} = 4^{w} * f_s \tag{3.6}$$

3.5. Limitations of oversampling and filtering

The limitations of oversampling and filtering are directly related to the two assumptions made. The assumptions are:

1. The quantization noise appears as white noise in the frequency spectrum
2. Filtering is perfect.

3.5.1. The quantization noise requires a white spectrum

Whenever the quantization noise doesn't appear as white noise, but ends up in the frequency range of interest instead, it can't be filtered away. Therefore, it's important to understand what makes the quantization noise white, so that this can be avoided. The first step in understanding the relation between the quantization error and the noise spectrum is to establish what white noise is in the discrete time domain.

"In discrete time, white noise is a discrete signal whose samples are regarded as a sequence of serially uncorrelated random variables with zero mean and finite variance"[14].

So, if those three criteria are met for the quantization error in the discrete time domain, the quantization noise is white.

The "**finite variance**" criterion is always met for the quantization error, because the error is always finite.

The "**serially uncorrelated random variables**" criterion is met if the quantization error is uncorrelated with the continuous input signal. If we were to digitize a single continuous sinewave whose frequency was harmonically related to the ADC's sample rate, we'd end up sampling the same input voltages repeatedly and the quantization error sequence would not be random. The quantization error would be predictable and repetitive. This can be shown with the extreme example of a constant signal, which is depicted in Figure 3.5. Figure 3.5 (a) shows a constant signal, which is rounded down to the output word that represents the analog signal the best. In (b) the error is displayed, which shows the underestimation that is created by rounding downwards. The FFT would therefore result in a 0 Hz component that under-represents the real signal. If we would think of the sampled signal as the real signal superposed with an error signal, the error signal has the exact same 0 Hz frequency as the signal of interest and can therefore not be filtered out.

In practice, complicated continuous signals such as music or speech, with their rich spectral content avoid this problem. However, in precision positioning where minimal movement is the goal, extra caution has to be taken. The correlation criterion might in this case depend on the amount of noise on the analog signal. If the quantization error isn't random, such as in the example, noise can be added voluntarily to make it random. This technique is called dithering.

The "**zero mean**" criterion needs an equal probability for the error to be rounded up or down, of which the effect can also be shown with simple examples. In Figure 3.5, the probability isn't equal. All errors are rounded down. When dithering is used and even the slightest white noise is added to the signal in Figure 3.5, it has an almost equal probability to be rounded up or down. When an infinite number of samples were to be taken and averaged, the value would converge to the middle of the 2 values, which is the true value of the signal.

However, when the signal isn't located perfectly between two values it doesn't work out as well. For example, when the mean of the signal is at 3.25 and white noise is applied with an amplitude of 1.25 bits of amplitude. Then $\frac{0.5}{2.5}$ of the range is rounded down to 2, $\frac{1}{2.5}$ is rounded to 3 and $\frac{1}{2.5}$ is rounded to 4. If an infinite number of values would then be averaged, it converges to $\frac{1}{5}*2+\frac{2}{5}*3+\frac{2}{5}*4=3.2$, which is not exactly 3.25. Increasing the amplitude of the noise makes the probability more equal and increases the possible accuracy that can be resolved from the signal by oversampling. The larger the amplitude the more precision that can be obtained.

Although dithering can help with the correlation and zero mean criterion and therefore increase the possible accuracy of an ADC, the following things have to be kept in mind:

1. The extra noise power that is added to the signal adds to the signal's noise. In general, the gain in randomization and zero mean is worth a small amount of noise. However, if too much white noise is added, the extra amount of oversampling required to reduce the noise might not be worth it. There is a technique that avoids this problem, called noise shaping. Noise with a frequency spectrum that is completely outside of the frequency range of interest, instead of white noise, can be added to the signal, so that it can be filtered out completely without extra oversampling. This technique is not implemented however.

2. Another thing to look out for when dithering is to make sure that the noise doesn't clip the ADC.

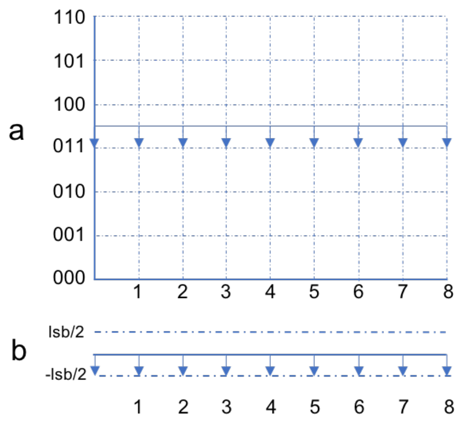


Figure 3.5: The quantization error for a DC signal. (a) Shows the analog signal (b) shows the resulting error due to quantization.

3.5.2. The low-pass filter used to filter the over-sampled data

So far, the filter was assumed to be a perfect rectangle, with a transfer function of 1 within the bandwidth of the system and 0 outside of it. In reality perfect filters do not exist and filter design plays an important role in the accuracy that can be gained with oversampling and filtering. For example, if the filter has 20 dB attenuation in the stop band, the quantization noise can't be attenuated by more than 20 dB. In general, better digital filters require more samples to be processed to get a steeper cutoff and more attenuation. This increases the latency, which influences the stability and other performance factors of the system.

3.6. Testing oversampling and filtering with the Arduino ADC

The theory of the technique to increase the SNR is now validated with experiments. Because optimal filter design with all of its trade-offs is a topic on its own and outside of the scope of this project, the Moving Average (MA) filter will be used to show the behavior and limitations of oversampling and filtering. The frequency response of the MA filter is shown in Figure 3.6 for several numbers of averaged samples M. It can be seen from the figure that the MA filter has the shape of a sinc function, instead of the ideal rectangle. The accuracy that will be gained by oversampling and filtering will therefore differ from Equation 3.6. The filter is implemented in the Arduino and the SNR will be tested with the same method as described in section 3.3.

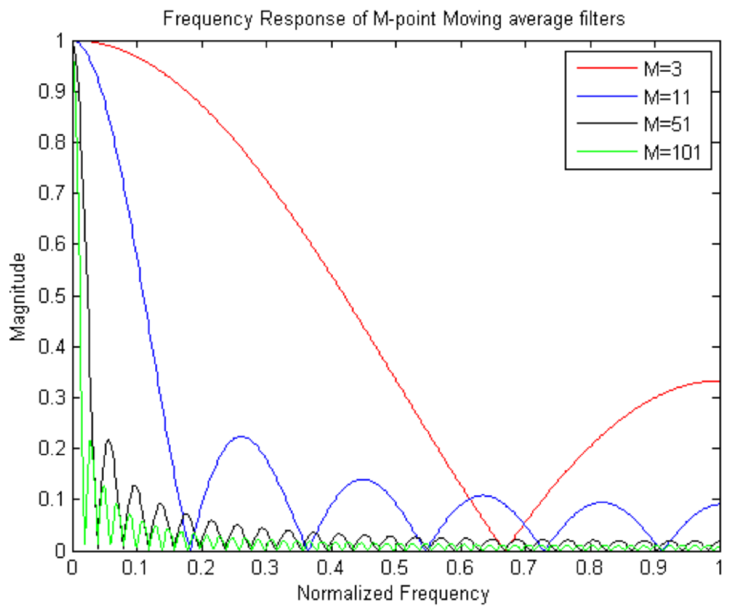


Figure 3.6: Frequency Response of Moving Average Filters of various lengths, normalized by the Nyquist frequency. A normalized frequency of 1 is therefore equal to the Nyquist frequency.[3]

3.6.1. Testing oversampling and filtering with a clean voltage signal from a signal generator

First a clean 1 Hz voltage signal from the signal generator is sampled with the Arduino ADC at a frequency of 250 Hz. Both the sampling frequency and the number of samples averaged are then increased by factors of 4, because this should theoretically result in about 1 ENOB (Equation 3.6). The results for different oversampling rates are combined in the first three rows of Table 3.1. The table shows that the theoretical increase in SNR is never achieved, but does show a considerable improvement. The diminishing returns can be explained by the clean and low frequency signal of the signal generator. The quantization errors aren't completely random and the noise spectrum is therefore not really white.

In the fourth row of the table Direct Memory Access (DMA) is used to reduce the sampling time to a negligible amount by writing measurement directly to memory, which will be discussed more in depth in chapter 5. This also increases the sampling frequency to 667 kHz. Sampling and averaging at this frequency showed no improvement at all compared to the accuracy without oversampling, because the sampling frequency is so high that the extra samples are for the most part identical and averaging leads to the same result.

3.6.2. Testing oversampling and filtering with the noisy signal from the PSD sensor

If the sensor signal would be as clean as the one from the signal generator, a noise generator should be used to improve randomization of the quantization error. However, this is not necessary for the PSD sensor that will be used, because it already has enough noise. The effects of oversampling and filtering on a noisy signal can therefore be tested with the PSD sensor output. Instead of using a 1 Hz signal from the signal generator, the output of the sensor is used when the stage is kept in the same position, leading to a constant 0 Hz signal. This time DMA is used in all cases. The FFT results of the oversampled and filtered data are shown in Figure 3.7. It shows a consistent improvement of 5 dB every time oversampling is increased by a factor of 4 and shows no

Table 3.1: The SINAD and ENOB of the Arduino ADC with oversampling and averaging a 1 Hz signal generator signal. The first three rows show the results when the sampling frequency of 250 Hz is increased by factors of 4. The last row shows the result when DMA is used with a sampling frequency of 667 kHz.

oversampling factor	SINAD(dB)	ENOB
1	50	8.0
4	55	8.8
16	58	9.3
256 (with DMA)	50	8.0

diminishing returns yet. The noise is however not solely quantization noise anymore and also includes other noise sources. This can easily be seen from the 50 Hz peak, which is the frequency of the power grid. The SNR is therefore worse than with quantization noise alone. The quantization noise is however reduced by the same 5 dB, assuming that the added noise made the quantization noise white.

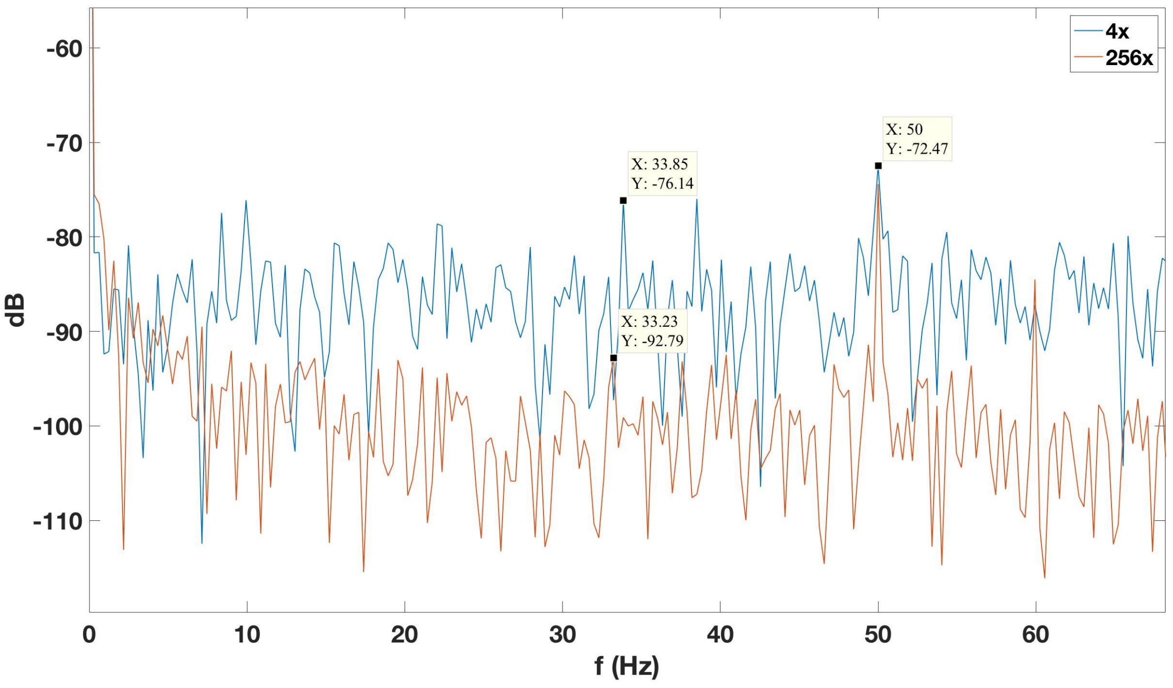


Figure 3.7: The Fourier transforms of the PSD sensor output for two different oversampling factors, when the position is passively kept constant. The noise floor is decreased by about 5 dB for every oversampling factor of 4.

4 The PWM based amplifier

The purpose of amplifiers is to amplify the control signals of the micro-controller. The micro-controller puts out low power control signals, which are unable to drive the actuator. When designing high precision stages actuated by Lorenz forces, the natural choice is to use linear current amplifiers. Linear current amplifiers are known for their high accuracy and very fast settling times. However, when portability and cost are of significance, linear current amplifiers have several disadvantages. Linear current amplifiers are very energy inefficient, dissipating a large portion of their power as heat. This requires bulky heat sinks to keep the temperature low, which make the amplifiers large. Another disadvantage is the requirement of DACs and high-power op-amps, which are relatively expensive for low-cost applications. An alternative solution is therefore explored. Amplifiers based on Pulse Width Modulation (PWM) are characterized by their very good efficiency, compact size and light weight. The difference in size is shown in Figure 4.1. This comes not without its drawbacks, which will be addressed in this chapter. The following topics will be discussed:

- The PWM based amplifier concept.
- The implementation of the PWM based amplifier to avoid audible noise and attenuate the higher harmonics of the pulse wave.
- The consequences of the amplifier acting as a voltage source.
- The relation between signal reconstruction and the program frequency.

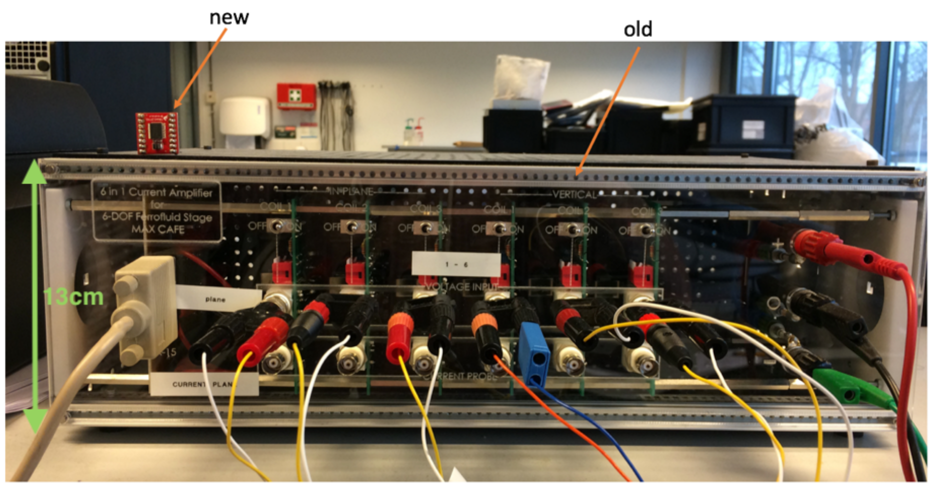


Figure 4.1: Size difference between the original and the new amplifier concept. The amplifier module from the original design contains 6 linear power amplifiers. The new design requires 3 of the PWM modules (TB6612FNG) shown.

4.1. The PWM based amplifier concept

Amplifiers based on PWM regulate the average voltage supplied to the load by switching the power supply on and off. A PWM based amplifier requires an H-Bridge, an external power supply and a micro-controller. The schematic of an H-Bridge is illustrated in Figure 4.2(a), where the actuator is portrayed as M and the power supply as V. By controlling the 4 transistors, depicted as the switches, the direction in which the current flows through the actuator can be controlled, (b) and (c). PWM is then used to regulate the percentage of time in which the power supply is connected to the actuator, by opening and closing the circuit. This results in a pulse wave voltage signal, illustrated in Figure 4.3. The percentage of "on time", called the duty cycle, determines the average voltage of the signal, which is the voltage of interest.

However, the switching comes with extra harmonic content, which can be observed when looking at the pulse signal in the frequency domain in Figure 4.3. The 0 Hz component is the desired average voltage, but the other frequency content is undesired, which will be called PWM noise. Two problems arise from this PWM noise. The first problem is that if the harmonics are in the audible frequency range, it will result in audible noise. The other problem is that these harmonics contribute to the position error of the system. For the PWM based amplifier to be a viable option, it has to be made sure that these errors are small enough. Both of the problems will be discussed in the next section.

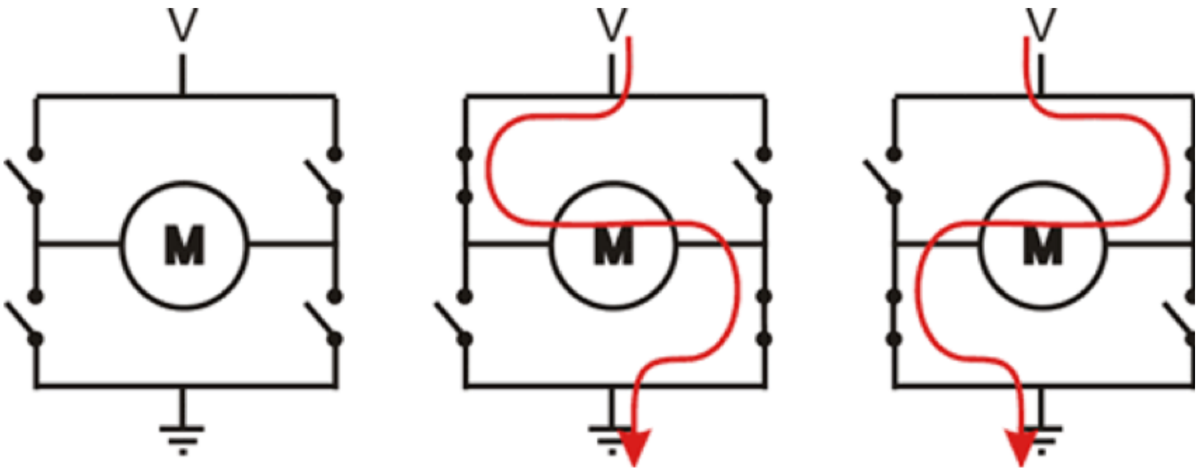
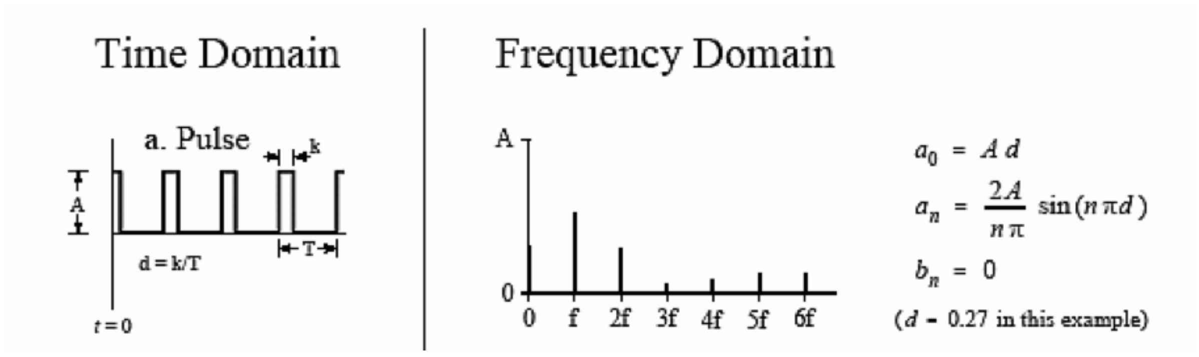


Figure 4.2: The three modes of operation of an H-bridge[15].



$$a_0 = A\,d$$

$$a_n = \frac{2A}{n\pi}\sin(n\pi d)$$

$$b_n = 0$$

(d - 0.27 in this example)

Figure 4.3: The pulse wave in the time and frequency domain. The equations can be used to calculate the spectral components of the pulse wave. The parameters are indicated as follows: A is the amplitude of the pulse wave, d the duty cycle of the pulse wave, and n the multiple of the fundamental frequency.[16]

4.2. PWM implementation

4.2.1. PWM frequency

Audible noise is easily avoided by setting the PWM frequency at a higher frequency than the human ear can hear, which is about 20 kHz. Important to note is that the fundamental frequency of a pulse wave is the frequency of the pulse wave itself and that all other harmonics are at higher frequencies. The lowest frequency can therefore be controlled by setting the frequency of the PWM module in the Arduino registers. The maximum PWM frequency is dependent on the clock frequency, the output resolution and the PWM mode.

4.2.2. PWM modes

There are two PWM modes, fast PWM and phase-correct PWM. The difference between the 2 modes is that fast PWM can have either twice the resolution or twice the frequency compared to phase-correct PWM, but whenever the duty cycle of fast PWM is changed, the phase of the PWM signal is changed as well. To understand why this is important, further examination is required for time varying signals. So far, we've considered the duty cycle independent of time. This is not the case if the stage has to be controlled and a change in force is required. To illustrate this, let's say that instead of a constant pulse wave, a pulse wave with a varying duty cycle of $d(t) = sin(2\pi f_{duty} t)$ is required. The effect is modulation of the harmonic content, which can easily be observed by substituting d into the formulas in Figure 4.3. Substitution in the a_0 band gives $a_0 = Asin(2\pi f_{duty} t)$, which shows that the band is amplitude modulated and instead of a non-varying 0 Hz band, a spectral band with an amplitude of A appears at the frequency f_{duty}, which is the desired amplitude at the desired frequency. Substitution of d in the equation for a_n shows that the undesired spectral harmonics are frequency modulated, instead of amplitude modulated. This will also lead to a change in spectral content. However, the undesired higher harmonic content will be discussed later in this chapter. So far, the change in duty cycle still gives the desired spectral content in the low frequency range. However, when the phase of the duty cycle is also a function of time, this is not the case. Rewriting to $d(t) = sin(2\pi f_{duty} t + \phi(t))$ and substitution gives $a_0 = A * sin(2\pi f_{duty} t + \phi(t))$. This shows that the phase also modulates the amplitude, which is undesired. This should be avoided and therefore phase-correct PWM is used.

4.2.3. The output resolution

Whenever phase-correct PWM is used, the number of output values that can be set is calculated with:

$$\#_{output\_values} = \frac{f_{clock}}{2 * f_{pwm}} = \frac{84\text{MHz}}{2 * 42\text{kHz}} = 1000 \tag{4.1}$$

The clock frequency of the Arduino Due is 84 MHz and the PWM frequency is set to be 42 kHz, to stay clear from the audible frequency range. The result is 1000 possible output values, equivalent to an output resolution of ~ 10 bits.

4.3. The attenuation of PWM's undesired harmonic content.

The pulse wave used to regulate the voltage source leads to undesired harmonic content, as discussed in the previous sections. The attenuation of the undesired content can be estimated with the process disturbance sensitivity function (Equation 2.9), derived in section 2.4.

$$\frac{y}{d} = \frac{G}{1+GC} \tag{2.9}$$

With G the transfer function of the plant and C the transfer function of the controller. The pulse wave only consists of harmonic content at its fundamental frequency of 42 kHz and higher harmonics. For a system with a 100 Hz bandwidth, the combined transfer function $GC << 1$ at 42 kHz and the process disturbance transfer function can therefore be estimated with G alone. The transfer function of the plant is the combination of the mechanical structure and the actuator, given by:

$$G(\omega) = \frac{1}{j\omega L + R} * \frac{1}{\omega^2 m + j\omega c} * Bl \tag{4.2}$$

, with the inductance of the coil $L = 80\,\mu\text{H}$, the resistance $R = 3.2\,\Omega$, the mass $m = 65\,\text{g}$, the damping coefficient $c = 18\,\text{N s}$, the magnetic field $B = 0.5\,\text{T}$ and the length of the wire in the magnetic field $l = 5.6\,\text{m}$. Evaluation of the transfer function at 42 kHz gives:

$$G(42\,\text{kHz}) = 2.5 \times 10^{-11}\,\text{m V}^{-1} \tag{4.3}$$

For a 5 V power supply, the maximum root mean square amplitude is 2.5 V. Converting this to a peak amplitude $2.5 * 2\sqrt{2} = 3.5$ V results in a worst-case position amplitude of 0.09 nm, which is insignificant and can be neglected.

4.4. Disturbances due to PWM being a voltage source

The PWM based amplifier is a voltage source and not a current source, which in simple terms means that it doesn't have current feedback. Undesired changes in the current are therefore not compensated for, which leads to current errors. Examples of error sources are the induced voltage due to the movement of the stage and changes in resistance due to a temperature change. Identification of the magnitude and the frequency ranges of the errors can be used together with the process disturbance transfer function(Equation 2.9) to estimate their contribution. The procedure is very similar to the analysis of the PWM's undesired harmonic content in the last section and is therefore added in section G.2. However, contrary to the undesired harmonic content of PWM, the transfer function can not be estimated solely with G, because the disturbances are in a much lower frequency range. In this case the controller affects the transfer function, but the disturbances still didn't show a significant effect on the position for this particular component setup and the precision aimed for. Moreover, the inductance of the actuator could be another error source, due to a delayed settling time compared to a linear current amplifier. However, in subsection 2.5.2 it is shown that the actuator acts as a first order low-pass filter with a cut-off frequency of 6.4 kHz, which is 64x the desired bandwidth and can therefore also be neglected.

4.5. Disturbances in signal reconstruction.

Although no real digital-to-analog converter (DAC) is required for PWM, because the pulse wave is basically a digital signal, it does require signal reconstruction to go from a discrete time signal to a continuous time signal. Signal reconstruction can introduce 2 additional disturbances, a disturbance due to imperfect interpolation and a disturbance due to an output quantization error.

4.5.1. Disturbances due to imperfect interpolation

The sensor signal is sampled at discrete time intervals, after which the required output voltage is calculated by the micro-controller. This leads to an impulse train of voltage amplitudes, shown in Figure 4.4a&c. To go from the discrete time signal to a continuous one, interpolation is required. The perfect reconstruction would be obtained by perfectly low-pass filtering the impulse train, which would be convolution by a sinc function in the time domain [18], shown in Figure 4.4b. In practice this is not possible and the signal is usually reconstructed by the micro-controller using zero-order hold. Every program cycle, the duty cycle corresponding to the required voltage is set and hold until the next program cycle, depicted in Figure 4.4d. The impulse train is in this case convolved with a rectangular function in the time domain, instead of with a sinc function. The rectangular function in the time domain is a sinc function in the frequency domain. The frequency of both ideal interpolation and zero-order hold interpolation are illustrated in Figure 4.5. All non-zero frequencies are not transferred over perfectly for zero-order hold. Whenever the time interval of the rectangular function is decreased in the time domain, the frequency ω_s increases. The sinc function is spread over a larger frequency range in the frequency domain, but it maintains its shape. For example, if for an output frequency the first valley of the sinc function happens to be at 100 Hz, then doubling the output frequency leads to the first valley being at 200 Hz. Doing so improves signal reconstruction within the bandwidth, because the frequency response becomes flatter and closer to 1 within the bandwidth.

4.5.2. Disturbances due to an output quantization error

The number of duty cycles is limited to 1000 by the clock frequency and the PWM frequency, as explained in section 4.2. This leads to another quantization error at the output of the micro-controller. The duty cycle will be rounded to the duty cycle that represents the desired duty cycle the best. This quantization error in the digital-to-analog conversion(as explained in section 3.4) can be analyzed in the same manner as the quantization error in the analog-to-digital conversion and quantization noise reduction can be achieved similarly as well. Oversampling is in this case replaced by an increased output frequency and instead of a digital low-pass filter, the low-pass characteristics of the plant itself can be used or an additional low-pass filter can be added. Similar to using dithering to randomize the rounding error, a randomized value could be added to the required output voltage, before it is converted to the best estimate duty cycle. However, the output quantization error enters the system in a different spot in the feedback control loop (Figure 2.4) and therefore doesn't have the same problem as the ADC quantization error, in that it transfers over almost perfectly within the bandwidth. The transfer function attenuates the output quantization error over the whole frequency range and is therefore negligible compared to the ADC quantization error.

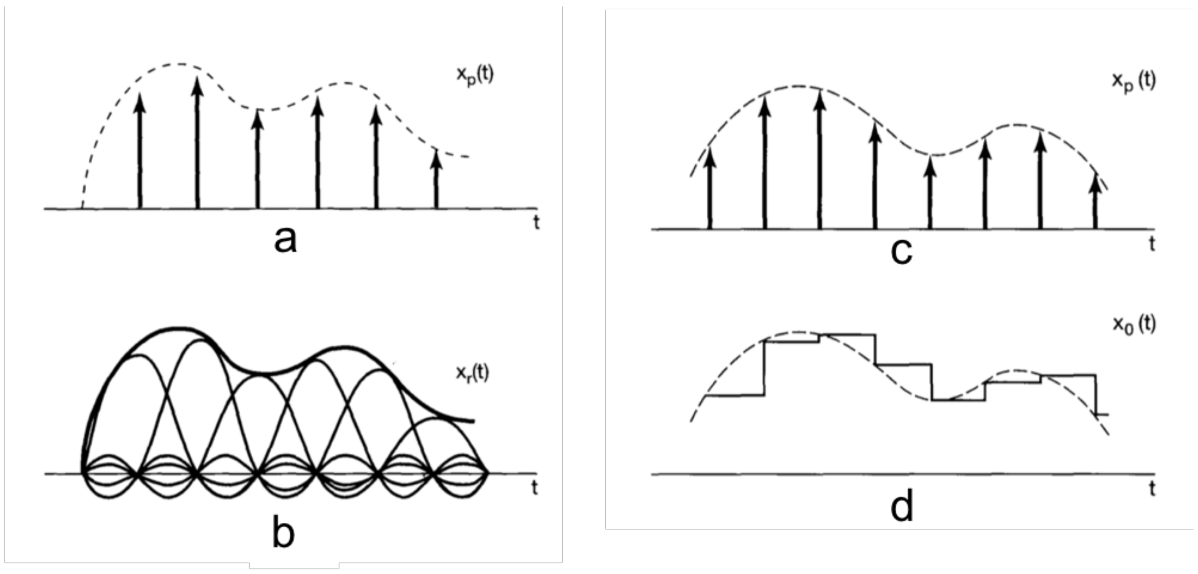


Figure 4.4: The difference between perfect interpolation with a sinc function and zero-order hold interpolation with a rectangular function in the time domain[18]. The discrete time signal in (a) is interpolated with the sinc function in (b). The discrete time signal in (c) is interpolated with the zero-order in (d).

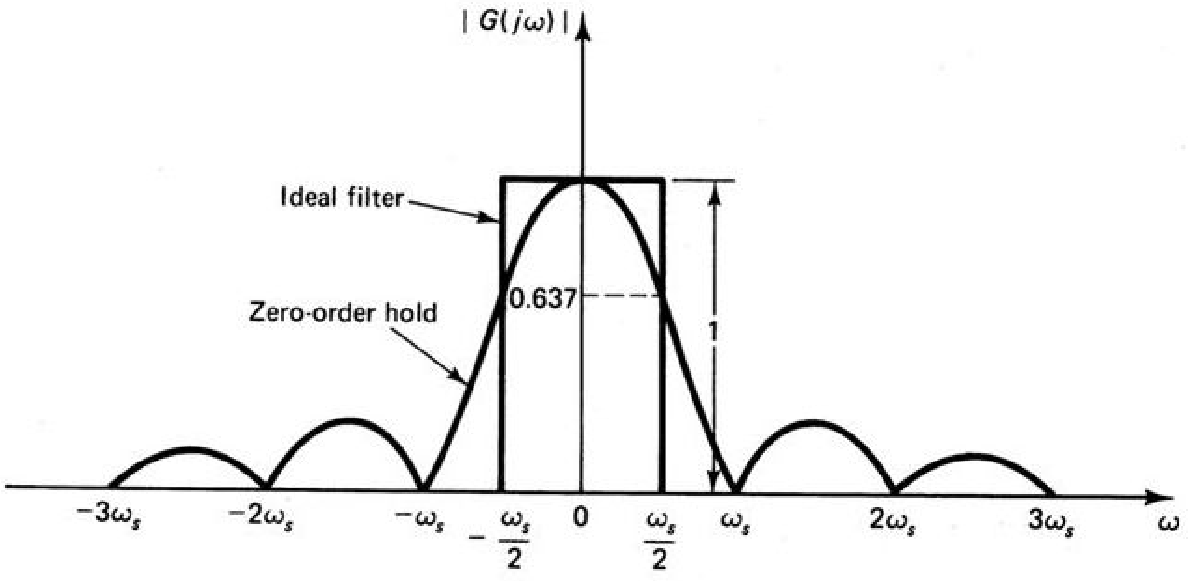


Figure 4.5: The difference between perfect interpolation and zero-order hold interpolation in the frequency domain.[6]

5 The micro-controller

The Arduino Due is the micro-controller, where the control algorithm is implemented. Both algorithm design and the hardware together determine the frequency at which the program can run. The program frequency determines both the sampling frequency and the output frequency, which proved to be important in both the data acquisition and signal reconstruction. The following topics will be discussed in this chapter:

- Why the Arduino is chosen over other micro-controllers.
- The implementation of the control algorithm.
- The processing time of the control algorithm.
- Implementation of oversampling using Direct Memory Access.

5.1. Why Arduino Due?

The choice is made to use the Arduino Due, because it's cheap, has no operating system, is easy to start with and has a large community. Arduino is a widely known open-source hardware and software company that produces cheap micro-controllers that are easy to use and program. Arduino is a simple computer, based on C/C++, which can only run one program over and over again. The programming language C is known to be one of the fastest languages around. The Arduino can be programmed with a development environment from the company itself, which includes a large number of libraries, which makes it easy to program the micro-controller. Arduino also does not have an operating system as other popular choices do. Raspberry Pi, for example, uses the operating system Linux. Micro-controller choices that have an operating system usually have faster CPU's and allow the user to run multiple programs at the same time. The disadvantage is the lack of control over what the operating system does. The operating system might choose to do other things while running the program (such as checking for updates) leading to inconsistent program run times. Inconsistency in sampling time and delay should be avoided when aiming for high precision and reliability. The required processor specifications are hard to predict, because a lot more goes into the speed than just the clock frequency. Things like CPU architecture and program design all influence the time in which a program can run. The fastest version of Arduino, the Arduino Due, is therefore chosen and the performance will be tested experimentally.

5.2. The implementation of the control algorithm

A simplified schematic of the control algorithm is displayed in Figure 5.1, where everything in green is done by the micro-controller. Eight analog signals are converted from which the position and rotation of the stage can be calculated. The position and rotation will be low-pass filtered, reducing sensor noise and quantization noise. A PID controller will calculate the actuator forces based on position measurements and the reference. The 6 individual coil forces are calculated with the use of a transformation matrix. The matrix transforms the actuator forces to the orientation of the coils. Once the individual coil forces are known, the required output voltages can be calculated and the duty cycles of the PWM outputs can be set appropriately. This process is programmed within a loop. All the code is added and commented in Appendix I.

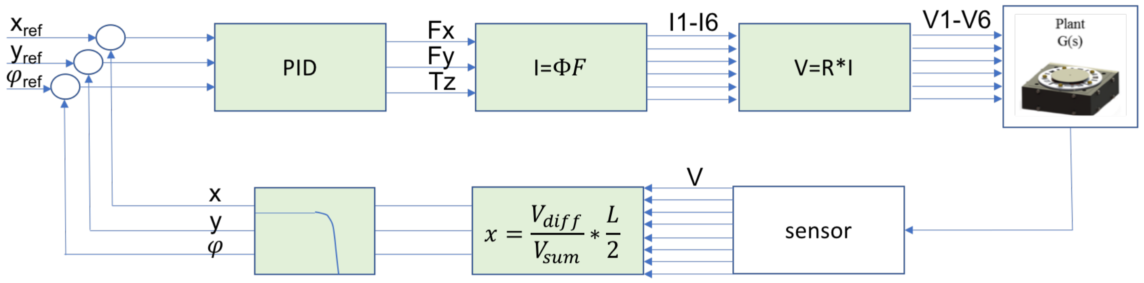


Figure 5.1: Schematic of the control algorithm. All the parts in green are done within the micro-controller. Optionally an extra analog low-pass filter can be used after the sensor.

5.3. The processing speed of the control algorithm by Arduino Due

The processing speed of the control algorithm is experimentally tested and broken down into their different functions in the control loop. The results are combined in Table 5.1. Implementation of the control algorithm shows a processing time of 700 µm, or a program frequency of 1450 Hz, which is 14.5x the bandwidth of 100 Hz.

Table 5.1: The duration a program cycle broken down into its functions.

Algorithm function	time(µs)
Getting position/rotation	300
PID	100
Transform Force to Coil Currents	200
Set Duty cycles	100
Total	700

5.4. Passive and active oversampling and filtering

There is an active and a passive way to implement oversampling and filtering to increase the accuracy of the measurement. In the loop as described above, the program frequency determines the sampling frequency. Increasing the program frequency can therefore be used to increase the sampling frequency and the low pass characteristics of the complementary sensitivity function can then be used as a filter to reduce the noise passively. However, the program speed can only be increased by code optimization or better hardware and is therefore very limited. With the 1450 Hz program frequency, a passive oversampling factor of 7.25 is achieved for 100 Hz bandwidth.

The active method to implement oversampling is to take multiple samples at the start of each loop, digitally filter the samples and then down-sample, so that a single sample with a better precision is used for processing. A high sampling frequency of 667 kHz with Direct Memory Access (DMA) can be used to write measurements directly to memory, without intervention of the CPU. This way the sampling can be done simultaneous with the filtering and the sampling time can be neglected. In this case, the latency due to oversampling is solely determined by the filtering process. The processing time required to filter scales linearly with the data size, which is shown in the chart in Figure 5.2. However, the data size grows exponentially(Equation 3.6). This method is therefore very beneficial to improve the SNR in the start, because the required processing time is low.

The total amount of measurement accuracy that is gained by oversampling and filtering is the combination of both passive and active oversampling. This leads a trade-off, because active oversampling and filtering will slow down the program frequency, which reduces passive oversampling and filter. This optimization problem will be addressed in the next chapter, where all the parts are implemented into a complete system.

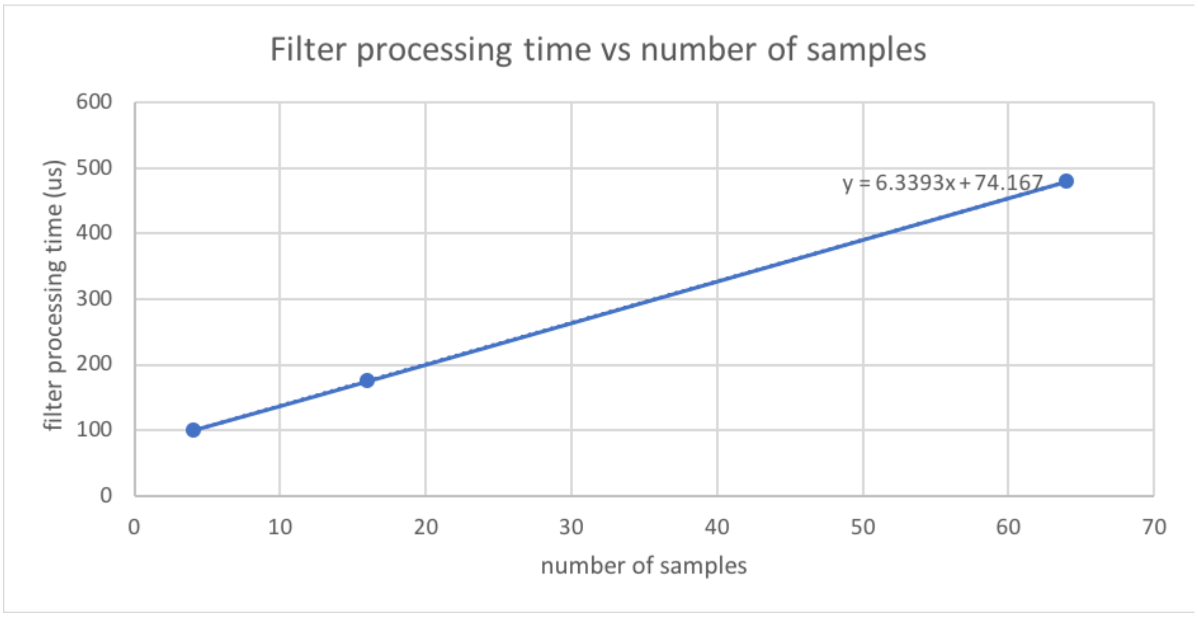


Figure 5.2: The linear scaling of the required processing time for filtering vs the number of samples.

6 Implementation

Now that all the individual components have been dealt with individually, the parts can be implemented into the complete system. The Arduino Due micro-controller(€40) and PWM based amplifier(€6) replace the dSPACE DS-1103 system (€20000) and linear current amplifiers (€1000) of the original design design[22].

A goal set at the start of the project was to maintain as much of the 200 μm precision as possible. Previous chapters already showed that complete preservation is unrealistic, due to the ADC accuracy. However, testing is still valuable to verify the technique of oversampling and filtering to gain accuracy and to verify predictions that can be made about the precision based on the theoretical analysis. Moreover, testing of the complete system will also show oversights in the design process.

This chapter addresses the implementation and evaluates the performance of the complete stage. A few things have to be considered whenever the precision of the stage is evaluated. Design is an iterative process and the prototype is therefore designed for convenience, where the system can be modified quickly and easily. As will become apparent, this can have a big effect on the overall performance of the stage, which doesn't necessarily originate from the components used or the concepts applied, but from a poor realization. Moreover, the procedure used to estimate the precision affects the accuracy of the obtained precision. A good evaluation requires the above mentioned points to be taken into consideration. This chapter addresses the following topics:

- The required signal processing circuits, the measurement noise and why an analog filter can't be used to fix it.
- Tamed PID controller design
- The consequences of using the sensor data of the system to estimate the precision, instead of an external sensor.
- Comparison of the precision for different oversampling frequencies.
- Discussion about the precision results.
- Performance comparison between the old and the new stage.

6.1. The required signal processing circuits, the measurement noise and why an analog filter can't be used to fix it

The output of the PSD can not be connected to the input of the ADC right away for 2 reasons. First off, the sensor puts out both positive and negative voltages, while the Arduino ADC can only be used in the $0 - 3.3\,\text{V}$ range. Furthermore, the output range of the sensor is much smaller than the 3.3 V range of the ADC and not making use of the full voltage range reduces the resolution. A signal processing circuit has therefore been developed to offset the sensor output, so that all sensor output voltages are positive. The output range is also scaled, so that the used ADC range is maximized. The electrical circuit schematics are added in Appendix H. Although the processing circuits are necessary, it is suspected that a lot of noise is added in the measurement process, because the prototyped circuits are made for convenience and not for optimal performance. The circuit is soldered on breadboard, so that it could be easily modified, it has long wires, so that it can be put anywhere on the desk and it is also not properly shielded. The processing circuits are shown in Figure 6.1.

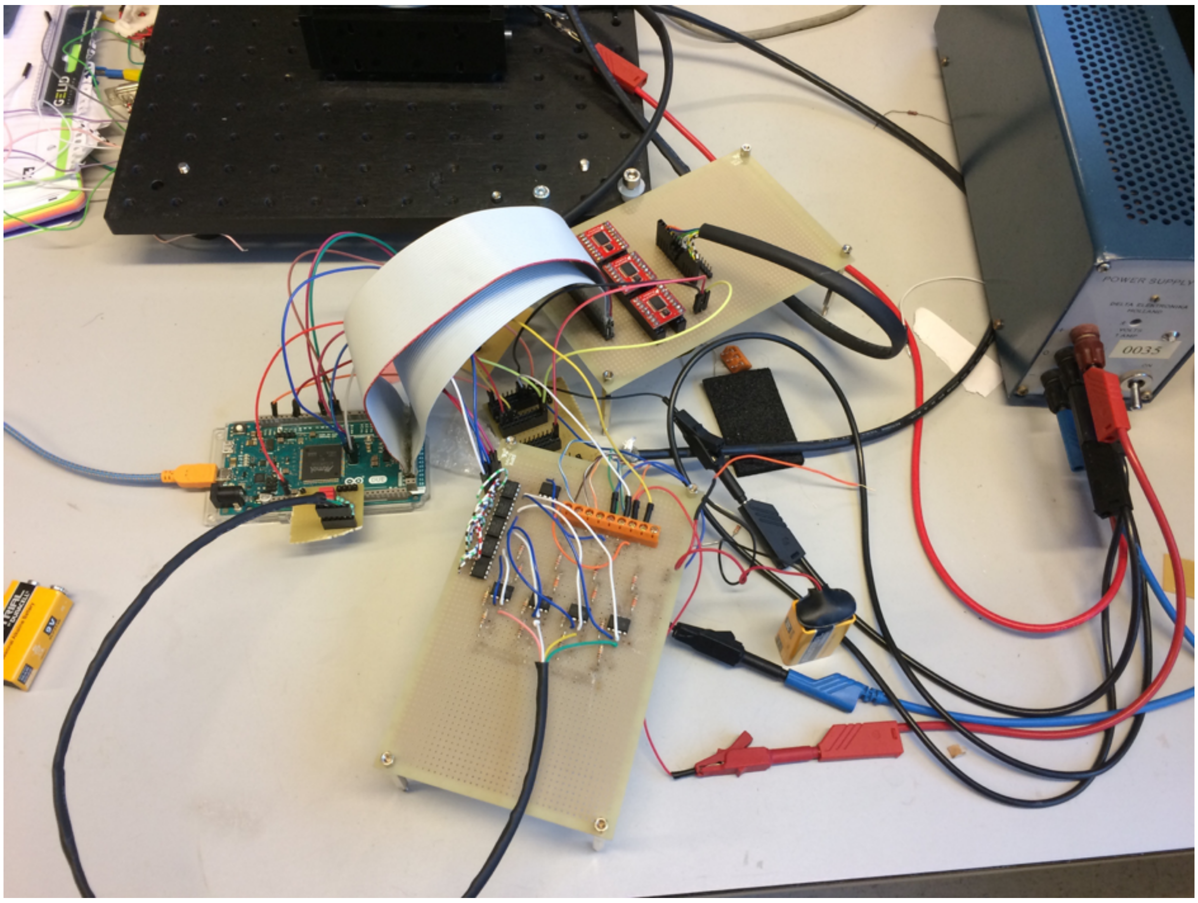


Figure 6.1: The prototyped signal processing circuits in the complete system. Due to improper shielding and long open wires noise will enter the system.

The noise is usually dealt with by adding an analog low-pass filter between the sensor output and the ADC input. However, the concept of this sensor system prevents heavy analog filtering. The problem is that one sensor is used to measure the position of 2 different light spots, one for the x/y position and another for the rotation (sensor concept is explained in Appendix C). With the current control algorithm, both measurements are taken shortly after each other, before further digital processing occurs. Filtering would increase the settling time of the sensor output and therefore increases the latency between measurements, which would slow down the program. Moreover, as seen in subsection 3.6.1 and as will become apparent once again in section 6.4, some noise is required for oversampling to work properly, which is reduced by an analog filter.

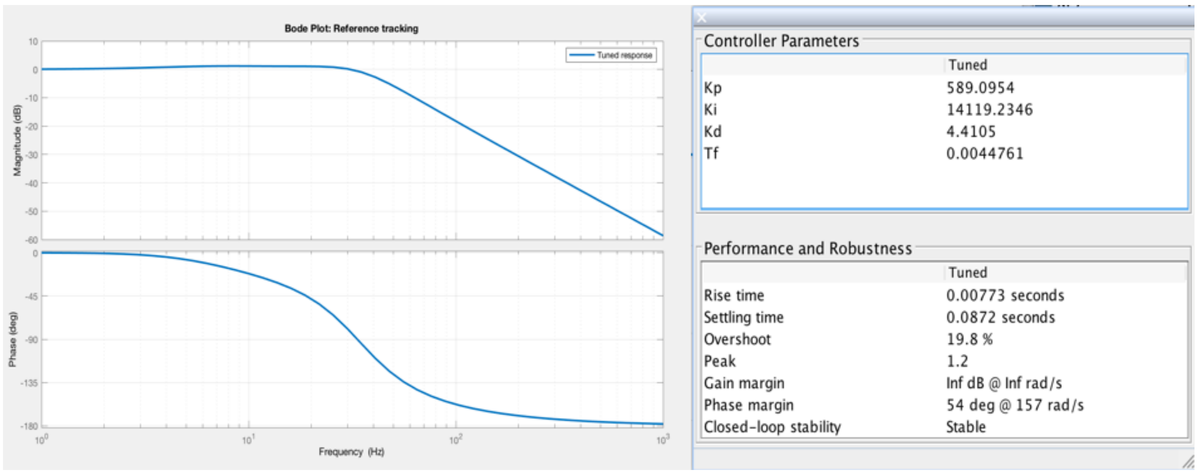
Experimentation showed that a low-pass filter with a cut-off frequency of 100 kHz could be used, without slowing down the program. This isn't claimed to be optimal however. Some latency might be worth the extra noise reduction, especially in the case where measurement noise is very significant. Moreover, it is possible that some of the reduced program speed can be avoided by optimizing/modifying the control algorithm, so that some of the data processing can be done while the sensor settles. This requires further investigation, however.

The measurement noise that can't be filtered in an analog manner has to be filtered digitally. This noise adds on top of the quantization noise of the ADC. Although the added noise is good for oversampling and filtering, because it randomizes the quantization error, the added noise requires more aggressive filtering to reduce the noise to the desired level and more aggressive filtering requires more processing power. The Arduino doesn't have the processing power required to deal with all the noise and the measurement noise is therefore the primary disturbance source.

6.2. Tamed PID controller design

In chapter 3 it is addressed that the disturbances entering the system in the data acquisition affect the performance in a negative way, when the control bandwidth is increased. In the current implementation, where the measurement noise is the primary disturbance source, designing the controller for a high bandwidth wouldn't make sense. The controller will therefore be designed for 25 Hz instead of 100 Hz, even though analysis until now has been done for 100 Hz.

Design of the tamed PID controller is done by making use of MATLAB tools. The first step is to determine the frequency response of the plant with a system identification. The frequency response of the plant is obtained by applying a chirp force signal to the plant, while measuring the sensor output. MATLAB's system identification tool can then be used to estimate the transfer function of the plant. Once the transfer function of the plant is obtained, PID parameters can be determined with MATLAB's PID tool. In the tool the required bandwidth and phase margin can be set and it will automatically calculate the PID parameters. The obtained complementary sensitivity function, the PID parameters and the performance estimations are combined in Figure 6.2.



Controller Parameters

	Tuned
Kp	589.0954
Ki	14119.2346
Kd	4.4105
Tf	0.0044761

Performance and Robustness

	Tuned
Rise time	0.00773 seconds
Settling time	0.0872 seconds
Overshoot	19.8 %
Peak	1.2
Gain margin	Inf dB @ Inf rad/s
Phase margin	54 deg @ 157 rad/s
Closed-loop stability	Stable

Figure 6.2: Controller design with MATLAB'S PID tool. The left part shows the bode plot of the system's complementary sensitivity function. The right part shows the corresponding PID parameters and performance estimations.

6.3. The consequences of using the systems sensor, instead of an external sensor for obtaining the precision of the stage.

The sensor used in the systems feedback loop is used to estimate the precision of the stage. Several things have to be taken into consideration whenever the sensor from the system itself is used instead of an external sensor, because the sensor output is not a perfect representation of the movement of the stage.

The high frequency noise that is added in the measurement process won't transfer over to the position of the stage with the same amplitude. The complementary sensitivity function(Figure 6.2), the transfer function that describes how measurement disturbances transfer over to the position, attenuates frequencies above the bandwidth of the system. The complementary sensitivity function is therefore used to filter the sensor data to get a more realistic view of the real movement of the stage. The effect of filtering the measurement data is shown in the frequency domain in Figure 6.3 and in the time domain in Figure 6.4. The blue line shows the measurement data of the system. The orange line shows the resulting estimated position of the stage.

Moreover, earlier in subsection 3.6.2, the sensor signal was used to test the technique of oversampling and filtering, while not controlling the stage at all, which resulted in the frequency spectrum of Figure 3.7. The frequency spectrum shows a clear peak at 50 Hz. This is the frequency of the power grid and is due to electromagnetic interference, instead of actual movement of the stage. When the stage has a bandwidth higher than 50 Hz, this signal will be followed and will show up attenuated in the sensor data. The performance will be overestimated because of it and should be compensated for in the precision. However, the stage will be designed for 25 Hz and will therefore not be compensated for. It might be able to reduce the 50 Hz noise peak by proper shielding of the electric circuits and/or powering the sensor from batteries.

The non-linearities of the ADC is another error that won't show up in the sensor data. If the stage is moving between 2 bits, then it will be read as if it moves a distance equal to the spacing of an ideal ADC, even though those values might in reality be further apart. Section 3.3 showed that the 12-bit Arduino ADC had only 8 ENOB. How much this should be compensated for exactly is hard to say. Not all the performance loss is due to non-linear spacing, but also due to other errors in the conversion which do show up in the frequency spectrum, such as bit noise. Calibration could be used to get more insight into the non-linear behavior of the ADC, but isn't done due to time limitations.

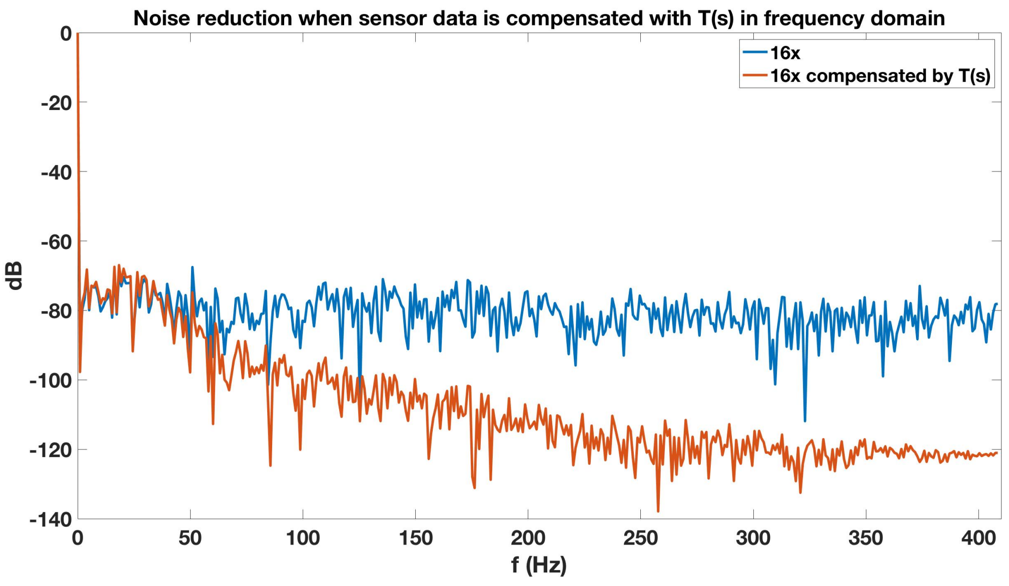


Figure 6.3: The frequency spectrum of the uncompensated measurement data vs the compensated measurement data with the complementary sensitivity function T(s).

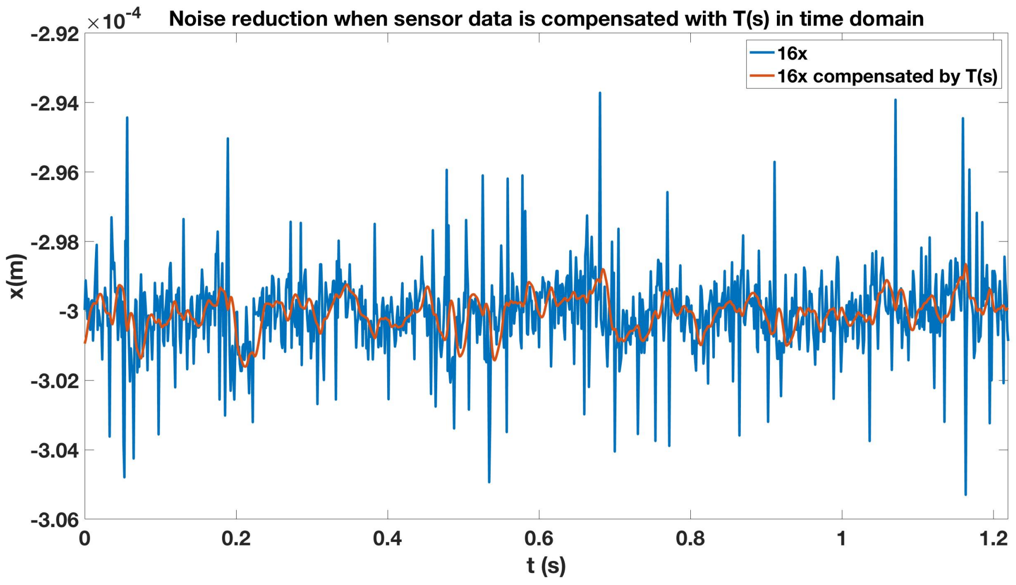


Figure 6.4: The uncompensated measurement data vs the compensated measurement data with the complementary sensitivity function T(s) in the time domain.

6.4. Estimation and comparison of the precision of the stage for multiple oversampling factors

The precision of the stage is estimated for multiple oversampling frequencies and both with and without the 100 kHz analog low-pass filter. The standard deviation (σ) is calculated for both the measurement data and the resulting estimated position, by filtering the data with the complementary sensitivity function T(s). The results without the analog low-pass filter are combined in Table 6.1 and the results with the low-pass filter in Table 6.2. The best precision is obtained by oversampling 16x and using the 100 kHz analog low-pass filter, leading to an estimated precision(3σ) of 1.5 µm.

Table 6.1: The standard deviation(σ) of the measured sensor data and the resulting estimated position by filtering with the complementary sensitivity function T(s) for different oversampling factors, without using an analog low-pass filter.

oversampling factor	$f_{program\_w/o\_filter}$ (Hz)	$\sigma_{measurement\_w/o\_filter}$ (µm)	$\sigma_{position\_w/o\_filter}$ (µm)
1x	930	1500.00	437.54
4x	905	4.42	2.90
16x	820	2.89	1.90
64x	550	1.72	1.31
256x	240	1.10	1.10

Table 6.2: The standard deviation(σ) of the measured sensor data and the resulting estimated position by filtering with the complementary sensitivity function T(s) for different oversampling factors, with an 100 kHz analog low-pass filter.

oversampling factor	$f_{program\_w\_filter}$ (Hz)	$\sigma_{measurement\_w\_filter}$ (µm)	$\sigma_{position\_w\_filter}$ (µm)
1x	930	3.64	1.86
4x	905	1.19	0.77
16x	820	1.16	0.50
64x	550	1.02	0.75
256x	240	1.02	0.95

6.5. Discussing the precision results

Although the best precision is obtained by oversampling 16x and using the 100 kHz analog low-pass filter, more can be learned from the results. Table 6.1 shows that without the analog low-pass filter, oversampling increases the standard deviation of both the measurement and position data significantly. The issue is that the program frequency suffers, because of the large data size that has to be filtered. The low-pass characteristics of the complementary sensitivity function are therefore not used as well, because the Nyquist frequency decreases. The precision results with the 100 kHz low-pass filter are better, because it doesn't require as much digital filtering and can keep the program frequency higher. It does however show much less of an improvement in σ, whenever oversampling is used. It is suspected that the randomization is lost due to the lack of high frequency content as explained in section 3.5. This is believed to be the case, because by extrapolation, $\sigma_{measurement\_w/o\_filter}$ seems to be able to get lower than $\sigma_{measurement\_w\_filter}$ and the slope of $\sigma_{measurement\_w/o\_filter}$ is way steeper. This is made visual in Figure 6.5.

A better result would be obtained if the slope of $\sigma_{measurement\_w\_filter}$ can be improved, so that it better resembles the theoretical improvement (Equation 3.6) where every factor of 4, σ should halve. It might be beneficial to add noise voluntarily after the output of the analog filter and before the input of the ADC. This noise should ideally only have frequency content in the range that ends up in the stopband of the digital low-pass filter, so that the added noise will be attenuated, while it still gives the required randomization.

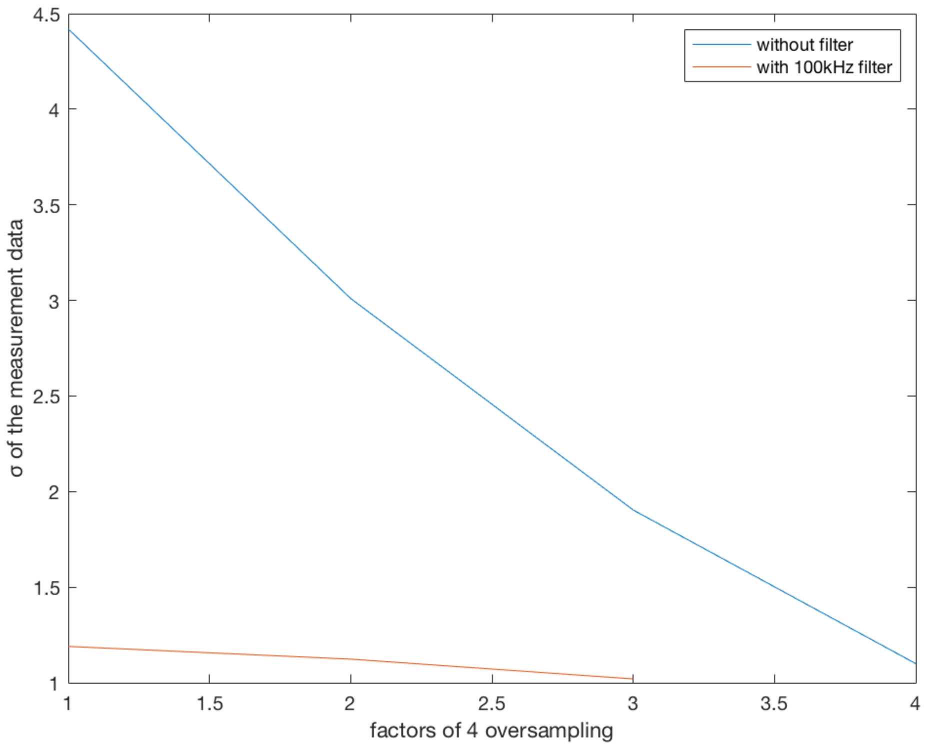


Figure 6.5: Comparison plot of the standard deviation(σ) for the measurement data per factor of 4 of oversampling and filtering.

6.6. Comparing the original and the new design

A comparison between the stage designed by Haris Habib[22] and the new stage in this thesis can now be made. The results are combined in Table 6.3. A precision of 1.5µm is obtained by the new low-cost design, while the original design managed to achieve a precision of 0.2µm. The difference can be explained primarily by the difference in measurement accuracy which has multiple reasons. The 12-bit ADC in the Arduino, which only showed 8 ENOB, doesn't compare to the 16-bit ADC used in the original design. Moreover, the Arduino ADC has an input range of $0-3.3$V, while the dSPACE ADC covers -10V to 10V, which makes the Arduino ADC much more susceptible to noise. Extra signal processing circuits are also required for the Arduino ADC to offset the negative output voltages of the sensor, introducing additional noise. Oversampling and filtering is used to increase the measurement accuracy, but the passive oversampling achieved in the original design outperforms the smart use of oversampling in the new design, due to a much faster processor, which results in a much higher program frequency of 25kHz. However, the new design is only 4% of the price and 5% of the volume. This makes the stage much more accessible to start-ups, individual consumers and 3rd world countries, which is the purpose of this cost-effective stage. The energy efficient amplifiers and micro-controller also open up the possibility to power the stage from batteries completely, which opens up the possibility for a wireless portable stage. If the precision of the new design is not sufficient for the desired application, several things can be explored to increase the performance for little to no extra cost. Recommendations regarding improvement of the stage are summarized in chapter 8.

Table 6.3: Comparison of the performance specifications between the original and the new design

	Original	New
Translational bandwidth	60 Hz	25 Hz
Translational precision(3σ)	0.2 µm	1.5 µm
Program frequency	25 kHz	820 Hz
Total oversampling factor (for 100 Hz bandwidth)	125x	64x
Volume	20 dm^3	1 dm^3
Energy consumption, stationary (10% force)	3 W	0.03 W
Energy consumption, moving (100% force)	30 W	3 W
Cost	€20000	€700

7 Conclusion

Looking into the possibilities to decrease the cost and size of actively controlled micrometer precision stages, an alternative micro-controller and amplifier concept has been successfully analyzed, validated and implemented in a 3DOF stage. A precision of 1.5 µm is achieved, using a single Arduino Due (€40) and 6 H-bridges (€6), to replace the very expensive and bulky D-SPACE system (€20000) and linear current amplifiers (€1000) used in the original design developed by Haris Habib, who managed to achieve a precision of 0.2 µm. The main bottleneck is the noise in the data acquisition. Although the precision is not fully maintained, a very good performance is achieved considering the price and size of the stage. Methods are presented and implemented to improve upon the performance losses that are inevitable with low-cost components. Identification of the disturbance characteristics of the ADC and the PWM amplifier show the system parameters that can be modified to reduce the effect of the disturbances on the position. Insight into these disturbance characteristics can be used for further optimization and/or better design decisions in subsequent designs.

7.1. Conclusions regarding the analog to digital conversion

- Testing of the 12-bit Arduino ADC showed only 8 effective number of bits, while 16 ENOB are required.
- The Signal-to-Noise Ratio of the A/D conversion can be increased by oversampling and filtering the measurement signal, which is directly related to the effective number of bits.
- The effectiveness of oversampling and filtering depends highly on the frequency content of the quantization noise and filter design.
- The quantization noise approaches white noise if the quantization error is random, which is usually the case for complex signals.
- If the quantization error isn't random it can be made random by a technique called dithering, where noise is added voluntarily.
- Experiments showed an improvement of about 5/6 ENOB every factor of 4 of oversampling, with the use of a moving average filter.
- The time required to filter the data and the randomization of the quantization errors are the main limitations of oversampling and filtering.
- Having a program frequency higher than twice the bandwidth is a passive form of oversampling that uses the low-pass filter characteristics of the system itself and doesn't require digital filtering.
- A trade-off between reduced program frequency, due to the latency of active digital filtering and the extra measurement accuracy that can be gained from oversampling and digital filtering leads to an optimization problem.

7.2. Conclusions regarding the PWM based amplifier

- PWM based amplifiers have a lot of benefits compared to linear current amplifiers in low-cost and compact designs. The amplifies can be very small, because the amplifiers are very energy efficient and don't require bulky heat-sinks. PWM based amplifiers are also much cheaper, because no DAC and no high power op-amps are required.
- Using a pulse wave to set the voltage adds extra frequency content to the output signal at the fundamental frequency of the pulse wave and its harmonics.
- Audible noise is avoided by setting the PWM frequency at 42 kHz, to stay clear from the audible range
- Evaluation of a worst-case scenario showed that the higher harmonics have a negligible effect on the position, for a pulse wave frequency of 42 kHz.
- To avoid undesired signal modulation phase-correct PWM mode should be used over fast-PWM.
- The resolution of the PWM amplifier depends on the clock frequency, the PWM mode and the pulse wave frequency. Thousand output voltages can be set for phase-correct PWM, with a pulse wave frequency of 42 kHz and a clock frequency of 84 MHz.
- The limited resolution of the amplifier leads to another quantization error, which can be reduced by increasing the output frequency in a similar manner as oversampling reduces the quantization noise in the A/D conversion.
- An error in signal reconstruction results due to imperfect interpolation of zero-order hold, when going from the discrete time domain of the processor to the continuous time analog output.
- The reconstruction error can be reduced by either increasing the output frequency or the use of a reconstruction filter.

7.3. Conclusions regarding the Arduino Due micro controller

- The Arduino Due is based on the programming language C/C++, which is one of the fastest programming languages around.
- The Arduino's processing time is very reliable, due to the lack of an operating system. Having a reliable program frequency is very important for the robustness of the system.
- Implementation of the control algorithm without any extra oversampling leads to a program frequency of 1450 Hz, which is 14.5 times the bandwidth. The program frequency itself, together with the low-pass characteristics of the complemntary sensitivity function leads to passive oversampling by a factor 7.
- Using Direct Memory Access(DMA) extra samples can be taken within a program cycle, without the sampling itself taking extra time. These samples have to be filtered and down sampled to the program frequency.
- Filtering scales with the data size. Experimentation showed that the time it takes to calculate a single position can be calculated with $t_{get} = 6.33 * os + 74$µs in the current implementation.

7.4. Conclusions regarding the implementation

- Measurement noise is the dominant disturbance source, due to the quantization error, the prototyped processing circuits and a noisy sensor.
- Heavy analog filtering can't be used in the current implementation, because 2 different measurements to be taken shortly after each other by a single sensor.
- An analog filter with a cut-off frequency of 100 kHz can used, while not influencing the program frequency.
- Leftover measurement noise after the analog filter has to be filtered digitally, together with the quantization noise.

- The Arduino Due doesn't have the processing resources to digitally filter the measurement noise to the desired level, which keeps the measurement noise as the dominant disturbance source.
- A lower bandwidth therefore gives a better performance and the PID controller is designed for 25 Hz.
- Oversampling by a factor of 16 with the 100 kHz analog low-pass filter gave the best result, leading to an estimated precision of 1.5 µm.

8 Recommendations

There are several modifications that could improve the precision of the stage, other than the obvious use of better hardware.

- The prototyped circuits should be printed on a PCB and be shielded properly. This should reduce measurement noise significantly. It is recommended to identify the causes of the measurement noise first however.
- Use dithering and noise shaping effectively to increase the benefits of oversampling and filtering, especially when a low-pass filter is used, as explained in section 6.5.
- Experiment with reducing the sampling frequency of the free running ADC. The high sampling frequency of 667 kHz might be causing the quantization error to lose its randomization, which might be the reason why oversampling and filtering doesn't work as well with an analog low-pass filter.
- Code optimization can increase the program frequency or allow for an analog filter with a lower cut-off frequency.

  If none of the above gets the desired precision, the hardware could be improved.

  - Using an ADC with a better SNR would be the best bet to increase the precision for the lowest cost. If an external ADC is used, it is important to investigate if direct memory access can still be used to write measurement directly to memory without intervention of the CPU.
  - Getting a better processor would increase the program frequency, which can be used for either passive or active oversampling and filtering. Very recently the Sony Spresense(€65) has been released, which has 6 cores and a clock frequency of 156 MHz. This is significantly better than the single core 84 MHz processor in the Arduino Due. The board does however only have a 10-bit ADC.
  - Using 2 separate sensors for position and rotation, which would allow for more analog filtering. Using a similar sensor is very costly however, because the sensor is already the most expensive part of the system. This would also require a complete redesign.

A
Extended Abstract

Precise positioning is important in almost all micro/nano fabrication and inspection applications. It is presented that in actively controlled micrometer positioning systems, also called stages, huge reduction in cost and size can be obtained by using low-cost alternatives for micro-controller boards and amplifiers. It is shown that with good mechatronic system design, performance of the precision stages can be largely maintained, obtaining a precision(3σ) of 1.5 µm when an Arduino Due ($40) and PWM based amplifiers ($6) replace much more expensive high-quality parts in conventional systems.

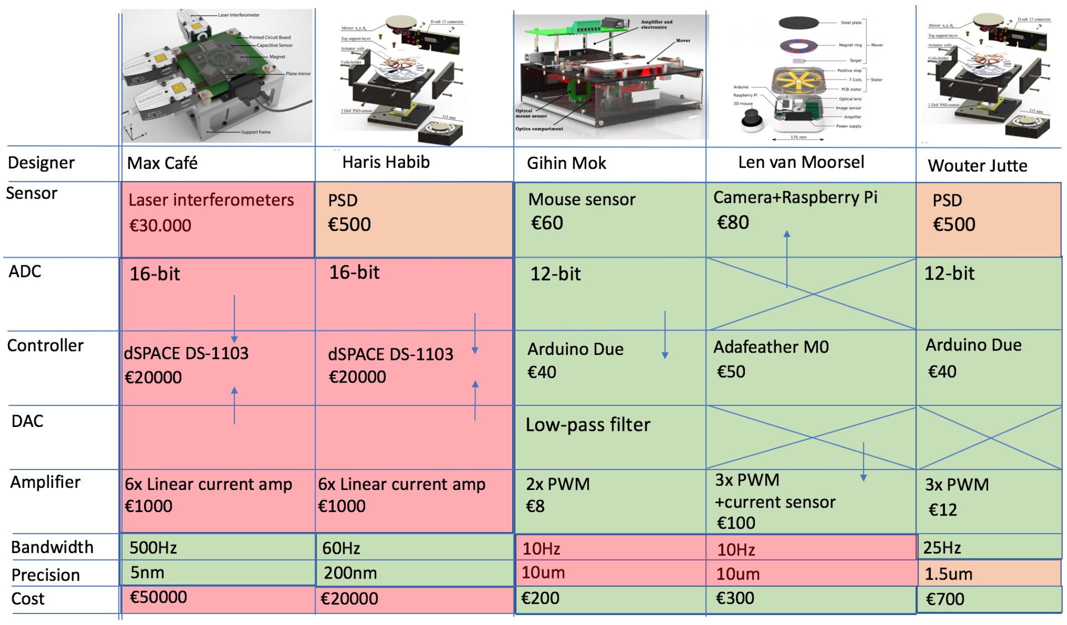
Fabrication and inspection in the micro/nano-scale are rapidly becoming more important in our lives. Industrial fabrication and inspection in this scale is currently done by expensive and bulky systems, optimized for large quantities. Due to their cost and size, these systems are inaccessible for parties that require smaller badge sizes, such as start-ups, individuals or 3rd world countries. Reducing the cost and size would open up all sorts of possibilities in prototyping, research and home health monitoring. Stages are part of what makes these systems so expensive and big and is the part that will be investigated in this research project.

In an extensive literature study into state-of-the-art low-cost stages (added in Appendix B), it became apparent that much of the research focused primarily on performance instead of cost. However, TUDelft's MSD group has developed novel low-cost stages, using promising sensor technologies and mechanical designs of which the most notable four are broken down into their essential components in Figure A.1. The stage designed by MSc student Max Café[19] is an example of a stage designed with the focus primarily on performance, without taking the cost much into consideration. It is able to reach a very high precision of 5 nm, but used all the (in the MSD-lab available) expensive technologies to achieve it. Another stage has been designed by the MSc student Haris Habib[22]. His 3 Degree Of Freedom(DOF) stage uses a single Position Sensitive Detector (PSD), which is a much cheaper sensor system than the conventional laser interferometer, while still achieving a precision of 0.2 µm. The stages designed by Gihin Mok[35] and Len van Moorsel[41] replaced not only the sensor systems, but all components by much cheaper alternatives leading to complete standalone systems of very low cost and both achieved a precision of 10 µm. Even though Gihin and Len replaced everything, most of their focus was on the sensor systems, without an in-depth analysis of low-cost alternatives for the micro-controller and the amplifier. Because none of the designs focused on the alternative micro-controller and amplifiers, the focus of this research project will be:

Alternative micro-controllers and amplifiers for micrometer positioning stages, that are low-cost and compact.

Haris' design is used as a **starting point** for the choice and analysis of the micro-controller and amplifiers. Even though it isn't the cheapest design, it is the only low-cost design that managed sub-micrometer precision. The Arduino Due (€40) and PWM based amplifiers (€6) specifically are addressed as replacements for the dSPACE DS-1103 system (€20000) and self-built linear current amplifiers (components only ~€1000). Replacement of the expensive micro-controller and amplifiers is done while aiming to maintain as much of the 0.2 µm precision as possible. However, insight into the capabilities and problems of these components in general is more important for TUD/MSD than the end result that is specific to the components used in this project. The characteristics of the components is obtained through an extensive theoretical analysis. The components will be analyzed and experimentally verified individually at first, after which it will be implemented and tested in the complete feedback-controlled system.

The **approach** of the analysis is done for all of the components in a very similar manner. Low-cost components usually have in common that some performance is lost, which introduces disturbances into the system. Examples are position measurements that deviate from the true position or output voltages/forces that are different from the intended output. The effect on the position can in general be reduced in 3 ways. Either (1) the disturbance itself is reduced, (2) the system dynamics are designed to attenuate the disturbance or (3) the frequency range of the disturbance is changed to a frequency range that is attenuated by the system dynamics. Frequency analysis is a valuable tool in determining the viability of a component. Once the frequency characteristics of the disturbances are determined, system parameters can be optimized to attenuate the effect of the disturbances on the position of the stage. Although the method is the same for all of the components, the characteristics of the disturbances and their transfer functions are very different and should be addressed separately.



Designer	Max Café	Haris Habib	Gihin Mok	Len van Moorsel	Wouter Jutte
Sensor	Laser interferometers €30.000	PSD €500	Mouse sensor €60	Camera+Raspberry Pi €80	PSD €500
ADC	16-bit	16-bit	12-bit		12-bit
Controller	dSPACE DS-1103 €20000	dSPACE DS-1103 €20000	Arduino Due €40	Adafeather M0 €50	Arduino Due €40
DAC			Low-pass filter		
Amplifier	6x Linear current amp €1000	6x Linear current amp €1000	2x PWM €8	3x PWM +current sensor €100	3x PWM €12
Bandwidth	500Hz	60Hz	10Hz	10Hz	25Hz
Precision	5nm	200nm	10um	10um	1.5um
Cost	€50000	€20000	€200	€300	€700

Figure A.1: Five stages broken down into their essential components, cost and performance.

The built in 12-bit **Analog-to-Digital Converter** (ADC) in the Arduino Due replaces the 16-bit ADC in the original design, which increases the quantization error. Testing of the real accuracy of the Arduino ADC showed only 8 effective number of bits (ENOB) and an estimation showed that 16 ENOB are required to meet the precision requirement of 0.2µm. A technique is presented to increase the ENOB by oversampling and filtering[33]. Analysis of the quantization error shows that, for complex signals, the error approaches white noise in the frequency domain. Oversampling can then be used to increase the Nyquist frequency, leading to the noise power being distributed over a larger frequency range. The part of the noise that falls outside the frequency range of interest, which is the bandwidth of the system, can be filtered out. This is illustrated in Figure A.2. The signal-to-noise ratio of the data acquisition is increased, which is directly related to the ENOB. Experiments showed that the ENOB is increased by 5/6, every time the sampling frequency is increased by a factor of 4 and with the use of a moving average filter, which is a little less than the theoretical improvement of 1 ENOB[33]. This technique doesn't solely attenuate the quantization noise, but also all the other high frequency content from other noise sources such as electromagnetic interference. The main limitations of this technique are the required randomization for the quantization error to be white and the processing time required to filter the data, which grows exponentially with the gain in accuracy.

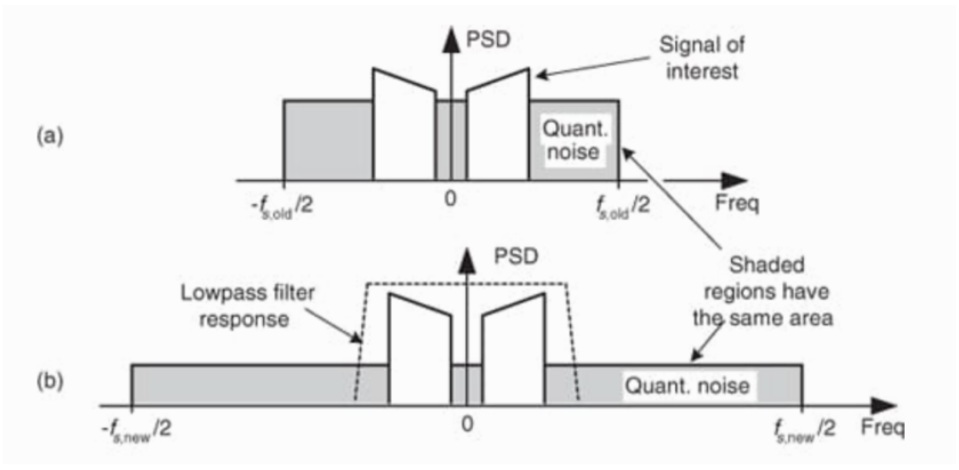


Figure A.2: The effect of oversampling and filtering[33]. (a) Noise PSD at an $f_{s,old}$ sample frequency. (b) Noise PSD at a higher $f_{s,new}$ sample frequency. Increasing the sampling frequency increases the amount of quantization noise that can be low-pass filtered.

The new **amplifier based on Pulse Width Modulation** (PWM) uses the duty cycle of the pulse wave to set the average voltage of the output signal. The pulse wave adds undesired frequency content to the output signal at the fundamental frequency of the pulse wave and its harmonics. This can, outside of a disturbance, also produce audible noise. A switching frequency of 42 kHz is used to stay clear from the audible frequency range. This automatically puts the fundamental frequency of the pulse wave at 42 kHz. Evaluation of the transfer function showed that the disturbances caused by the extra harmonic content of the pulse wave are attenuated to a negligible amount, even in a worst-case scenario. The PWM frequency can't however be set independently.

Increasing the PWM frequency reduces the **output resolution**. The limited resolution leads to another quantization error, which can be analysed in a similar manner as the ADC's quantization error and can be reduced by increasing the output frequency. They do however have different transfer functions, because they enter the system in different places in the feedback control loop. As can be seen from Figure A.3, amplifier errors enter the system as process disturbances, while the ADC quantization error enters as a measurement disturbance. The output quantization error is attenuated much more than the quantization error in the data acquisition and therefore much less of a problem.

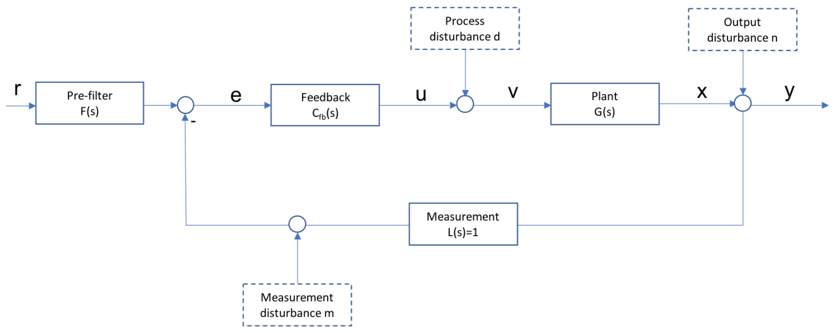


Figure A.3: The feedback control loop of a positioning stage, containing the dynamics and the disturbances of the system.

A disturbance due to **interpolation** is another error that has to be considered in signal construction, which is not specific to the PWM based amplifier. The micro-controller measures and calculates output voltages at discrete time intervals. To go from a discrete time signal to a continuous one, some kind of interpolation is required, which is zero-order hold for most micro-controllers. However, reconstructing a signal with zero-order hold interpolation does not represent the real signal perfectly and leads to a loss in signal fidelity, which is illustrated in Figure A.4. Increasing the output frequency increases the signal fidelity. If the maximum possible output frequency of the system is not sufficient, a reconstruction filter can be used to compensate for the imperfections of zero-order hold interpolation[18]. A reconstruction filter is not implemented however, because the disturbance due to interpolation is negligible compared to the measurement disturbances.

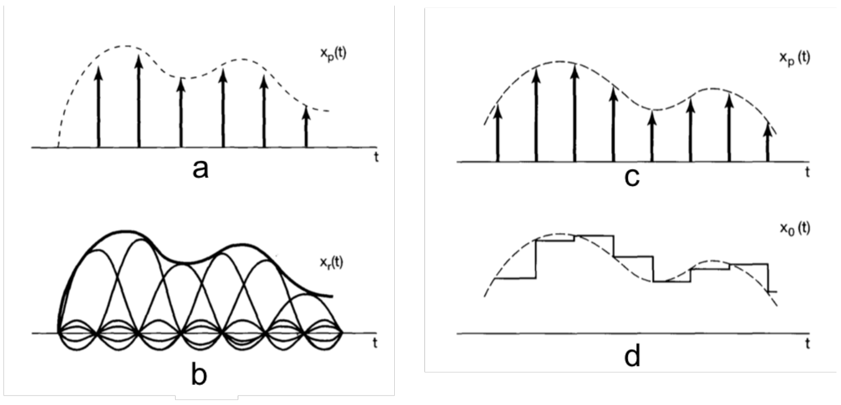


Figure A.4: The difference between perfect interpolation with a sinc function and zero-order hold interpolation with a rectangular function[18]. The discrete time signal in (a) is interpolated with the sinc function in (b). The discrete time signal in (c) is interpolated with the zero-order in (d).

The **Arduino Due** is the new selected **micro-controller**, where the control algorithm is implemented. Both algorithm design and the hardware together determine the frequency at which the program can run. The program frequency determines both the sampling frequency and the output frequency, which proved to be important in both the data acquisition and signal reconstruction. There are 2 ways to implement oversampling and filtering to reduce the ADC's quantization noise. The first method is to use a program frequency higher than twice the Nyquist frequency, where the low-pass characteristics of the complementary sensitivity function is used as a passive filter. The other 'active' method is to take multiple samples within a single program cycle, which can then be digitally low-pass filtered and down sampled to the program frequency. This is implemented with the use of the Arduino Due feature Direct Memory Access(DMA), which allows samples to be written to memory without intervention of the CPU. This way sampling frequencies of up to 667 kHz can be achieved and sampling can be done simultaneous with the processing. The sampling time itself can then be neglected and the extra time required is purely defined by filtering and down sampling of the measurement data. Using the 'active' form of oversampling and filtering is very efficient to get ADC accuracy at first, because the required data size is small, but scales exponentially. Filtering 16 extra samples within a single program loop leads to an estimated total reduction(passive+active oversampling) in quantization noise equivalent to 2.5 bits, while using passive filtering alone gives only 1 extra bit.

The Arduino Due and PWM based amplifiers are **implemented** into a complete system after the components are addressed individually. Measurement noise became so dominant over the other disturbance sources, due to the required measurement processing circuits and an already noisy sensor, that optimal PID design for precision meant designing for low control bandwidth. Heavy analog low-pass filtering can't be used to solve the issue, due to the sensor concept, which requires a single PSD sensor to take 2 different measurements shortly after each other. Heavy filtering would reduce the response of the sensor and slow down the control loop/program frequency. Experimentation showed that an analog low-pass filter with a cut-off frequency of 100 kHz could be used without slowing down the program. The left over measurement noise has to be filtered digitally, together with the quantization noise. However, the required processing resources to filter the extra noise to the desired level are not available. Because sensor noise is so dominant, the system is designed for 25 Hz. Testing results for different oversampling factors without an analog low-pass filter are presented in Table 6.1 and with the 100 kHz low-pass filter in Table 6.2. Oversampling by a factor of 16 with the 100 kHz low-pass filter turned out to be optimal, leading to an estimated precision of 1.5 µm. The improvement gained by oversampling and filtering with the analog low-pass filter doesn't reflect what is theoretically possible. It is suspected that much of the randomization required for oversampling and filtering to work properly is lost due to the lack of high frequency content. Adding noise voluntarily might in this case increase the precision. The current stats of the complete system are combined in Table A.1.

Table A.1: Comparison of the performance specifications between the original and the new design

	Original	New
Translational bandwidth	60 Hz	25 Hz
Translational precision(3σ)	0.2 µm	1.5µm
Program frequency	25 kHz	820 Hz
Total oversampling factor (for 100 Hz bandwidth)	125x	64x
Volume	20 dm^3	1 dm^3
Energy consumption, stationary (10% force)	3 W	0.03 W
Energy consumption, moving (100% force)	30 W	3 W
Cost	€20000	€700

In **conclusion**, looking into the possibilities to decrease the cost and size of precision stages, an alternative micro-controller and amplifier concept has been successfully analyzed, validated and implemented in a 3DOF stage. A precision(3σ) of 1.5µm is maintained out of the original 0.2µm, using a single Arduino Due (€40) and 6 H-bridges (€6), instead of the very expensive and bulky D-SPACE system (€20000) and linear current amplifiers (€1000). The main bottleneck is the noise in the data acquisition. Although smart use of oversampling and filtering can reduce the noise considerably, it is not sufficient to make up for the total disparity in precision (yet). However, considering the cost and size, the new design still performs very well. Further optimization and making smart use of dithering together with noise shaping look very promising to increase upon the precision with little to no extra cost.

B Literature study into state-of-the-art precision positioning systems

Rapid progress in several high-tech industries has significantly increased the need for dimensional micro- and nano-metrology. Measurable structures are becoming more and more complex, meaning that measurements are being made on larger surface regions with higher precision and sidewalls with larger aspect ratios as well as fully 3D micro- and nanostructures. The advancements being made in making those 3D structures, results in a high demand for a system to examine and map these kind of structures. New systems and technologies have been developed to overtake the metrological challenges of accuracy, repeatability and stability when positioning is required at the micro- and nanoscale.[34] However, the technologies used in those systems are very costly. There is a large gap to be filled in affordable precision stages for the examination and mapping of 3D-structures.

A very large number of different precision stages have been developed over the years. In [40] a large number of stages are compared and broken down into their essential components. Each of the systems could be subdivided in five major categories, which are the following: The stage structure, it's actuator, the choice of the bearing, its sensor and the micro-controller.

B.1. Stage Structures

The different stages could be classified into two stage structures (see Figure B.1). The two groups are the following:

1. *Stages with stacked linear axes:* Conventional designs based on stacked linear axes are characterized for long kinematic chains with an unfavorable force transfer behavior [27, 32]. Generally speaking, stacking up driving systems limits the performance, on account of some additional drawbacks: large-size and unbalanced structure, lack of stability, accumulated geometric and assembly errors, necessity of sophisticated control systems, etc.

2. *Plane stages:* They present some advantages compared to stacked structures, considering precision design features. The co-planar scheme minimizes positioning deviations, so that it is applied in multiple systems [19, 22, 23, 40, 41]. It does however require bearings and sensor concepts that can work with multiple dimensions.

B.2. Actuator

There are multiple types of actuators on the market, which have different benefits. Which actuator to use is dependent on the requirements of the stage. In the majority of literature, three different types are used, which are the following:

1. *Piezoelectric actuators:* This kind of actuator is capable of making very fine and precise movements and is therefore used in very high precision stages. The stroke of a piezoelectric actuator is very limited and therefore only used if only a very short range is required. However, it can be stacked on a platform that is driven by another actuator.

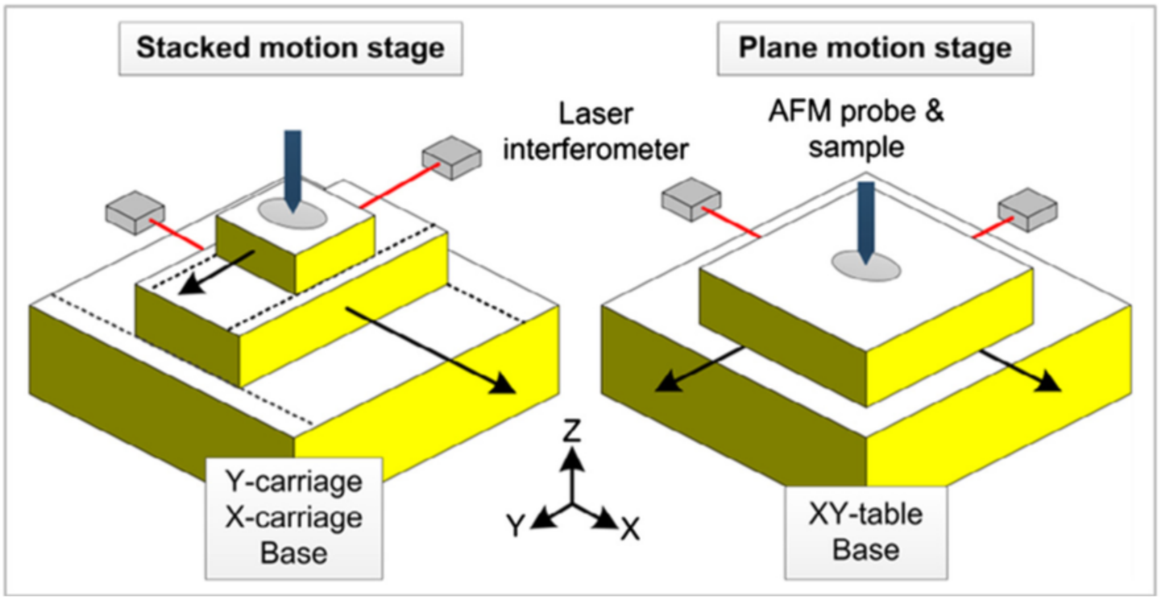


Figure B.1: Stacked motion stage vs plane motion stage[40]

2. *Voice coil motors(VCM):* Voice coil motors are work with the lorenz principle and work in 1 degree of freedom. Reversing polarity is used to change the motion direction avoiding commutation, whcih more simplicity and reliability. Other performance advantages are their high bandwidth and easy control. The main drawback is the relative limited stroke and resolution.[39] Commercial solutions are available up to 100mm, but with increased sizes in diameter and length, and poor performance. In [26] voice coil motors are integrated in a stacked configuration of VCM and linear guides, which concludes in micrometer errors.

3. *Linear motors:* Most used applied solutions are brushless DC motors. These linear motors are the combination of a coil and a permanent magnet assembly, with the peculiarity of the parallel and flat structure between both non-contact parts. They provide high forces and high speeds with a compact design. Less number of elements, no mechanical wear and simple maintenance are also advantages. These actuators are also characterized by low stiffness, vibrations and velocity ripples, because of the control commutations.The precision along the motion is achieved by the integration of a guiding system and positioning feedback devices. Two are the configurations that lead the 1D-displacement: linear bearings and the own architecture of the actuator. Linear bearings, such as traditional lead screws and linear rails, introduce the disadvantages of backlash, lack of stiffness, friction, vibrations, and maintenance.

4. *Homemade Actuators* Homemade actuators can be made based on the linear motor principle, where the geometric placement of the coils and the magnets can produce forces in multiple directions. With position feedback a plane stage stucture can be used, instead of a stacked stage with linear axis. As already said, the planar configuration can minimize geometrical errors, improve dynamics and stability and facilitate control tasks.

B.3. Bearing

To reduce the friction between the moving platform and the structural frame some kind of bearing is required. At the same time it should restrict the movement in the undesired degrees of freedom. The following types of bearing are the most common.

1. *Roller bearings:* Although widely available, the presence of friction between mechanical parts results in wear, lower accuracy, lower repeatability and the need for a lubricant.

2. *Teflon bearing:* Teflon contacts would be the easiest solution. This type of bearing introduces stick-slip behavior however [35], which is difficult to control.

3. *Hydrostatic bearings:* Hydrostatic bearings are known for their low stick-slip and are based on a fluid such as oil, water or air.There are different ways to achieve an hydrostatic bearing.

   (a) *Magnetic levitation:* This kind of levitation is based on the creation of magnetic fields to generate opposite forces to the gravitational effect. This type of stage makes use of an Halbach array where at some points in space the magnetic field is perpendicular to the moving platform to produce the required translational movement, and at other points the magnetic field is parallel to the platform to produce the levitation force [24] (see Figure B.2).

   (b) *Air bearing levitation:* Alternatively, positioning stages can successfully integrate air bearings. An air flow through a porous surface of the bearing provides relative motion between the main stationary base and the positioning table. Air bearings are limited by airflow stability and bearing surfaces quality, which increases manufacturing costs.

   (c) *Ferro fluid bearing:* Several research projects in the topic of ferrofluid bearings have been conducted in recent years in TUDelft's MSD research group, by Simon van Veen[43], Max Café[19] and Stefan Lampaert[29]. Ferrofluid consists of nanometre-size ferromagnetic particles suspended in a liquid carrier, usually oil. Ferrofluid brought into contact with a permanent magnet, collects at the corners of that magnet, the locations of highest magnetic field intensity. The ferrofluid can be used to seal a pocket of air. This pocket of air acts as a load carrier.

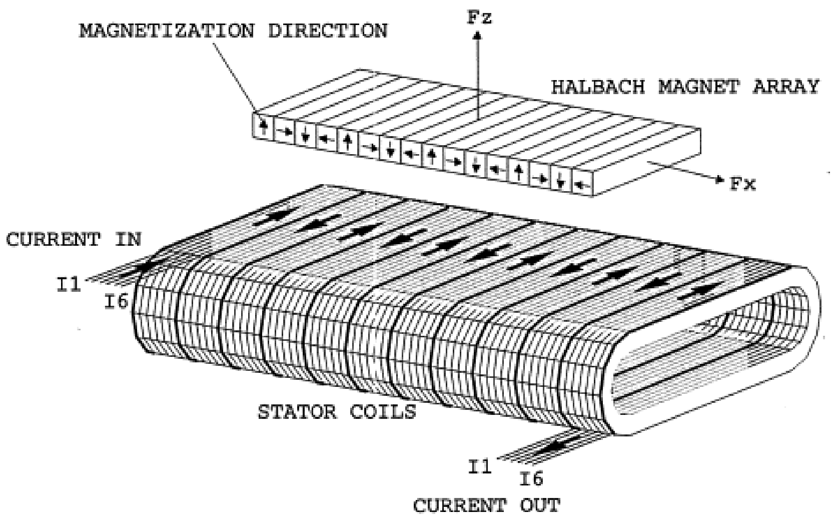


Figure B.2: Magnetic levitation, with the use of a Halbach array[24]

B.4. Sensor Concepts

B.4.1. Laser Interferometer

Laser interferometers are widely used for the measurement of small displacements. A light beam is split into two different light beams, a reference beam and a sample beam, that travel different optical paths. The reference beam travels to a mirror with a fixed path length, while the sample beam travels to an object that has a reflective surface after which both beams travel to a detector (Figure B.3). The resulting interference fringes give information about the difference in the optical path length. [7]

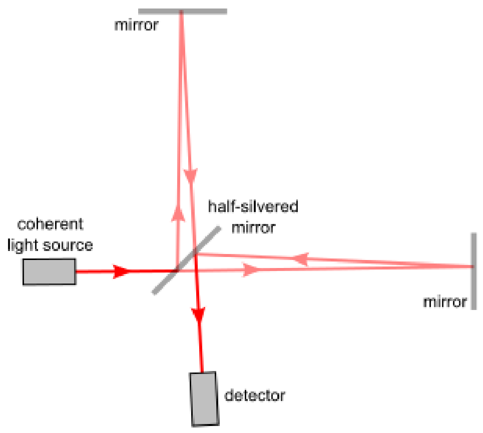


Figure B.3: The laser interferometer concept. The difference in path length between a reference beam and a beam hitting a mirror on the object to be measured leads to interference, from which the incremental distance can be obtained.[7]

Advantages:

- Extremely high resolution<1nm

Disadvantages:

- Expensive
- Susceptible to misalignment of the reflective surface, which makes them a bad choice for stages that rotate.
- susceptible to variation in refractive index of the medium (air), which leads to a loss of performance over long distances.
- Incremental encoder.

B.4.2. Linear Encoders

A linear encoder is a sensor, transducer or read-head paired with a scale that encodes position. The sensor reads the scale in order to convert the encoded position into an analog or digital signal, which can then be decoded into position. Linear encoders are transducers that exploit many different physical properties in order to encode position. There are optical, magnetic, capacitive, inductive and eddy current linear encoders[1]. Optical encoders have the highest resolution, but are more susceptible to contamination and are generally more expensive than the other ones. Optical linear encoders can be subdivided into different categories that use a different type of encoding. There are incremental and absolute optical encoders. Incremental optical encoders often use the moiré effect to amplify very small movements. The Moiré effect occurs when two patterns with parallel lines that have a slightly different spatial frequency are superposed. This gives rise to a pattern with a different spatial frequency as shown in Figure B.4. The patterns can be obtained using diffraction gratings [9] or by making smart use of the pixel grid of a CCD camera and a printed pattern [20] [45] [25]. Absolute linear encoders use a pattern that is unique in every position[21][30][31][10]. Instead of lines that are equally spaced, it uses a coded pattern such as a barcode or QR code.

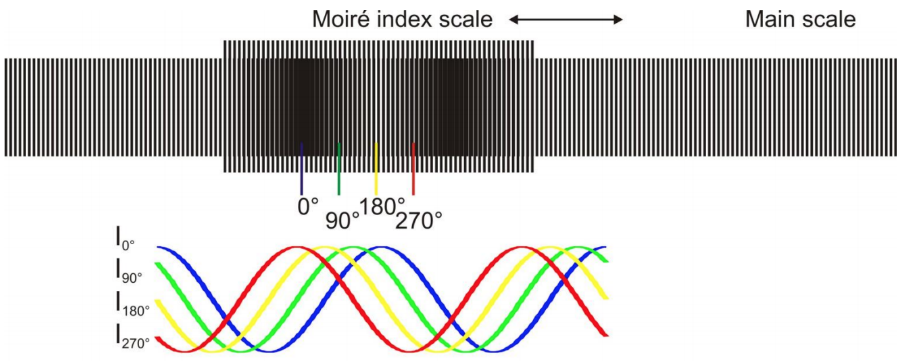


Figure B.4: The Moiré effect. Two gratings with a slightly different spacial frequency lead to a third pattern with a different spacial frequency.[5]

Advantages:

- High resolution possible (1nm)[42][9]
- Large ranges possible

Disadvantages:

- 1 DOF
- Stacked stages are required to get multiple DOF
- The distance between the read head and the scale is often very important, which requires some kind of guiding system that keeps the required distance constant.
- Susceptible to contamination

B.4.3. Optical image recognition

Optical image recognition sensors use images taken by a camera. Optical image recognition sensors are capable of multi-DOF position measurement. Incremental image recognition sensors take multiple images and determine the movement by comparing successive images. Examples are the optical mouse sensor [35] and the image correlation sensor [2]. An absolute image recognition position sensor has been developed in [41]. The sensor uses an algorithm that can determine the position in 6 DOF, based on images of a raster of QR codes. Based on the QR codes in the image and the orientation the absolute position can be determined, illustrated in Figure B.5.

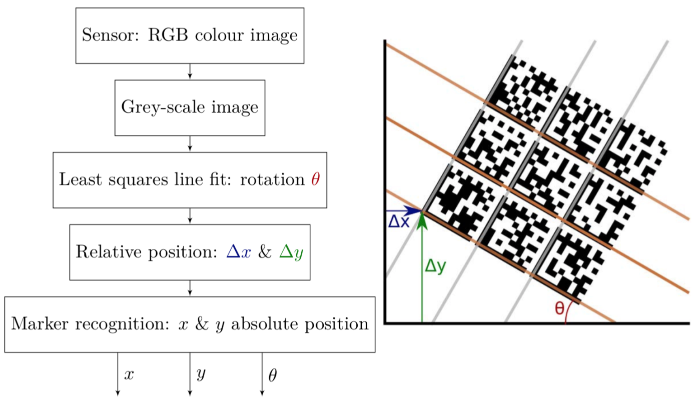

Sensor: RGB colour image
Grey-scale image
Least squares line fit: rotation θ
Relative position: Δx & Δy
Marker recognition: x & y absolute position
x
y
θ
Δx
Δy
θ

Figure B.5: The optical image recognition concept[41]

Advantages:

- Very low cost
- Multi-DOF
- Large ranges possible without loss of performance
- Also work with rotations

Disadvantages

- Fairly new in stages and therefore underdeveloped.
- Performance not comparable to other sensor techniques.

B.4.4. Position Sensitive Device PSD

The PSD sensor uses a light sensitive surface to determine the x/y position of the centroid of a light spot. The PSD sensor is used in multiple sensor concepts. It can be used as an incremental 2D surface grid encoder [44]. In this concept a light beam is reflected of a sinusoidal machined surface grid (Figure B.6). The angle of reflection changes when the surface moves relative to the light beam and will therefore change the position of the light spot on the PSD.

An absolute 3 DOF sensor concept has been developed in [22]. Three different LEDs are switched on in a successive order, so that only 1 LED is on at a time. The x/y translation and rotation around z can be determined based on the measured position of the different light spots, illustrated in Figure B.7.

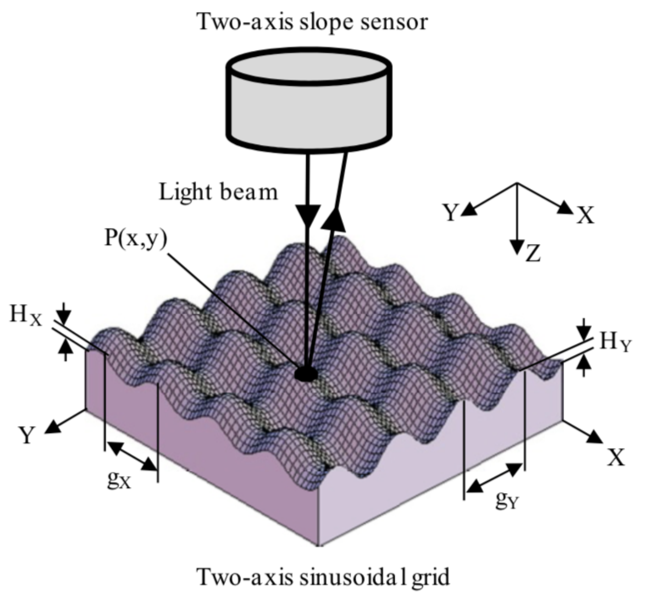
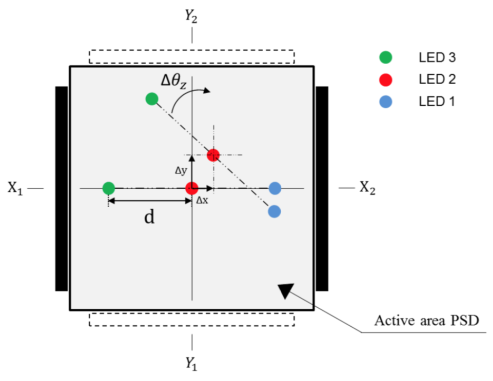
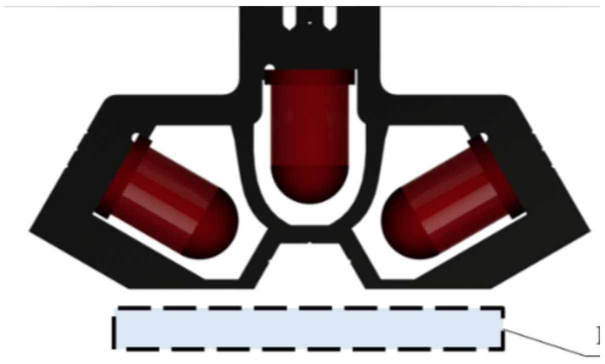


Figure B.6: The 2D surface grid encoder [44]



(a)



(b)

Figure B.7: A 3DOF absolute position sensor concept[22]

B.4.5. Capacitive Proximity Sensor

Capacitive sensing is a technology based on capacitive coupling. Changing the airgap between the measurement probe and another surface changes the capacitance, which is measured by the probe**??**. A capacitive sensor is used in [19] to determine the pitch, roll and z-translation of a stage. Capacitive sensors have a resolution comparable to laser interferometers, while being relatively cheap. However, capacitive sensors only work well for very small displacements.

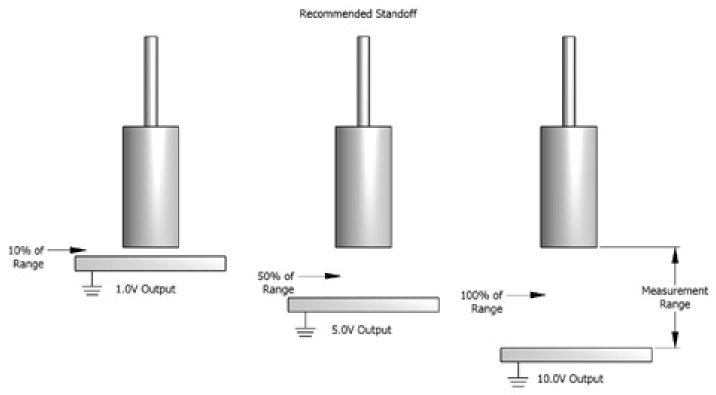


Figure B.8: The capacitive proximity sensor concept.[11]

B.4.6. Current Tunneling Sensor

A current tunneling sensor is proposed in [28] as a cheaper and faster alternative to capacitive sensors. The distance of small air gap between 2 conductors determines the current that will flow, when a voltage is applied. This concept works only for very small distances and is susceptible to contamination.

B.5. Low-cost controllers

Micro-controllers are used for data processing and control. It is the part of the system where the control algorithm is implemented. Micro-controllers can be divided into sub categories based on their use of an operating system. The most popular low-cost micro-controller brand with an operating system is the Raspberry Pi and the most popular brand without an operating system is the Arduino. Due to a lack of experience with micro-controllers at the start of this project, these are the only micro-controllers that are considered, because of their large community. Both controllers have their benefits.

B.5.1. Raspberry Pi

A Raspberry Pi is a general-purpose computer with a 1.4GHz 64-bit quad-core processor, usually with a Linux operating system, and the ability to run multiple programs. However, it does not have a real-time clock. Moreover, current Linux kernels don't support real time and standard Linux installations generate lots of overhead. The RPi can therefore not generate deterministic pulses to control DC motors and the program processing time is also inconsistent.[12]

B.5.2. Arduino

An Arduino is a simple computer that can run one program at a time, over and over again. It does have real time capabilities and doesn't have an operating system. Processing and pulse generation is therefore very deterministic and can be used to control DC motors, contrary to the RPi. The Arduino programming language is based on C/C++, which is one of the fastest languages around. The Arduino does have a much lower clock frequency however, with only 84MHz for their fastest model, the Arduino Due. Another advantage of the Arduino Due is the large amount of GPIO pins, which is required to control all the components in the stage.

C
Overview of the mechanical structure and the sensor concept

[22] In this appendix the overall design will be discussed. The choice has been made to not put the focus of my research on redesigning the stage developed by Haris Habib and his mechanical design and sensor concept will be used in my project with very little changes. An overview of his design will be summarize in this appendix, where a lot of the important parts for my project are just copied from Haris' original thesis[22]. For a more in-depth analysis of the stage, the reader is should consult the original document.

C.1. Mechanical Design

The complete mechanical design including all components is shown in Figure C.1. The essential components are the mover, which include both the magnets and the LED's, the actuator coils and the PSD, which will all be discussed separately in the following subsections.

C.1.1. Mover

The mover is illustrated in Figure C.2. The mover consists of two alternating magnetic rings, two iron plates to guide the magnetic field and 3 LED's that are part of the sensor system.

The magnetic rings form a magnetic field through the coils located in the stationary part of the stage. Driving current through the coils, can then be used to create a Lorentz force between the mover and the static coils, which is used to actuate the stage. Iron plates are attached to the magnetic rings with the purpose of guiding the magnetic field. The actuator concept is illustrated in Figure C.3.

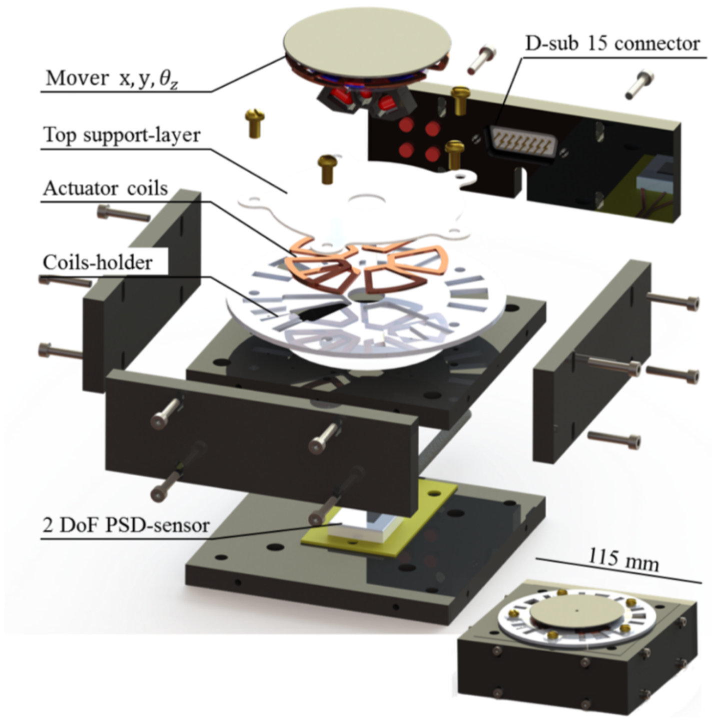


Figure C.1: Visualization of the demonstrator stage showing the mover, actuator coils, PSD and laser-cut (support) structures.

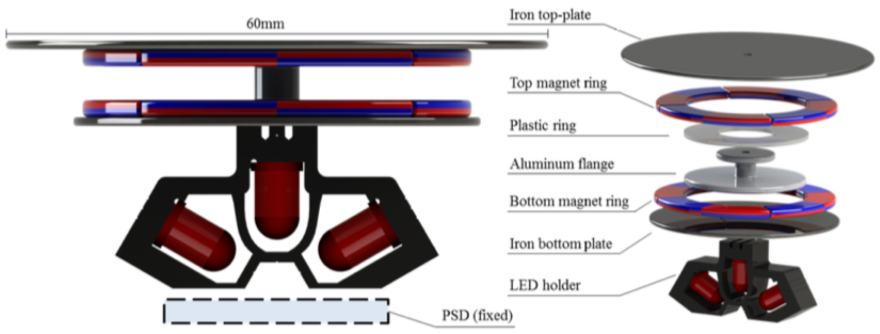
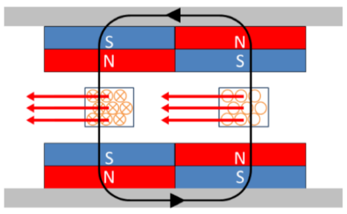
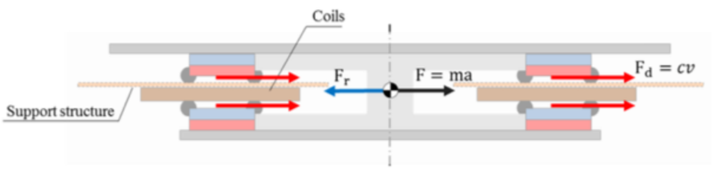


Figure C.2: Complete overview of the mover containing two metal plates, two ring-shaped magnet arrays, an aluminum flange and a LED holder containing three LEDs.





(b)

(a)

Figure C.3: (a) Lorenz Actuator concept, using iron plates to guide the magnetic field. (b) Schematic visualization of the forces produced by the actuator.

C.1.2. Ferrofluid bearings

To reduce the friction between the moving platform and the structural frame some kind of bearing is required. At the same time, it should restrict the movement in the undesired degrees of freedom. Ferrofluid bearings are used, because of the absence of stick-slip behavior, which allows for very precise positioning.

The concept of ferrofluid as explained in Haris' thesis:

"Ferrofluids consist of a carrier fluid, nanoscale ferromagnetic particles and a coating on each particle which prevents agglomeration (Figure C.4a). Ferrofluids move to the region with the highest magnetic flux. For permanent magnets this is around the edges of the magnet. When ferrofluids are applied to a permanent magnet it clusters around the edges. The magnetic forces build up a pressure in the fluid. Figure C.4b shows how this looks for a single magnet. The pressure created in the fluid is able to carry the load. The fluid pressure is directly related to the load it can carry.

Translational ferrofluid bearings can be divided into two main categories: the first one being bearings that only use the flotation effect and secondly bearings that employ the ferrofluid as a seal for a pressurized volume which bears the load. Both categories utilize the great advantage of ferrofluid bearings, namely that there is no direct contact between the moving parts. This results in almost no static friction and stick-slip. However, the viscous friction can be large depending on the fluid viscosity."

When Haris designed his stage, understanding of ferrofluid bearings was fairly limited. Therefore, it is not mentioned that the use of two magnetic rings, with opposing polarity, distributes the magnetic flux much more uniformly. The fluid won't cluster at the edges as much, but spreads over the entire magnet. Therefore, the bearing primarily uses the flotation effect. For future designs it has to be kept in mind that there is a trade-off to be made. The advantage of having two magnet rings is an increased actuator force and a force acting on both the top and bottom side, resulting in a moment cancellation when the stage is moved. However, the total contact area of the ferrofluid is increased, increasing the damping of the bearing.

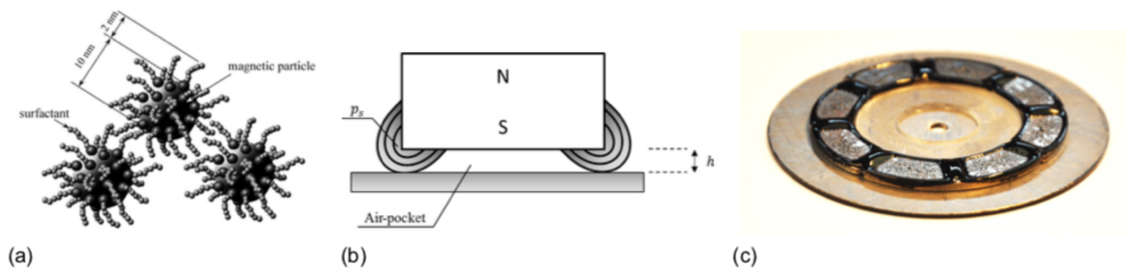


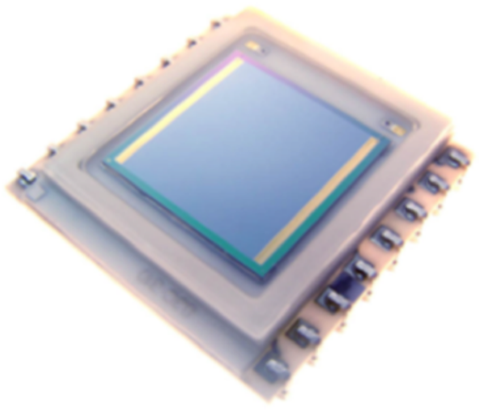
Figure C.4: (a) Schematic view of the coated magnetic ferrofluid particles [14]. (b) Schematic representation of the pressure built-up in a ferrofluid volume around the magnet edge. (c) Ring-shaped magnet array containing ferrofluid. In total there are 8 air-pockets that are sealed by ferrofluid.[36]

C.2. Sensor Concept

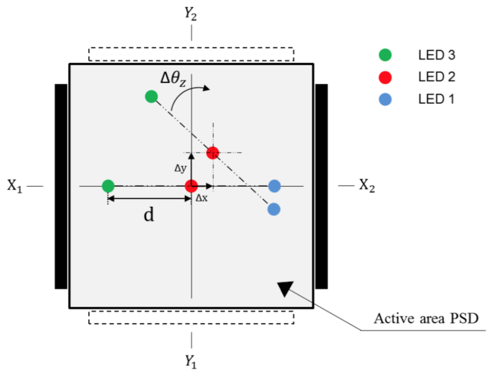
The three LED's in the lower part of the mover together with the Position Sensitive Detector (PSD) form the sensor system. The PSD consists of a uniform resistive layer on both surfaces of a semiconductor substrate. When a light beam is projected, a photopotential is produced proportional to the location of the centroid of the light spot. The position signal is extracted via pairs of electrodes on the ends of the resistive layers. This phenomenon is called the lateral photo effect (LPE). In Figure C.5a a picture of the PSD sensor is shown. The sensor concept is illustrated in Figure C.5b. First the middle LED is used to get the x/y coordinates of the mover. Then one of the side LEDs is used to determine the rotation of the stage. Only one LED can be on at the same time and the LEDs will be put on successively. The x/y coordinates of the centroid of a light spot can be calculated with Equations C.1 and C.2.

$$x = \frac{V_{x1} + V_{x2}}{V_{x1} - V_{x2}} \cdot \frac{l}{2} \tag{C.1}$$

$$y = \frac{V_{y1} + V_{y2}}{V_{y1} - V_{y2}} \cdot \frac{l}{2} \tag{C.2}$$



(a)



(b)

Figure C.5: (a) A duolateral two-dimensional PSD used in the final design, model: SiTek 2L10_SU65_SPC01. (b) Sensor concept for obtaining x/y/ϕ_z

D

Derivation of the minimum number of bits

The propagation of the quantization error to the measured position depends on how the position is calculated from the sensor signals. The PSD sensor workings are explained in depth in APPENDIX A and shows that the position can be calculated with the following equation:

$$x(U_{diff}, U_{sum}) = \frac{l}{2} * \frac{U_{diff}}{U_{sum}} \tag{D.1}$$

, where x represents the position, l the width of the sensor and U_{diff} and U_{sum} the measured sensor signals. The possible measured positions range from $-\frac{l}{2}$ to $\frac{l}{2}$, which are the boundaries of the sensor and are obtained when$U_{diff} = -U_{sum}$ and $U_{diff} = U_{sum}$ respectively. The possible values for U_{diff} ranges therefore from $-U_{sum}$ to U_{sum}. Ideally this range spans over the full $3.3V$ range of the Arduino ADC and the sensor signals should be amplified, so that $U_{sum} = 1.65V$.

How the calculated position changes with respect to changes in the measured signals requires partial differentiation. The change Δx, as the result of a change ΔU_{diff} is obtained by partial differentiation of x to U_{diff} multiplied by the change ΔU_{diff}. The change Δx, as the result of a change ΔU_{sum} is obtained by partial differentiation of x to Usum multiplied by the change ΔU_{sum}. Adding both gives the relation between the change in x, due to changes in the measured potentials. This shown in Equation D.2 and worked out in Equation D.3.

$$x = \frac{\mathrm{d}x}{\mathrm{d}U_{diff}} \Delta U_{diff} + \frac{\mathrm{d}x}{\mathrm{d}U_{sum}} \Delta U_{sum} \tag{D.2}$$

$$\Delta x_{max} = \frac{l}{2} * \frac{1}{U_{sum}} * \Delta U_{diff} - \frac{l}{2} * \frac{U_{diff}}{U_{sum}^2} \tag{D.3}$$

Substitution of $l = 1$cm, $U_{sum} = 1.65V$ and the maximum quantization error of $\frac{lsb}{2}$ for U_{diff} and U_{sum} in Equation D.3 results in:

$$\Delta x_{max} = \frac{1e-2}{2} * \frac{lsb}{2}\left(\frac{1}{1.65} + \frac{1.65}{1.65^2}\right) \tag{D.4}$$

This shows that the measured position error depends on the resolution of the ADC and U_{diff}. The maximum error is obtained when $U_{diff} = -1.65V$, which means that the center of the lightspot is on the negative boundary. Substitution gives:

$$\Delta x_{max} = \frac{1e-2}{4} * lsb * \left(\frac{1}{1.65}\frac{1.65}{1.65^2}\right) = 3e-3 * lsb \tag{D.5}$$

Assuming an ideal positioning system with a complementary sensitivity function with the value 1 across the whole frequency range, results in a position error and measurement error of equal magnitude. The number of required bits can now be estimated, when all other errors and disturbances are neglected. The maximum value of the least significant bit of 66.7µV is calculated with substitution of $\Delta xmax = 200nm$ in Equation D.5. The lsb and number of bits are related with:

$$lsb = \frac{fullvoltagerange}{2^n} \tag{D.6}$$

, where the full voltage range is the Arduino analog input range of 3.3V and n is the number of bits. this results in a minimum number of 50,000 output words or n=16 bits.

E Ferrofluid Damping Coefficients

This Appendix is a copy of Appendix B in Len van Moorsels thesis [41], but modified for the parameters of Haris' Stage[22].

E.1. Translation

The translational damping coefficient c of a ferrofluid bearing can be calculated according to [29]:

$$c = \eta_r \eta_b \frac{A_{bearing}}{h_{bearing}} \tag{E.1}$$

where η_r is dependent on the velocity profile of the fluid and is approximately 4 in case of trail formation. η_b is the ferrofluid viscosity, $h_{bearing}$ is the fly height of the bearing and $A_{bearing}$ the total contact area of the bearing:

$$A_{bearing} = 2 * w_{ferrofluid}(\pi(D_{outer} + D_{inner}) + m\frac{D_{outer} - D_{inner}}{2}) \tag{E.2}$$

where $w_{ferrofluid}$ is the width of each ferrofluid line segment, D_{outer} and D_{inner} are the outer and inner diameter of the magnet ring and m is the number of poles and the 2, because the stage has 2 layers of magnets.With, $\eta_r = 4$, $\eta_b = 0.25 Pas$, m=8, $D_{outer} = 43mm$, $D_{inner} = 29mm$, and assuming a fly height of $h_{bearing}$ = 100µm and a ferrofluid line thickness of $w_{ferrofluid} = 3mm$, a damping of $c = 17Ns/m$ is obtained.

E.2. Rotation

The torque T needed to overcome rotational damping can be calculated according to [29]:

$$T = \int\int_S \tau r^2 dr d\phi = \int\int_S \eta \frac{rw}{h} r^2 dr d\phi = \eta \frac{w}{h} \int\int_S r^2 dr d\phi = \eta \frac{w}{h} J = \eta_r \eta_b \frac{w}{h} J \tag{E.3}$$

where $\int\int_S$ is the surface integral, defined by the radius r and the angle ϕ, w is the angular velocity of the bearing and J the polar moment of inertia of the ferrofluid contact. The rotational damping coefficient C_{rot} can thus be calculated with:

$$C_{rot} = \frac{T}{w} = \eta_r \eta_b \frac{J}{h} \tag{E.4}$$

For a mover with m magnet poles, the polar moment of inertia consists of three terms:

$$J = J_{inner} + m * J_{line} + J_{outer} \tag{E.5}$$

where $J_{inner}, J_{line}, J_{outer}$ correspond to the inner ring, the m line segments and the outer ring of the bearing. Special care must be taken to decide which η_r is used for what part of the polar moment of inertia. When the disk rotates around its centre, the ferrofluid at the inner and outer ring has a triangular velocity profile, corresponding to $\eta_r = 4$. This gives the following rotational damping coefficient:

$$C_{rot} = \frac{\eta_b}{h}(J_{inner} + m * 4J_{line} + J_{outer}) \tag{E.6}$$

Rings The polar moment of inertia of the inner and outer rings can be calculated with:

$$J_{ring} = \frac{\pi}{4}(r_2^4 - r_1^4) \tag{E.7}$$

This gives: $J_{inner} = 2.90 * 10^{-8}\mathrm{m}^4$ and $J_{outer} = 9.41 * 10^{-8}\mathrm{m}^4$.

Line segments To calculate the polar moment of inertia of the lines, the perpendicular axis theorem is used. It relates J_z to the area moments of inertia about the other two mutually perpendicular axes: I_x and I_y. $J_z = I_x + Iy$

$$I_x = \frac{((r_{outer} - r_{inner})w_{ferrofluid}^3}{12} \tag{E.8}$$

for the long axis of each line segment and

$$I_y = \frac{w_{ferrofluid}(r_{outer}^3) - r_{inner}^3}{3} \tag{E.9}$$

for the short axis of each line segment. Assuming $w_{ferrofluid} = 3mm$ gives a polar moment of inertia of each line segment of $J_{line} = 6.90 * 10^{-9}\mathrm{m}^4$. Assuming a bearing fly height of 100µm, a total rotational coefficient C_{rot} of $1.7 * 10^{-3}\mathrm{N\,m\,s\,rad^{-1}}$ is obtained.

F

Quantization Noise Theory[33]

In this appendix the increase of A/D converter performance will be discussed. First the theory behind the relation between the signal to noise ratio (SNR) and the amount of bits will be explained and how increasing the signal to noise ratio therefore increases the effective amount of bits. Then it will be showed how you can increase the SNR (or bits) with oversampling followed by the process of decimation.

F.1. Quantization error and noise

First we have to explain how the amount of bits of an ADC is related to the Signal to noise ratio, which is explained by the effects of having finite fixed-point binary word length. Using finite word lengths prevents us from representing values with infinite precision. Commercial ADC's are categorized by their output word lengths, which are normally in the range from 8 to 16 bits, and for the Arduino Due, the maximum number of bits is 12. The least significant bit (lsb) would represent:

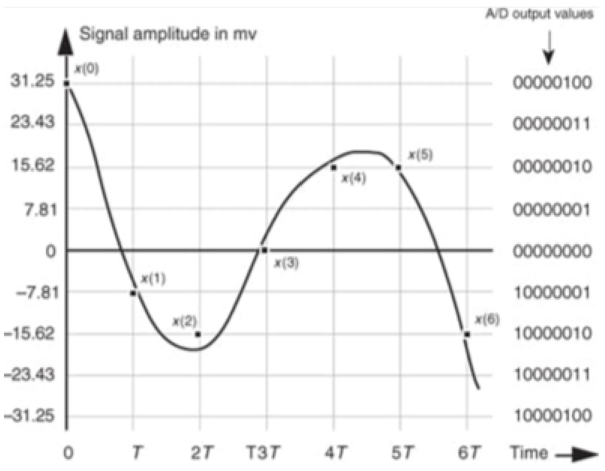
$$lsb = \frac{\text{full voltage range}}{2^n} = \frac{3.3V}{2^12} = 0.8\text{mV} \tag{F.1}$$

This means that we can represent continuous voltages perfectly as long as they are integral multiples of 0.8mV. Any other input value will result in an output word that is a best estimate digital data value. The inaccuracies in this process are quantization errors. For an ideal A/D converter, this error is a round-off error which is never greater than $\pm\frac{1}{2}lsb$. This is shown in Figure F.1, where Figure F.1(a) shows the signal, where the dots $x(n)$ represent the samples of the signal and Figure F.1(b) show the quantization errors.

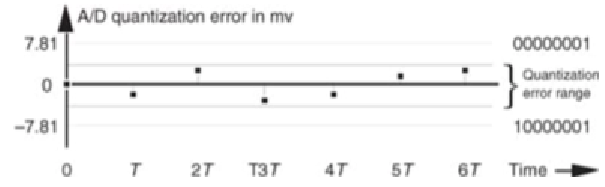
While Figure F.1(b) shows the noise in the time domain, we can also illustrate this noise in the frequency domain. Which is shown in Figure F.2. In Figure F.2(a) we see a perfect sine wave and the resulting samples. Notice how the dots are only integral values, which give a stair-step effect, oscillating above and below the unquantized signal. In (b) we see the discrete Fourier transform of the signal, where the values are not forced to take integer values, which with a single sine wave, results as expected in a single nonzero value in the spectrum at the first harmonic. In (c) the DFT of the quantized sine wave is shown of the 4-bit quantized sample, where quantization effects have induced noise components across the entire spectral band, with about equal magnitude over its entire spectral range. Intuitively these additional bands make sense, because we go from a perfect sinewave to a more 'cornery' sinewave, which indicate additional spectral components.

Another thing to notice in both fig 1b and 2c is that the error seems random, which it is. Even though the quantization noise is random, we can still quantify its effects in a useful way. In the field of communications, people often use the notion of output signal-to-noise ratio, or SNR=(signal power)/(noise power), to judge the usefulness of a process or device. We can do likewise to obtain an important expression for the output SNR of an ideal ADC, SNR_{ADC}, accounting for finite word-length quantization effects. Because quantization noise is random, we can't explicitly represent its power level, but we can use its statistical equivalent of variance to define SNR_{ADC} measured in dB as:

$$SNR_{ADC} = 10log_{10}(\frac{\text{input signal variance}}{\text{ADC quantization noise variance}}) = 10\log_{10}(\frac{\sigma^2_{signal}}{\sigma^2_{ADC}}) \tag{F.2}$$



(a) Digitized x(n) values of a continuous signal



(b) Quantization error between the actual analog signal values and the digitized signal values

Figure F.1: Quantization errors[33]

A depiction of the likelihood of encountering any given quantization error value, called the probability density function p(e) of the quantization error is shown in Figure F.3.

It should be noted however, that this even distribution is only true if the rounding error in the samples is uncorrelated. It can be easily shown that when a sinewave is sampled with exactly the same frequency as the period of the sinewave, exactly the same error will occur in all samples. If this criterion is met however, and the probability density function is as above, this simple rectangular function has much to tell us. It indicates an equal chance for any error value between $-\frac{q}{2}$ and $\frac{q}{2}$ to occur. By definition, because probability density functions have an area of unity, the amplitude of the p(e) density function must be the area divided by the width, or $p(e) = \frac{1}{q}$. From this the variance of our uniform p(e) is:

$$\sigma^2_{A/Dnoise} = \int_{-q/2}^{q/2} e^2 p(e) de = \frac{1}{q}\int_{-q/2}^{q/2} e^2 de = \frac{q^2}{12} \tag{F.3}$$

So now we have the Noise variance. To arrive at a general result for the SNR, lets express the input signal in terms of root mean square(rms), the A/D converter's peak voltage and a loading factor LF defined as:

$$LF = \frac{\text{rms of the input signal}}{V_p} = \frac{\sigma_{signal}}{V_p} \tag{F.4}$$

Which is the ratio of the amount of scale we use. This is relevant, because when a constant noise amplitude is present, the lower the voltage of the signal, the harder it will be to distinguish it from the noise. Squaring and rearranging gives:

$$\sigma^2_{signal} = (LF)^2 V_p^2 \tag{F.5}$$

Which gives an expression for the signal to noise ratio:

$$SNR_{A/D} = 10\log_{10}\left(\frac{(LF)^2 V_p^2}{V_p^2/(3*2^{2b})}\right) = 10*\log_{10}[(LF)^2(3*2^{2b})] \tag{F.6}$$

$$= 6.02*b = 4.77 + 20\log_{10}(LF) \tag{F.7}$$

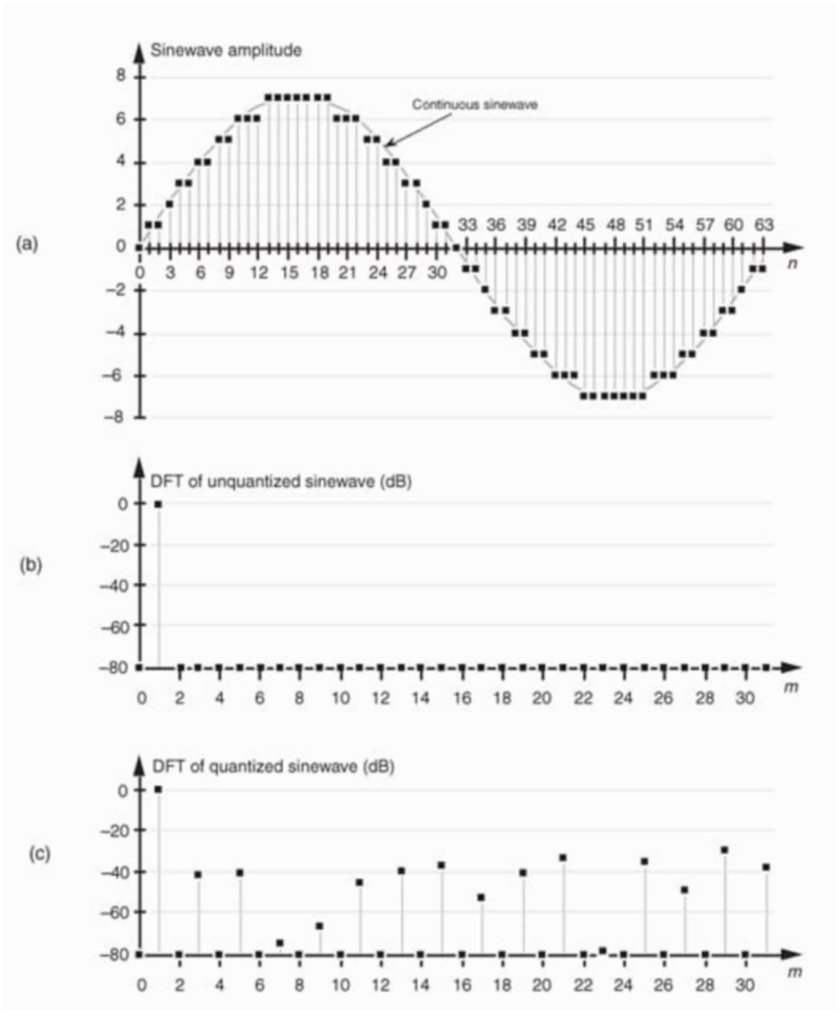


Figure F.2: Quantization noise effects: (a) input sinewave applied to a 64-point DFT; (b) theoretical DFT magnitude of high-precision sinewave samples; (c) DFT magnitude of a sinewave quantized to four bits.

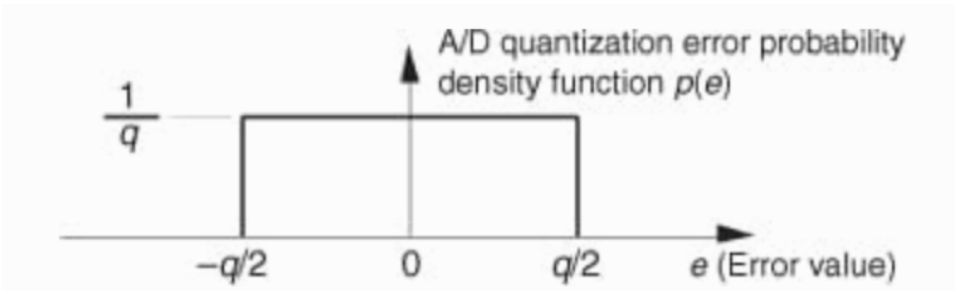


Figure F.3: Probability density function of A/D conversion roundoff error (noise).

This equation gives us the SNR of an ideal b-bit ADC in terms of the loading factor and number of bits. Note that increasing the ADC by 1 bit is equal to increasing the SNR by 6dB. Something else to be noticed is that when signals exceed a loading factor of 1 the ADC will clip and there is rapid SNR degradation. For sinusoidal inputs, this means that the rms value must not be greater than $\frac{Vp}{\sqrt{2}}$, which is 3 dB below Vp. The figure below plots different SNR values for different loading factors and different numbers of bits. A plot is shown below for the Signal to noise ratios for different resolutions.

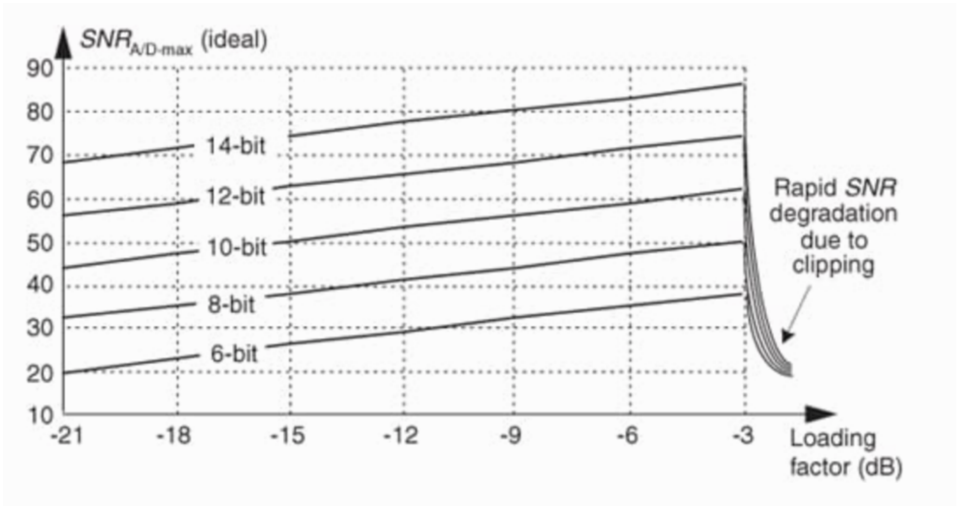


Figure F.4: $SNR_{A/D}$ of ideal A/D converters as a function of loading factor in dB.

A word of caution is appropriate here concerning our analysis of the ADC quantization errors. The derivations of equations are based upon three assumptions:

1. The cause of ADC quantization error is a stationary random process; That means that the performance of the ADC does not change over time. Given the same continuous input voltage, we always expect an ADC to provide exactly the same output binary code.
2. The probability density function of the A/D quantization error is uniform. We're assuming that the ADC is ideal in its operation and all possible errors between -q/2 and q/2 are equally likely. An ADC having stuck bits or missing output codes would violate this assumption. High quality ADC's being driven by continuous signals that cross many quantization levels will result in our desired uniform quantization noise probability density function.
3. The ADC quantization errors are uncorrelated with the continuous input signal. If we were to digitize a single continuous sinewave whose frequency was harmonically related to the ADC's sample rate, we'd end up sampling the same input voltage repeatedly and the quantization error sequence would not be random. The quantization error would be predictable and repetitive, and our quantization noise variance derivation would be invalid. In practice, complicated continuous signals such as music or speech, with their rich spectral content avoid this problem. In positioning extra caution has to be taken because ideally when the stage is being hold in the same position, a DC signal is obtained if perfectly hold. The validity of this correlation criterion is therefore dependent on the amount of noise in the signal. If not enough noise is present in the system, this can be added voluntarily. This process is called dithering.

F.2. Reducing the ADC's quantization noise

Now that the theory behind the relation between the number of bits and the quantization noise are out of the way, the next step is to reduce the amount of quantization noise to increase the effective number of bits of the ADC. To reduce the quantization noise in the ADC, two tricks are used to reduce the converter's quantization noise. Those schemes are called oversampling and dithering.

F.2.1. Oversampling

The process of oversampling to reduce A/D converter quantization noise is straightforward. The analog signal is sampled at a sample rate fs, that is higher than the minimum rate needed to satisfy the Nyquist criterion (twice the analog signal's bandwidth), and then lowpass filter. The theory behind oversampling is based on the assumption that an ADC's total quantization noise power (variance) is the converter's least significant bit (lsb) value over 12, or

$$\text{total quantization noise power} = \sigma^2 = \frac{\text{lsb}^2}{12} \tag{F.8}$$

Which has been derived before. The next assumptions are: The quantization noise values are truly random, and in the frequency domain the quantization noise has a flat spectrum. A truly random error, an error without any time correlation (basically a bunch of impulses), is white noise and has a flat spectrum in the frequency domain (Fourier transform of impulses). This noise spectrum reaches far beyond the Nyquist frequency, but because of aliasing, this will fold back onto the spectrum within the Nyquist frequency, resulting in the full white noise power evenly distributed in the region of $\pm\frac{1}{2}fs$(half the sampling frequency).

Next, we consider the notion of quantization noise power spectral density (PSD), a frequency-domain characterization of quantization noise measured in noise power per hertz as shown in figure below. Thus, we can consider the idea that quantization noise can be represented as a certain amount of power (watts) per unit bandwidth. In discrete systems, the flat noise spectrum assumption results in the total quantization noise, which is a fixed value based on the ADC's lsb voltage, being distributed equally in the frequency domain, from $-\frac{fs}{2}$ to $+\frac{fs}{2}$ as indicated in the figure below.

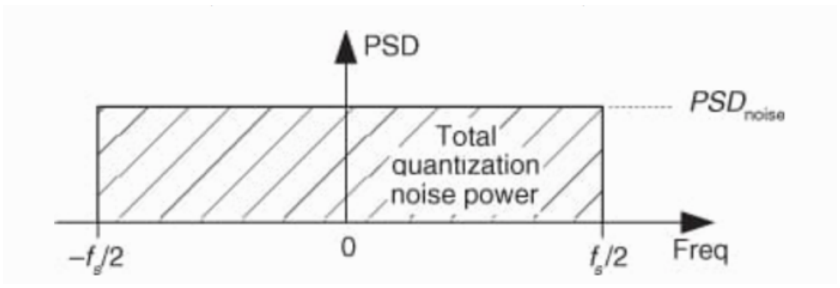


Figure F.5: Frequency-domain power spectral density of an ideal A/D converter.

The amplitude of this quantization noise PSD is the rectangle area (total quantization noise power) divided by the rectangle width (fs) or

$$PSD_{noise} = \frac{lsb^2}{12}\frac{1}{f_s} = \frac{lsb^2}{12f_s} \tag{F.9}$$

Measured in watts/Hz. So how do we now reduce the noise? The first thing we can do is reduce the lsb value, or increase the number of bits. This will certainly reduce the noise, because noise power will be reduced, which is easily observed in the figure, because the area will decrease. This requires us to increase the bits of the ADC however, which is an expensive solution. Extra converter bits cost money.

The other way we can decrease the amount of quantization noise is by considering the denominator. By increasing the sampling frequency, the total quantization noise will be spread over a larger frequency range. This in itself will not reduce the quantization noise, it will only spread it out over a larger frequency range, shown in figure 6(b). However, when we spread the quantization noise over a frequency range that reaches beyond the range of the signal of interest, we can lowpass filter the signal in such a way that the signal of interest is not affected, and noise in the frequency range beyond the range of interest will be attenuated. Thereby reducing the total quantization noise. After filtering, the digital signal can be downsampled back to twice the Nyquist frequency. In this process it's important that the downsampled signal uses more bits to represent the samples, otherwise another quantization error will be introduced, which negates the whole process. The techniques of low pass filtering and downsampling combined is called decimation.

Increasing the sampling frequency by a factor of 2 will reduce the quantization noise power by half, which equals -3db. As shown previously, increasing an ADC by 1 bit, will reduce the noise by 6db, which means that increasing the effective amount of bits, by oversampling, requires us to oversample by a factor of 4, which results in the formula:

$$f_{os} = 4^w * fs \tag{F.10}$$

Where w is the increase in effective bits and fs is the sampling frequency. Another advantage of increasing the sampling frequency is that an analog anti-aliasing filter with a lower performance can be used, because the Nyquist frequency is increased. Because of this the filtering is shifted to the digital domain.

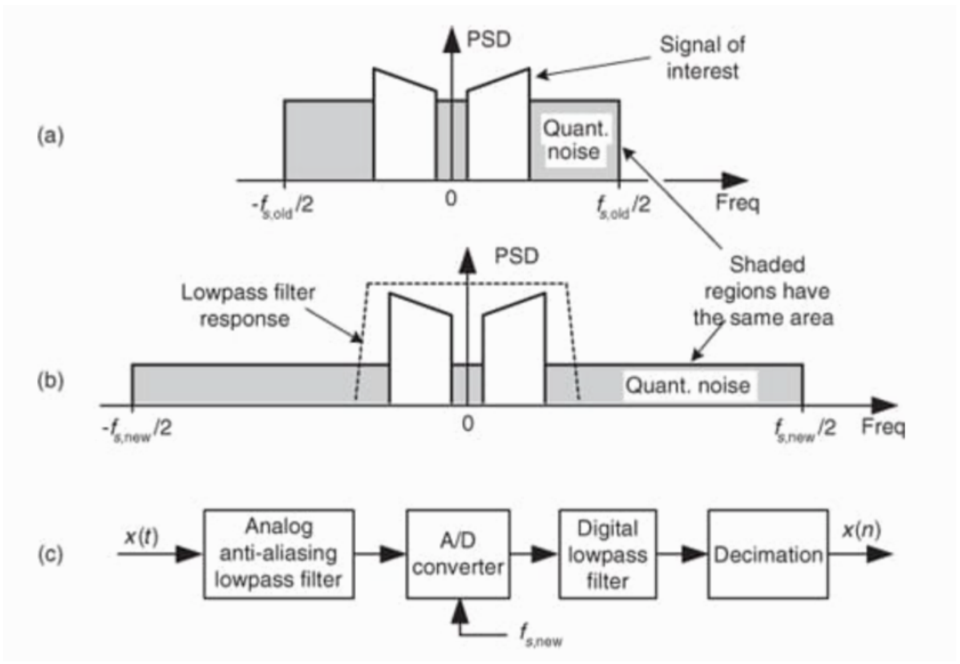


Figure F.6: Oversampling example: (a) noise PSD at an $f_{s,old}$ samples rate; (b) noise PSD at the higher fs,new samples rate; (c) processing steps.

F.3. Dithering

Another technique used to minimize the effects of ADC quantization noise, is the process of adding noise to our signal prior to the A/D conversion. This scheme, which doesn't seem at all like a good idea, can indeed be useful and is easily illustrated with an example. Consider digitizing the low-level analog sinusoid shown in the figure below, whose peak voltage just exceeds a single ADC's least significant bit voltage level, yielding the converter output samples in figure b.

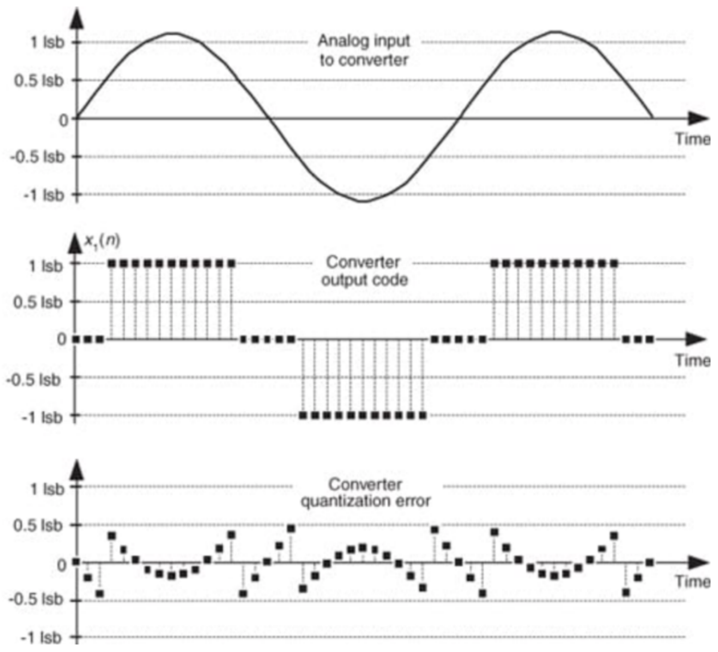


Figure F.7: Dithering: (a) a low-level analog signal; (b) the A/D converter output sequence; (c) the quantization error in the converter's output.

The output sequence is clipped. This generates all sorts of spectral harmonics. Another way to explain the spectral harmonics is to recognize the periodicity of the quantization noise. Because the quantization noise is highly correlated with our input sinewave (the quantization noise has the same time period as the input sinewave), the quantization noise is not equally distributed and the noise level of certain harmonics will be higher. Dithering is the technique where random noise is added to the analog signal, this technique results in a noisy analog signal that crosses additional converter lsb boundaries and yields a quantization noise that's much more random. Dithering forces the quantization noise to lose its coherence with the original input signal. Even though the noise floor is increased by adding extra noise, the SNR is increased in the cases of:

- low-amplitude analog signals
- Highly periodic analog signals
- Slowly varying (including DC) analog signals

The effect of dithering is shown in the figure below: In standard implentations the random wideband

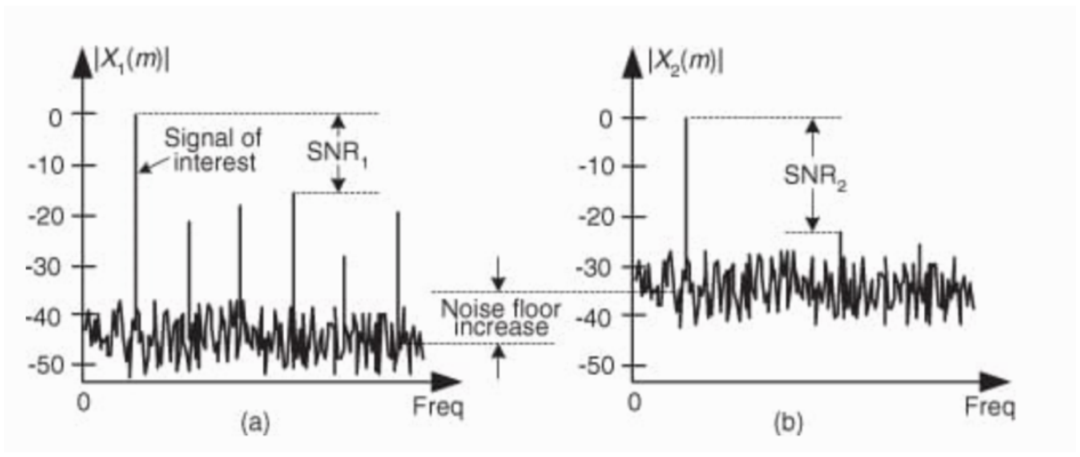


Figure F.8: Spectra of a low-level discrete sinusoid: (a) with no dithering; (b) with dithering.

analog noise used in this process, provided by a noise diode or noise generator IC, the rms level is equivalent to 1/3 to 1lsb.

G Disturbances

G.1. Floor vibrations

G.1.1. Amplitude

The analysis of the floor vibrations is based on an extensive study done in [38]. The study is summarized in Figure G.1, which contain the measured quasi-steady floor vibrations at various manufacturing plants in the former Soviet Union (broken lines) in the 1960's and recent measurements in the U.S(dots). From the data an approximation can be made for general quasi-steady floor vibrations in manufacturing plants, which is drawn with the solid line.

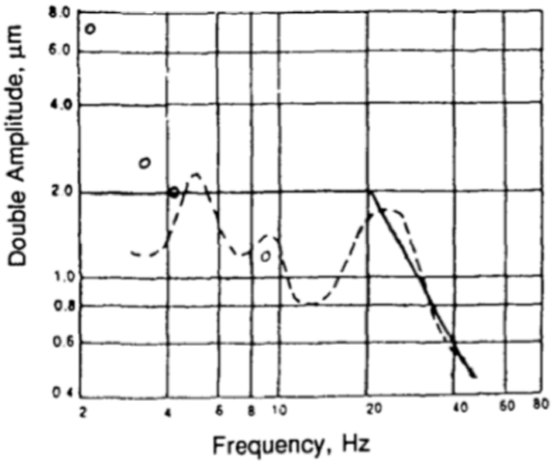


Figure G.1: Schematic of a simplified precision system without control.

The solid line has the shape of a lowpass filter with slope -3, with a cut-off frequency at 20Hz. Therefore, the magnitude of the vibrations can be approximated with the transfer function of a 3rd order Butterworth lowpass filter. The normalized transfer function of a 3rd order Butterworth low pass filter is:

$$Butterworth_{3rd}(s) = \frac{1}{(s+1)(s^2+s+1)} \tag{G.1}$$

With a cut-off frequency of $\omega = 40\pi$ rad/s and an amplitude of 1um, this gives the transfer function:

$$N_{xfloor}(s) = \frac{1e-6}{(\frac{s}{40\pi}+1)(\frac{s}{40\pi}^2+\frac{s}{40\pi}+1)} \tag{G.2}$$

Plotting this in MATLAB gives the proper magnitude plot, shown in figure G.2.

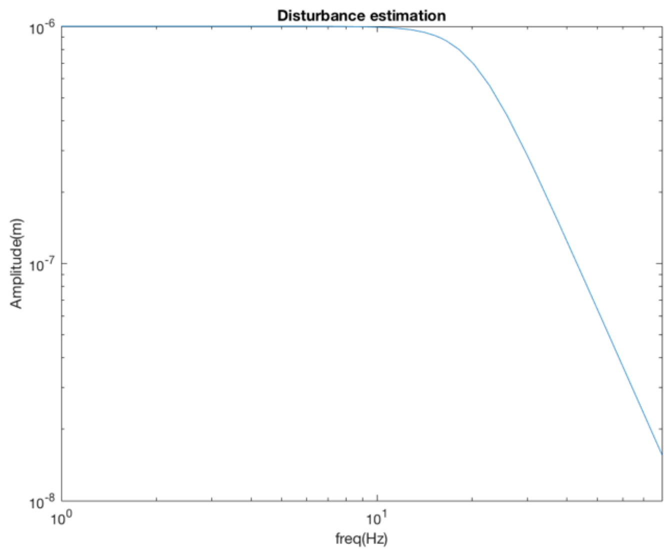


Figure G.2: Floor vibration amplitude estimation with 3rd order low pass filter.

From this we can easily calculate the power spectral density, which is the Amplitude squared, shown in fig G.3. Power refers to the fact that the magnitude of the PSD is the mean-square value of the signal being analyzed. It does not refer to the physical quantity power (as in watts or horsepower). But since power is proportional to the mean-square value of some quantity (such as the square of current or voltage in an electrical circuit), the mean-square value of any quantity has become known as the power of that quantity.

$$PSD = \left|N_{xfloor}(s)\right|^2 = \left|\frac{1e-6}{(\frac{s}{40\pi}+1)(\frac{s}{40\pi}^2+\frac{s}{40\pi}+1)}\right|^2 \tag{G.3}$$

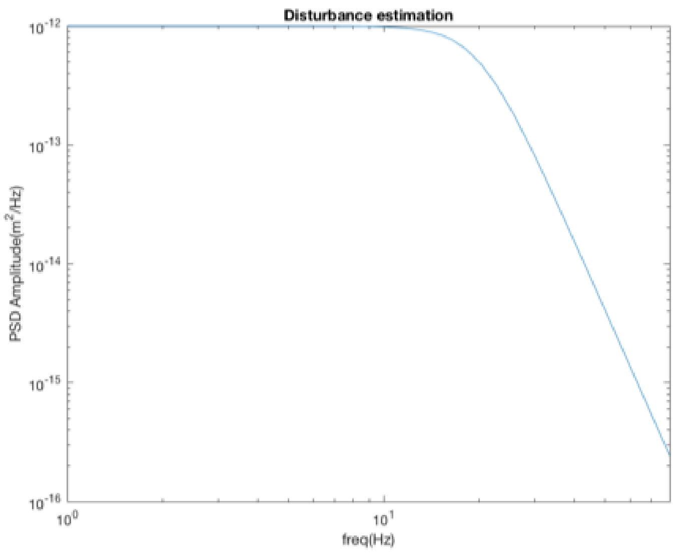


Figure G.3: Power spectral density of the floor vibrations.

Integrating the PSD over all frequencies gives the total power of the vibrations, which for a zero mean signal is the same as its variance.

$$Variance = \sigma^2 = \mathbb{E}\{\left|N_{xfloor}(t) - \mu\right|^2\} = \{\left|N_{xfloor}(t)\right|^2\} = \{\left|N_{xfloor}(s)\right|^2\}$$
$$= \int_0^{\inf} \left|N_{xfloor}(s)\right|^2 df = 5 * 10^{-10}\mathrm{m}^2 \tag{G.4}$$

When the probability density function of the amplitude is assumed to be normally distributed, the probability of an error occurring within a certain range is easily deducted. The standard deviation is simply the root of the variance.

$$\sigma = \sqrt{variance} = 23\mu\mathrm{m} \tag{G.5}$$

Using the “68–95–99.7” rule[13], this means a probability of 95% that the amplitude of the floor vibrations is smaller than 46µm and a probability of 99.7% that the amplitude is smaller than 69µm.

G.1.2. Acceleration

This means that Control is required for disturbance rejection to get the required 1 um. To get an estimate of what control bandwidth is required, we have to define the noise in accelerations. The accelerations can be obtained by differentiating the position twice. The is very easy to do in the la Place domain, because differentiating in the time domain is the same as dividing by s in the la Place domain. Dividing twice gives the following transfer function for the acceleration of the noise:

$$N_{afloor}(s) = \frac{1}{s^2} \frac{1e-6}{(\frac{s}{40\pi}+1)(\frac{s}{40\pi}^2 + \frac{s}{40\pi}+1)} \tag{G.6}$$

With the PSD:

$$PSD = \left|N_{afloor}(s)\right|^2 = \left|\frac{1}{s^2} \frac{1e-6}{(\frac{s}{40\pi}+1)(\frac{s}{40\pi}^2 + \frac{s}{40\pi}+1)}\right|^2 \tag{G.7}$$

Variance:

$$\sigma^2 = \int_0^{\inf} \left|N_{afloor}(s)\right|^2 df = 3.68e-4\mathrm{m}^2/\mathrm{s}^2 \tag{G.8}$$

Which gives a 2σ of about 38mm/s^2 and 3σ = 58mm/s^2.

G.2. PWM Amplifier disturbances for acting as a voltage source

Using PWM and an H-bridge instead of a linear current amplifier leads to extra harmonic content at the fundamental frequency of the pulse wave and at its harmonics as explained in chapter 4. Other than the extra frequency content of the pulse, an PWM based amplifier also acts as a voltage source instead of a current source. In simple terms this means that there is no current feedback and that changes in current due to changes in the circuit because of a temperature change and induced voltages are not compensated for. To see how much of a problem this is we have to look at the process disturbance transfer function, which is described by:

$$\frac{y}{d} = \frac{G}{1+GC} \tag{2.9}$$

, which depends on the transfer function of the plant and the transfer function of the controller. The transfer function of the plant is derived in Equation 4.2, leading to:

$$G(\omega) = \frac{1}{j\omega L + R} * \frac{1}{\omega^2 m + j\omega c} * Bl$$

Haris used the following tamed PID control parameters for the 90Hz original stage[22]: $K_p = 7.25 \cdot 10^3$, $K_i = 4.35 \cdot 10^5$, $K_d = 38.5$ and $T_f = 5.36 \cdot 10^{-4}$ leading to a control transfer funtion of:

$$C(s) = K_p + \frac{K_d s}{(T_f s + 1)} + \frac{K_i}{s} = 7.25 \cdot 10^3 + \frac{38.5s}{(5.36s+1)} + \frac{4.35 \cdot 10^5}{s} \tag{G.9}$$

Substitution of C and G in the process disturbance function gives the transfer function illustrated in Figure G.4. The attenuation is depends on the frequency leading to a minimum attenuation of $\pm 1 \cdot 10^{-4}\mathrm{mV}^{-1}$.

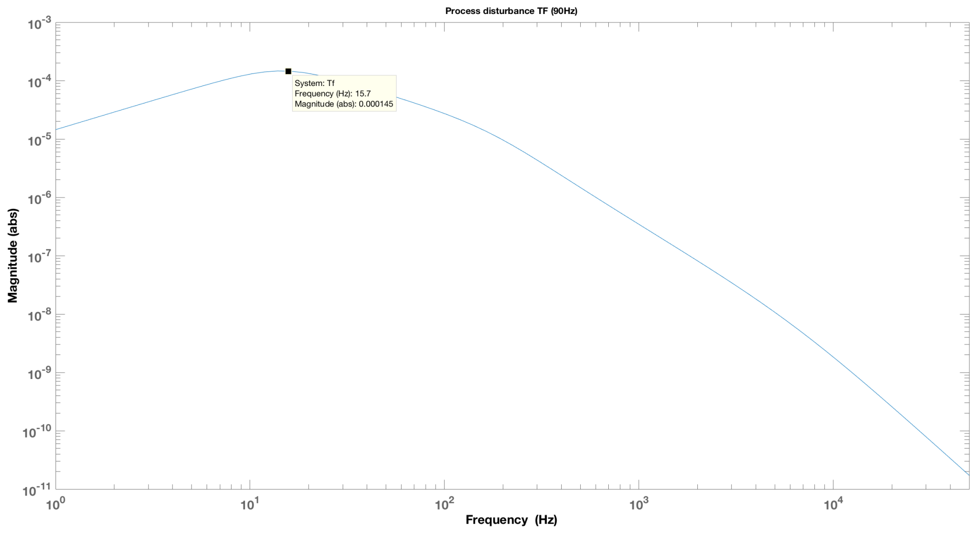


Figure G.4: Process disturbance transfer function.

G.2.1. Induced voltage due to movement of the stage

The two alternating magnetic rings of the mover lead to an alternating magnetic field. When the mover moves relative to the coils, the movement also leads to an alternating flux through the coils, which induces a voltage following faraday's law:

$$V_{induced} = -\frac{d\phi}{dt} = -\frac{d\phi}{dx}\frac{dx}{dt} = -\frac{d\phi}{dx}v \tag{G.10}$$

, where v is the velocity. To get an estimate about the induced voltage, we have to estimate the change in magnetic flux enclosed by the coil. To estimate the change in magnetic flux, the problem is simplified so that the surface enclosed by the coil is square and the width of a part of the magnet that has similar polarity is equal to the length of the side of the square, with a side lenght l_{coil}. This is illustrated in Figure G.5.

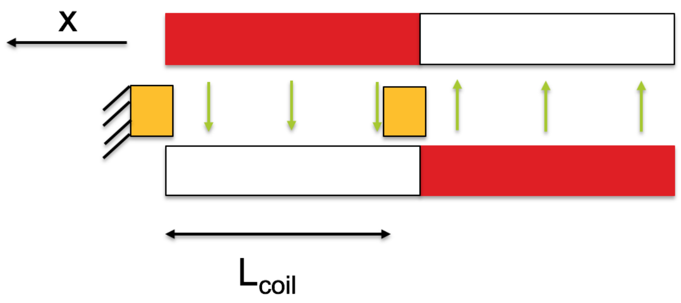


Figure G.5: Schematic of the Lorenz actuator, showing the magnetic field through the coils.

The magnetic field enclosed by the coil can then be calculated with:

$$\phi_{enclosed} = nBl_{coil}^2 \tag{G.11}$$

, with n the number of coil winding and B the magnetic field strength. If the magnet would move a length exactly equal to l_{coil}, the magnetic flux through the coil would be of equal magnitude but in opposite direction, illustrated in Figure G.6. The result can be substituted into Faraday's law:

$$V_{induced} = -\frac{d\phi}{dx}v = \frac{2nBl_{coil}^2}{l_{coil}}v = 2nBl_{coil}v \tag{G.12}$$

For $v_{max} = 1mm/s$, $B = 0.5T$, $l_{coil} = 1cm$ and $n = 50$, which are realistic values based on the geometry and the performance of the original stage designed by Haris Habib[22], Faraday's law gives an induced voltage of $V_{induced} = 5 \cdot 10^{-4} V$. Using the worst case scenario transfer function value of 10^{-4} this leads to a position error of 50nm. Although 50nm is a considerable amount when aiming for 200nm precision, it is based on the maximum speed of the stage, which is in general not a problem when the stage is hold in the same position. It does have to be taken into account when precision is required for a moving reference.

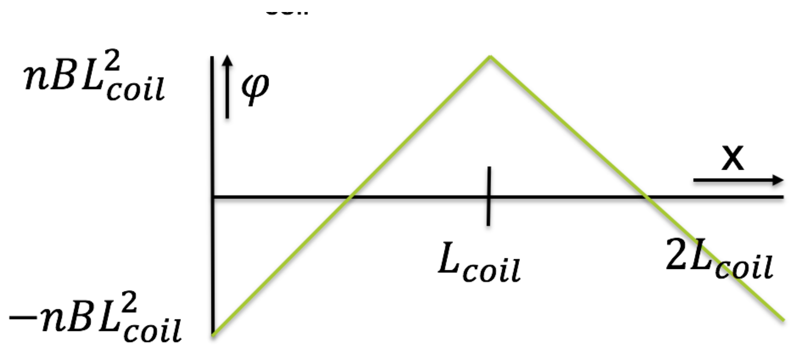


Figure G.6: The magnetic flux enclosed by the actuator.

G.2.2. The effect of a coil temperature change

Haris' included a coil temperature plot, which is shown in Figure G.7. From this figure we can approximate a slope of $\frac{dT}{dt} \approx \frac{35}{25} = 1.4°Cs^{-1}$ for 0.8A. The resistance of copper changes about 0.4% for each °C[17]. The frequency response of the process disturbance transfer function is worst at 15Hz. If the coil is switched on and off with a frequency of 15Hz, so a period of $\frac{1}{15}s$, the coil is on half the time, so $\frac{1}{30}s$. In this time the coil heats $\sim \frac{1}{30} \cdot 1.4°C$, leading to a resistance change of $\Delta R \approx \frac{1}{30} \cdot 1.4 \cdot 0.004\Omega$. However, this is peak to peak and should be divided by 2 to get the amplitude. The resistance of the coil is measured to be around 2.8Ω.This leads to an equivalent voltage change of:

$$\Delta V \approx \frac{1}{30}\frac{1}{2} \cdot 1.4 \cdot 0.004 \cdot 2.8 \cdot 0.8 \approx 2 \cdot 10^{-4} V \tag{G.13}$$

With a transfer function of 10^{-4} at 15Hz in Figure G.4, this leads to a position error amplitude of $2 \cdot 10^{-8} = 20nm$. Considering that a very bad scenario is used, this can most likely be neglected. Also a new copper plate is added in the new stage to reduce the coil temperature, which should also reduce this problem.

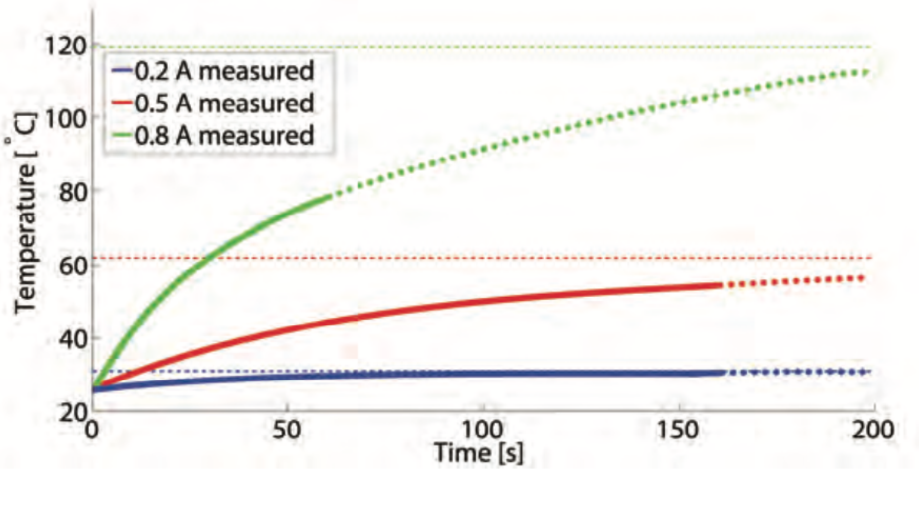


Figure G.7: The temperature of the coils over time for different currents driven through the coils.

H

Signal Processing Circuits

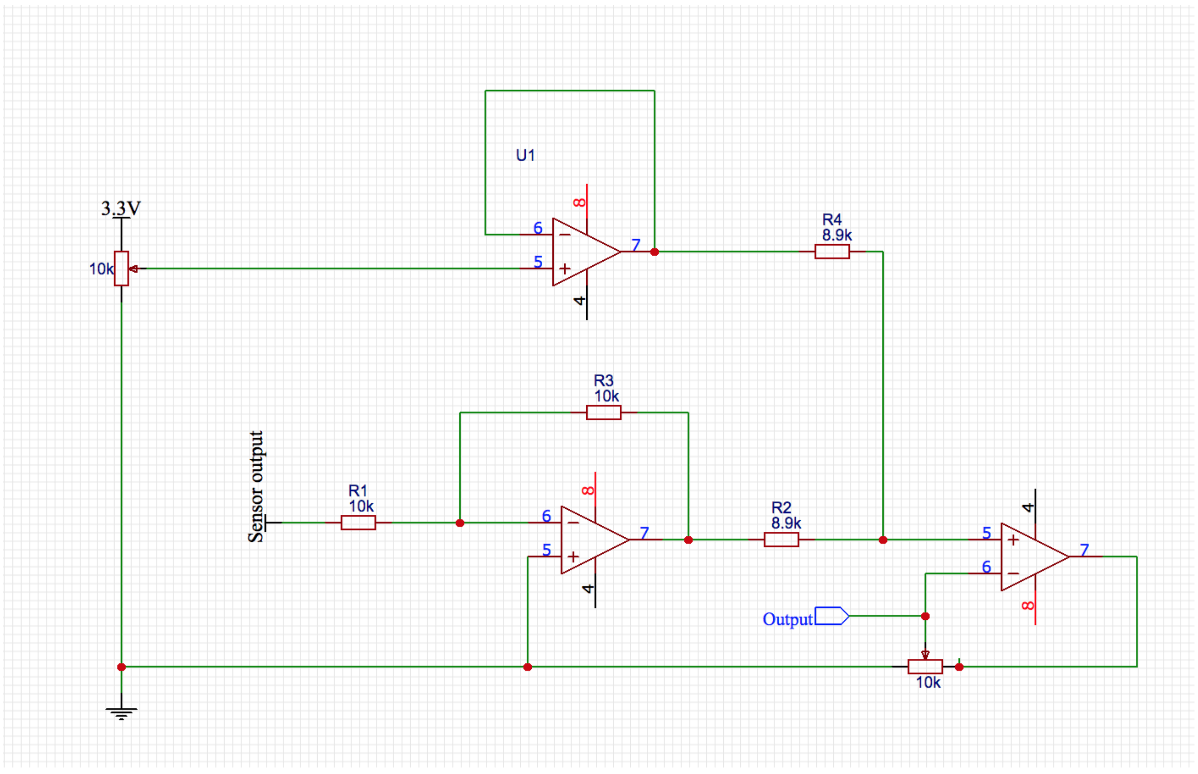


Figure H.1: Schematic of the sensor output signal processing circuit

I

Control Algorithm Code

This Appendix contains the implementation of the Arduino code. All the required code to control the stage is added.

I.1. Main file

I.1.1. Loading the libraries

```
//load libraries
#include <SPI.h>
#include <MatrixMathfloat.h>
#include "SimplePID.h"
#include "SimpleFIR.h"
```

I.1.2. Initiation of Constants, variables and classes

```
//25Hz
//initiate PID controllers (kp, kd, ki, tf, limit)
SimplePID Fx=SimplePID(512.9259, 5.662, 6978.9568, 0.000055915,1);
SimplePID Fy=SimplePID(512.9259, 5.662, 6978.9568, 0.000055915,1);
SimplePID Tz=SimplePID(0.087, 0.00024718, 0.050885, 0.0,100000);

//initiate FIR filter coefficients and classes
float FIR_coef[4]{0.144665150707102,  0.352275583919487,
0.352275583919487, 0.144665150707102};
float FIR_coef2[10]{0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1};
SimpleFIR xMidFIR=SimpleFIR(4,FIR_coef);
SimpleFIR yMidFIR=SimpleFIR(4,FIR_coef);
SimpleFIR xSideFIR=SimpleFIR(4,FIR_coef);
SimpleFIR ySideFIR=SimpleFIR(4,FIR_coef);
SimpleFIR angleFIR=SimpleFIR(10,FIR_coef2);

SimpleFIR xrefFIR=SimpleFIR(10,FIR_coef2);
SimpleFIR yrefFIR=SimpleFIR(10,FIR_coef2);
SimpleFIR arefFIR=SimpleFIR(10,FIR_coef2);


//initiate constants
constexpr int oversampling=16; //oversampling factor
constexpr int Adiffx=5; //the Arduino port nr connected to the diffx Sensor output
constexpr int Asumx=4;//the Arduino port nr connected to the sumx Sensor output
constexpr int Adiffy=7;//the Arduino port nr connected to the diffy Sensor output
constexpr int Asumy=6;//the Arduino port nr connected to the sumy Sensor output
constexpr int ledMid=3; //Arduino port connected to the middle LED
constexpr int ledLeft=2; //Arduino port connected to the left LED
constexpr int ledRight=5; // "" right LED


//required variables for Direct Memory Access
//what Arduino calls port 1, is port 7 for microprocessor.
//Make sure diffx etc are between 0 and 7!
int channelnrs[4]{7-Adiffx,7-Asumx,7-Adiffy,7-Asumy};
volatile int counter;
volatile int bufnr;
uint16_t buffa[4][256];    // 4 buffers of 256 readings


//inititate global variables
float offset=0.0;
float posMid[2];
float posLeft[2];
float posRight[2];
float posSide[2];
```

I.1.3. Setup

This code runs once when at Arduino startup

```
void setup() {
// put your setup code here, to run once:
Serial.begin(115200); // begin serial communication
SPI.begin(10);  //begin SPI communication on arduino port 10
analogReadResolution(12); //set ADC resolution

//set the Pins used for the LEDs to output ports and low
pinMode(ledMid, OUTPUT);
pinMode(ledRight, OUTPUT);
pinMode(ledLeft, OUTPUT);
digitalWrite(ledMid,LOW);
digitalWrite(ledLeft,LOW);
digitalWrite(ledRight,LOW);

setupPWM();// function to setup PWM registers
setResistors(); // function to set variable resistors in the signal processing circuits
setupADC();   // function to setup the ADC registers
offset=getAngleOffset(); //function get the angle offset when using the left vs right led to get the angle
}
```

I.1.4. Loop

This part of the program will run over and over again. One function is used to move the stage to certain position.

```
//this will loop forever
void loop() {
moveTo(−0.3/1000, −0.3/1000, 0.0/180*PI,1); //function to move to a position
}
```

I.1.5. All the required functions

Setting the variable resistors of the signal processing circuit :

```
void setResistors(){
constexpr int bits=pow(2,12); //total number of integers used by the ADC
int i=0;
digitalWrite(ledMid,HIGH); //put on middle LED

//set inital resistance of variable resistors
SPI.transfer(10,0,SPI_CONTINUE);
SPI.transfer(10,i);

delay(10);
//read out the sumx and sumy values of the sensor
int sumx=analogRead(Asumx);
int sumy=analogRead(Asumy);

//iterate on resistance until the sensor spans 90% of the ADC range, from 5% till 95%
while ((sumx>0.05*bits)&(sumy>0.05*bits)&(sumx<0.95*bits)&(sumy<0.95*bits)){
i++;
SPI.transfer(10,0,SPI_CONTINUE);
SPI.transfer(10,i);

delay(10);

sumx=analogRead(Asumx);
sumy=analogRead(Asumy);

}
digitalWrite(ledMid,LOW); //put off middle LED

}
```

Setup ADC registers :

```
void setupADC(){
//for procedure read Atmel sam3x data sheet
ADC->ADC_CR=1; //reset ADC
analogReadResolution(12);
pmc_enable_periph_clk(ID_ADC);
adc_init(ADC, SystemCoreClock, ADC_FREQ_MAX, ADC_STARTUP_FAST);

ADC->ADC_CHDR=0xFFFF; //disable all channels
ADC->ADC_MR |=0x80;// free running
NVIC_EnableIRQ(ADC_IRQn);
ADC->ADC_PTCR=1; // pdc transfer control register

}
```

Setup PWM registers :

```
void setupPWM(){
//setup PWM, check datasheet
PMC->PMC_PCER1 |= PMC_PCER1_PID36;                    // enable PWM clock

REG_PIOC_ABSR |= 0x000C02A8;                          // Set PWM pin perhipheral type
REG_PIOC_PDR |= 0x000C02A8;                        // Disable PIO control and set Peripherial control

REG_PWM_ENA |= 0x6F;                         // Enable the PWM channels (see datasheet page 973)
REG_PWM_CLK |= 0x00000000;                   // Set the PWM clock rate to maximum
REG_PWM_SCM |= 0x6F;                         // sync PWM channels
REG_PWM_CMR0 |= 0x101;                       // Set pwm mode, The period is center aligned, clock source as MCK/2
REG_PWM_CPRD0 = 1000;                        // set resolution

REG_PWM_CDTY0 = REG_PWM_CPRD0;                  //Set the PWM duty cycle registers to 0
REG_PWM_CDTY1 = REG_PWM_CPRD0;
REG_PWM_CDTY2 = REG_PWM_CPRD0;
REG_PWM_CDTY3 = REG_PWM_CPRD0;
REG_PWM_CDTY5 = REG_PWM_CPRD0;
REG_PWM_CDTY6 = REG_PWM_CPRD0;
}
```

Get angle offset between left and right LED :

```
float getAngleOffset(){
bool inposition=0;

// move stage to about middle
while (inposition==0) {
inposition=moveTo(1.0/1000, 1.0/1000, 0.0,0);

}
int timer=millis();

//keep it there for 6seconds so that it's stabilized
while(millis()-timer<6000) moveTo(1.0/1000, 1.0/1000, 0.0,0);


//get the position of the middle led
getPosFastWithReset(ledMid, oversampling, posMid);
float posxMid=posMid[0];
float posyMid=posMid[1];
//get the position of side LEDS
getPosFastWithReset(ledLeft, oversampling, posLeft);
float posxLeft=posLeft[0];
float posyLeft=posLeft[1];

getPosFastWithReset(ledRight, oversampling, posRight);
float posxRight=posRight[0];
float posyRight=posRight[1];

//get angle of both leds
float angle1=getAngle(posxMid, posyMid, posxRight, posyRight);
float angle2=getAngle(posxLeft, posyLeft, posxMid, posyMid);

//put off actuators
float currents0[3]{0,0,0};
CurrentToPWM(currents0, currents0);

//return the difference between the left and right led
return angle2-angle1;
}
```

Move to reference :

```
bool moveTo(float xref, float yref, float aref,bool initiated){
static float angle=0.0;
static int duration=micros();

//input filter to avoid spikes
xref=xrefFIR.filter(xref);
yref=yrefFIR.filter(yref);
aref=arefFIR.filter(aref);

//write the x/y coordinates of the middle LED to variables
getPosFastWithReset(ledMid, oversampling, posMid);
float xPos1=posMid[0];
//xPos1=xMidFIR.filter(xPos1); //filter x
```

```
float yPos1=posMid[1];
//yPos1=yMidFIR.filter(yPos1); //filter y


//choose which of the side leds to use, based on the position and the rotation in the last loop
int ledSide;
if (initiated==1) ledSide=chooseLED(xPos1, yPos1, angle, ledRight, ledLeft);
else ledSide=ledLeft;


//write the x/y coordinates of the side LED to variables
getPosFastWithReset(ledSide, oversampling, posSide);
float xPos2=posSide[0];
//xPos1=xMidFIR.filter(xPos1);
float yPos2=posSide[1];
//yPos1=yMidFIR.filter(yPos1);


// get the angle from the middle and side led positions
if (ledSide==ledLeft) {
angle=getAngle(xPos2, yPos2, xPos1, yPos1);
}
else{
angle=getAngle(xPos1, yPos1, xPos2, yPos2);
angle=angleOffset(angle, offset);

}
//angle=angleFIR.filter(angle); //filter angle

// calculate the forces with PID, using the PID class funtion
float Forcex=Fx.calculate(xref,xPos1);
float Forcey=Fy.calculate(yref,yPos1);
float Torquez;
if (initiated==1) Torquez=Tz.calculate(aref,angle);
else Torquez=0;
```

```
//transform forces to coil currents
float currents1[3]{};
float currents2[3]{};
ForceToCurrent(Forcex,Forcey,Torquez,angle,currents1,currents2);

//set the output voltages of the PWM ports
CurrentToPWM(currents1, currents2);


//this is only used in initiation to calculate the angle offset
if ((abs(xref-xPos1)<1e-3)&(abs(yref-yPos1)<1e-3)){
//Serial.print(xPos1,6);Serial.print('\t');Serial.println(yPos1,6);
return 1;
}
else return 0;

}
```

Setup ADC registers :

```
void getPosFastWithReset(int LED, int readings, float * pos){
//get x/y coordinates of sensor, with oversampling and filtering
//LED= which led to put on, readings=how many samples to take and filter,
//*pos= pointer vector for coordinates

constexpr float l= 10e-3; //sensor range in m
long int sum[4]{0,0,0,0}; //initialze sum array
double average[4];        //initialyze average array

digitalWrite(LED,HIGH);
int ledtimer=micros();

counter=readings; //set amount of samples to take
bufnr=0;          //reset buffer number
int readnr=0;     //reset buffer read number
```

```
ADC->ADC_RPR=(uint32_t)buffa[bufnr];// DMA buffer, set memory pointer
ADC->ADC_CHER=1<<channelnrs[bufnr]; // enable first ADC channel, from channelnrs array
int channelsettlingtimer=micros(); //set channel timer


while (micros()-ledtimer<5); //do nothing until required settling timers have passed.
while (micros()-channelsettlingtimer<5);

ADC->ADC_RCR=counter;  //set counter, this will automatically start filling buffers

//enable interupts, this will call the ADC_Handler when buffer is filled.
ADC->ADC_IER=1<<27;

//process buffers with averaging
while (readnr<4){   //Process buffers, untill all 4 buffers are processed
//if more buffers are filled than processed, process next buffer
if(bufnr>readnr){
//sum all values
for (int i=0;i<counter;++i) sum[readnr]=sum[readnr]+buffa[readnr][i];
average[readnr]=sum[readnr]/double(counter);// get the average

//offset, because the sensor analog range is mapped onto 0-3.3V
average[readnr]=average[readnr]-2048;
++readnr;
}
}

float diffx=average[0]; //write to variables
float sumx=average[1];
float diffy=average[2];
float sumy=average[3];


//calculate position and writhe to the array given at start of function
pos[0]=diffx/sumx*l/2;
pos[1]=diffy/sumy*l/2;
```

```
digitalWrite(LED,LOW);
}
```

Interrupt handler for ADC full buffer :

```
void ADC_Handler(){      // fills all 4 buffers
//interupt handler for full buffer, used by getPosFastWithReset function,
//switches automatically between sensor channels and fills buffers
//check atmel sam3x data sheet
int f=ADC->ADC_ISR;
if ((f&(1<<27))){                          //check end of buffer interupt
ADC->ADC_CHDR=1<<channelnrs[bufnr];     //disable old channel

if (bufnr<3){                           //if not the last channel


bufnr+=1;                              //increase buffernummer
ADC->ADC_RPR=(uint32_t)buffa[bufnr];  //set memory pointer
ADC->ADC_CHER=1<<channelnrs[bufnr];   //enable next channel
delayMicroseconds(5);                  //settling time

ADC->ADC_RCR=counter;//set counter, starts filling new buffer

}
else {                                 //if last buffer
ADC->ADC_IDR=1<<27;                   //disable interrupt
ADC->ADC_CHDR=1<<channelnrs[bufnr];  //disable channel
bufnr+=1;                             //increase buffernr
}
}
}
```

Choose which side LED to use :

```
int chooseLED(const float x, const float y, float angle, int right, int left){
float maxSensorValue=5.0; // maximum absolute sensor value for sensor from -5 to +5;
float cosPhi=cos(angle);
float sinPhi=sin(angle);
int signCos=(cosPhi>=0)-(cosPhi<0);
int signSin=(sinPhi>=0)-(sinPhi<0);

// distances from position to sensor edges
float dxRight=(maxSensorValue/signCos-x)/cosPhi;
float dyRight=(maxSensorValue/signSin-y)/sinPhi;
float dxLeft=(maxSensorValue/signCos+x)/cosPhi;
float dyLeft=(maxSensorValue/signSin+y)/sinPhi;

// choose LED based on which edge of the sensor is closest
if (min(dxRight,dyRight)<=min(dxLeft,dyLeft) ) return left;
else return right;
}
```

Calculate angle between 2 coordinates :

```
float getAngle(const float x1, const float y1, const float x2, const float y2){
//simple function to calculate the angle from the 2 led posisitions
float dy=y2-y1;
float dx=x2-x1;

float rad=sqrt(sq(dy)+sq(dx)); //pythogoras
float angle=acos(dx/rad);

if (dy<0) angle=-angle; // range of acos is only [0,pi]

return angle;
}
```

Add angle offset so that both side LED's give same angle and in the same range :

```
float angleOffset(float angle, float offset){
angle=angle+offset;
angle=angle+3*PI;
angle=angle - int(angle/(2*PI))*(2*PI); //modulo 2 PI
angle=angle-PI;

return angle;
}
```

Transform calculated x/y/z Forces to coil currents :

```
float ForceToCurrent(const float Fx,const float Fy,const float Tz, float angle,
float(&currents1)[3],float(&currents2)[3]){
//transform forces to the required coil currents
float forces[3][1]={Fx,Fy,Tz};

//physical constants of the stage required to calculate the current
constexpr float   currentAllowed=0.5;
constexpr float   B=0.35; //magnetic field
constexpr float   l=0.007*2; //twice the width of the magnet
constexpr float   windings=50.0; //windings
constexpr float   eta=sin(67.5/180*PI); // efficiency of actuator
constexpr float   Fa=B*l*windings*eta; // resultant force
constexpr float   r=(29+43)/4/float(1000);

//geometrical angles of the coils
//placement of the coil + 90 because the force is perpendicular to coil
constexpr float   offAngle1=(0.0)/180*PI; //make sure this is float division
constexpr float   offAngle2=(225.0)/180*PI;
constexpr float   offAngle3=(135.0)/180*PI;
constexpr float   offAngle4=(-67.5)/180*PI;
constexpr float   offAngle5=(67.5)/180*PI;
constexpr float   offAngle6=(157.5)/180*PI;

//how the transformation matrix values are calculated
//Bln*transformation matrix
//  constexpr float Trans1[3][3]={
//                    {cos(offAngle1),cos(offAngle2),cos(offAngle3)},
//                    {sin(offAngle1),sin(offAngle2),sin(offAngle3)},
//                    {r,r,r}};
```

```
//
//
//
//  constexpr float Trans2[3][3]={
//                          {cos(offAngle4),cos(offAngle5),cos(offAngle6)},
//                  {sin(offAngle4),sin(offAngle5),sin(offAngle6)},
//                  {r,r,r}};
//
//
//  Matrixx.Invert((float*) Trans1, 3);
//  Matrixx.Invert((float*) Trans2, 3);

//using actual numbers saves a lot of processing time
constexpr float Trans1[3][3]= {{0.585786437626905 , 4.41778381645398e-17 , 23.0118645762831},
{-0.292893218813453 , -0.707106781186547,  16.2718454896363},
{-0.292893218813453  ,0.707106781186548 ,16.2718454896363}};

constexpr float Trans2[3][3]= {{0.224170764583982,-0.541196100146197,23.0118645762831},
{0.541196100146197,0.541196100146197,16.2718454896363},
{-0.765366864730180,-4.05359937449976e-17,16.2718454896363}};

//initiate force arrays
float Fset1[3][1];
float Fset2[3][1];


//matrix multiplication for transformation
Matrixx.Multiply((float*) Trans1,(float*) forces,3,3,1,(float*) Fset1);
Matrixx.Multiply((float*) Trans2,(float*) forces,3,3,1,(float*) Fset2);


//commutate based on angle
float gain1=gain(angle,0);
float gain2=gain(angle,-22.5/180*PI);//gain of coilset 2 is 22.5 degrees behind


//get the maximum coil current
```

```
float currentMax{0};
for (int i=0;i<3;i++){
currents1[i]=gain1*Fset1[i][0]/Fa;
currents2[i]=gain2*Fset2[i][0]/Fa;

float intermediate=max(abs(currents1[i]),abs(currents2[i]));
currentMax=max(abs(currentMax),abs(intermediate));
}


//if one of the currents is above the max, normalize currents to maximum allowable current
if (currentMax>currentAllowed){
float normalize=currentAllowed/currentMax;
for (int i=0;i<3;i++){
currents1[i]=normalize*currents1[i];
currents2[i]=normalize*currents2[i];
}
}

return 0;
}
```

Commutation :

```
float gain(const float anglerad, const float offsetrad){
//this function deals with commutation

float angle=anglerad/PI*180.0; //convert to degrees
float offset=offsetrad/PI*180.0;
float offangle=angle+offset;// + offset?
float modulo45=offangle - int(offangle/45)*45;
float modulo90=offangle - int(offangle/90)*90;
float phi=abs(modulo45);
float gain{};
if (phi<=7) gain=1;
if (7<phi && phi<=15.5) gain=1+(7-phi)/8.5;
if (15.5<phi && phi<=29.5) 0;
if (29.5<phi && phi<=38) gain=0+(29.5−phi)/8.5;
if (38<phi && phi<45) gain=−1;
if (abs(modulo90)>=45)gain*=−1;
return gain;
}
```

Set PWM registers to get appropriate currents :

```
void CurrentToPWM(float *currents1, float *currents2){
//pin numbers pwm modules
constexpr int stby1=53;
constexpr int stby2=47;
constexpr int stby3=31;
constexpr int pwm1=45;
constexpr int pwm2=41;
constexpr int pwm3=44;
constexpr int pwm4=39;
constexpr int pwm5=37;
constexpr int pwm6=35;
constexpr int ain1=51;
constexpr int bin1=50;
constexpr int ain2=49;
constexpr int bin2=48;
constexpr int ain3=43;
constexpr int bin3=42;
constexpr int ain4=40;//38
constexpr int bin4=38;//40
constexpr int ain5=34;//36
constexpr int bin5=36;//34
constexpr int ain6=32;//33
constexpr int bin6=33;//32

constexpr float R1[3]{3.2, 3.2, 3.2}; //resistances set1
constexpr float R2[3]{3.2, 3.2, 3.2}; //resistances set2
constexpr float Vsupply=9;            //9V supply
constexpr int pin1[3]{pwm1,pwm2,pwm3};
constexpr int pin2[3]{pwm4,pwm5,pwm6};
float D1[3]{0,0,0};
float D2[3]{0,0,0};

//calculate and set PMW duty cycles in registers for all coils
for (int i=0;i<3;++i){
D1[i]=*(currents1+i)*R1[i]/Vsupply; //dutycycles
setDuty(pin1[i], D1[i]);//set registers

D2[i]=*(currents2+i)*R2[i]/Vsupply; //dutycycles
setDuty(pin2[i], D2[i]);
}
//set the transistors of the H-briges
setDirection(stby1, ain1, bin1, D1[0]);
setDirection(stby1, ain2, bin2, D1[1]);
setDirection(stby2, ain3, bin3, D1[2]);
setDirection(stby2, ain4, bin4, D2[0]);
setDirection(stby3, ain5, bin5, D2[1]);
setDirection(stby3, ain6, bin6, D2[2]);

}
```

Setup PWM duty cycle register for an output pin :

```
void setDuty(int pin, float dutyCycle){
//sets the dutycycle for a certain pwm pin

int value=REG_PWM_CPRD0-abs(dutyCycle)*REG_PWM_CPRD0; //calculate register values

//if wrong duty cycle input print error message
if (value<0 || REG_PWM_CPRD0<value){
Serial.println("Voltage out of range");
Serial.println();
return;
}

//set the PWM register for the appropiate pin
switch(pin){
```

```
case 35: REG_PWM_CDTY0 = value;
break;
case 37: REG_PWM_CDTY1 = value;
break;
case 39: REG_PWM_CDTY2 = value;
break;
case 41: REG_PWM_CDTY3 = value;
break;
case 44: REG_PWM_CDTY5 = value;
break;
case 45: REG_PWM_CDTY6 = value;
break;
default: Serial.print("Wrong pin selected in setDuty");
break;
}
return;

}
```

Set direction of the H-bridge :

```
void setDirection(int standby, int in1, int in2, float dutycycle){
//inputs are pin numbers that control the pwm module
// set the direction of the h-bridge
digitalWrite(standby, HIGH);
if(dutycycle>=0){ //forward
digitalWrite(in1, HIGH);
digitalWrite(in2, LOW);
}
else{
digitalWrite(in1, LOW);
digitalWrite(in2, HIGH);
}
}
```

I.2. Self-written Libraries

I.2.1. PID

PID.h file :

```
#ifndef SimplePID_h
#define SimplePID_h

#include "Arduino.h"

class SimplePID
{
        public:
        SimplePID(float Kp, float Kd, float Ki, float Tf, float limit);
        float calculate(float setpoint, float currentPos);
        void set(int Kp, int Kd, int Ki, int Tf);

        private:
        float _prev_time;
        float _dt;
        float _Kp;
        float _Kd;
        float _Ki;
        float _Tf;
        float _prev_error;
        float Iout;
        float _limit;
};

#endif
        content...
```

PID.ccp file :

```
#include "Arduino.h"
#include "SimplePID.h"
```

```
SimplePID::SimplePID(float Kp, float Kd, float Ki, float Tf, float limit):
_Kp{Kp},_Kd{Kd},_Ki{Ki},_Tf{Tf},_limit{limit}
{
        _prev_time=micros();
        _prev_error=0;
        Iout=0;

}

float SimplePID::calculate(float setpoint, float currentPos){
        _dt=(micros()-_prev_time)/1000000.0;
        //Serial.println(_dt,10);
        // calculate error
        float _error=setpoint-currentPos;
        // proportional part
        float Pout=_Kp*_error;
        // integral part
        Iout+=_Ki*_error*_dt;
        if(Iout> _limit) Iout=_limit;
        else if(Iout< -_limit) Iout=-_limit;

        //derivative part
        float _derivative=(_error-_prev_error)/_dt;
        float Dout=_Kd*_derivative/(_Tf*_derivative+1);
        //Output
        float output=Pout+Iout+Dout;
        if(output > _limit) output = _limit;
        else if(output < -_limit) output = -_limit;

        //save pre_error
        _prev_error=_error;
        _prev_time=micros();


        return output;

}
```

I.2.2. FIR

FIR.h file :

```
#ifndef SimpleFIR_h
#define SimpleFIR_h

#include "Arduino.h"

class SimpleFIR
{
        public:
        SimpleFIR(int buffersize);
        SimpleFIR(int buffersize, float* filtercoefficients);
        float bestEstimate();
        void push(float datapoint);
        void printBuffer();
        void setCoefficients(float* filtercoefficients);
        float filter(float datapoint);

        private:

        float *_bufferpointer;
        float *_filtercoefficients;
        float *_head;
        int _buffersize;
};

#endif
```

FIR.ccp file :

```
#include "Arduino.h"
#include "SimpleFIR.h"

SimpleFIR::SimpleFIR(int buffersize){
        _buffersize=buffersize;
        _bufferpointer=new float[buffersize];
        _head=_bufferpointer;
        float *p=_bufferpointer;
        for (int i=0;i<buffersize;i++){
                *p=0;
                ++p;

        }
}

SimpleFIR::SimpleFIR(int buffersize, float *filtercoefficients){
        _buffersize=buffersize;
        _bufferpointer=new float[buffersize];
        float *p=_bufferpointer;
        _head=_bufferpointer;

        _filtercoefficients=new float[buffersize];
        for (int i=0;i<buffersize;i++){
                *p=0;
                p++;
                *(_filtercoefficients+i)=*(filtercoefficients+i);
        }
}

void SimpleFIR::push(float datapoint){
        *_head=datapoint;
        if (++_head<_bufferpointer+_buffersize) _head;
        else _head=_bufferpointer;
}

void SimpleFIR::printBuffer(){
        for (int i=0;i<_buffersize;i++){
                if (_head+i<_bufferpointer+_buffersize) Serial.println(*(_head+i));
                else Serial.println(*(_head+i-_buffersize));
        }

        Serial.println();
}
```

```
float SimpleFIR::bestEstimate(){
        float bestEstimate{0};
        for (int i=0;i<_buffersize;i++){
                if (_head+i<_bufferpointer+_buffersize) bestEstimate+=*(_filtercoefficients+i)* *(_head+i);
                //if head pointer goes out of buffer bounds, go back to start
                else bestEstimate+=*(_filtercoefficients+i)* *(_head+i-_buffersize);

        }
        return bestEstimate;
}

float SimpleFIR::filter(float datapoint){
        push(datapoint);
        return bestEstimate();

}
```

Bibliography

[1] Linear encoder.

[2] Mitutoyo image correlation system, 2009. URL https://web.archive.org/web/20111013164722/http:/www.mitutoyo.com/pdf/1976_MICSYS.pdf.

[3] Moving average filter (ma filter), 2010. URL https://www.gaussianwaves.com/2010/11/moving-average-filter-ma-filter-2/.

[4] Spectral audio signal processing, 2011. URL https://ccrma.stanford.edu/~jos/sasp/Processing_Gain.html.

[5] Encoding with moiré, 2017. URL https://www.iap.uni-jena.de/iapmedia/de/Lecture/Optical+Metrology+and+Sensing1519858800/OMS17_Metrology+and+Sensing+Lecture+4+Fringe+projection-p-20002762.pdf.

[6] z-plane analysis of discrete-time control systems, 2017. URL https://slideplayer.com/slide/10490520/.

[7] Laser interferometer, 2019. URL https://en.wikipedia.org/wiki/Interferometry.

[8] Automatic hematology imaging analyzer - effective solution for automated differentiation of peripheral blood cells in large-sized laboratories, 2019. URL http://visionhemaultimate.com/.

[9] Incremental optical linear and rotary encoders now offer 1 nm resolution and ultra-low positional noise, 2019. URL https://www.renishaw.com/en/incremental-optical-linear-and-rotary-encoders-now-offer-1-nm-resolution-and-ultra-low-positional

[10] Resolute™ encoder series, 2019. URL https://www.renishaw.com/en/resolute-encoder-series--37823.

[11] Capacitance-based measurement principles, 2019. URL https://www.mtiinstruments.com/technology-principles/capacitance-based-measurement/.

[12] How to perform real-time processing on the raspberry pi, 2019. URL https://www.socallinuxexpo.org/sites/default/files/presentations/Steven_Doran_SCALE_13x.pdf.

[13] 68–95–99.7 rule, 2019. URL https://en.wikipedia.org/wiki/68%E2%80%9395%E2%80%9399.7_rule.

[14] White noise, 2019. URL https://en.wikipedia.org/wiki/White_noise.

[15] Arduino basics, 2019. URL https://sites.google.com/site/hogeropmetarduino/arduino-basisvaardigheden/les-7-motor/h-bridge.

[16] The scientist and engineer's guide to digital signal processing, 2019. URL http://www.dspguide.com/ch13/4.htm.

[17] Temperature coefficient of copper, 2019. URL https://www.cirris.com/learning-center/general-testing/special-topics/177-temperature-coefficient-of-copper.

[18] Alan s Willsky Alan v. Oppenheim. *Signals and systems.* Prentice Hall Signal Processing Series. Prentice Hall, United States of America, 2nd ed. edition, 1997. ISBN ISBN 0-13-814757-4.

[19] M. Café. *Nanometer precision Six Degrees of Freedom Planar Motion Stage with Ferrofluid Bearings.* Master thesis, 2014.

[20] Rong-Seng Chang, Jin-Yi Sheu, Ching-Huang Lin, and He-Chiang Liu. Analysis of ccd moiré pattern for micro-range measurements using the wavelet transform. *Optics & Laser Technology*, 35(1):43–47, 2003. ISSN 0030-3992. doi: https://doi.org/10.1016/S0030-3992(02)00122-6. URL http://www.sciencedirect.com/science/article/pii/S0030399202001226.

[21] Kai Engelhardt and Peter Seitz. Absolute, high-resolution optical position encoder. *Applied Optics*, 35(1):201–208, 1996. doi: 10.1364/AO.35.000201. URL http://ao.osa.org/abstract.cfm?URI=ao-35-1-201.

[22] H. Habib. *Design of a three Degrees of Freedom planar precision stage using a single Position Sensitive Detector*. Master's thesis, 2005.

[23] S. Hesse, C. Schäffel, H. U. Mohr, M. Katzschmann, and H. J. Büchner. Design and performance evaluation of an interferometric controlled planar nanopositioning system. *Measurement Science and Technology*, 23(7):074011, 2012. ISSN 0957-0233 1361-6501. doi: 10.1088/0957-0233/23/7/074011. URL http://dx.doi.org/10.1088/0957-0233/23/7/074011.

[24] Mike Holmes, Robert Hocken, and David Trumper. The long-range scanning stage: a novel platform for scanned-probe microscopy. *Precision Engineering*, 24(3):191–209, 2000. ISSN 0141-6359. doi: https://doi.org/10.1016/S0141-6359(99)00044-6. URL http://www.sciencedirect.com/science/article/pii/S0141635999000446.

[25] Li Junfei, Zhang Youqi, Wang Jianglong, Xiang Yang, Wu Zhipei, Ma Qinwei, and Ma Shaopeng. Formation mechanism and a universal period formula for the ccd moiré. *Optics Express*, 22(17):20914–20923, 2014. doi: 10.1364/OE.22.020914. URL http://www.opticsexpress.org/abstract.cfm?URI=oe-22-17-20914.

[26] W. Jywe, Y. Jeng, Y. Teng, H. Wang, and C. Wu. Development of the nano-measuring machine stage. In *IECON 2007 - 33rd Annual Conference of the IEEE Industrial Electronics Society*, pages 2970–2973. ISBN 1553-572X. doi: 10.1109/IECON.2007.4460035.

[27] Jong-Ahn Kim, Jae Wan Kim, Chu-Shik Kang, and Tae Bong Eom. Metrological atomic force microscope using a large range scanning dual stage. *International Journal of Precision Engineering and Manufacturing*, 10(5):11–17, 2009. ISSN 2005-4602. doi: 10.1007/s12541-009-0087-z. URL https://doi.org/10.1007/s12541-009-0087-z.

[28] P. Klapetek, M. Valtr, and M. Matula. A long-range scanning probe microscope for automotive reflector optical quality inspection. *Measurement Science and Technology*, 22(9):094011, 2011. ISSN 0957-0233 1361-6501. doi: 10.1088/0957-0233/22/9/094011. URL http://dx.doi.org/10.1088/0957-0233/22/9/094011.

[29] S.G.E Lampaert. *Planar Ferrofluid Bearings Modelling and Design Principles*. Msc, 2015.

[30] Doug Leviton and Mario S. Garza. *Recent advances and applications of NASA's new ultrahigh-sensitivity absolute optical pattern recognition encoders*. 2000. doi: 10.1117/12.405797.

[31] Douglas B. Leviton and Brad Frey. *Ultrahigh-resolution absolute position sensors for cryostatic applications*, volume 4850 of *Astronomical Telescopes and Instrumentation*. SPIE, 2003.

[32] Chien-Hung Liu, Wen-Yuh Jywe, Yeau-Ren Jeng, Tung-Hui Hsu, and Yi-tsung Li. Design and control of a long-traveling nano-positioning stage. *Precision Engineering*, 34(3):497–506, 2010. ISSN 0141-6359. doi: https://doi.org/10.1016/j.precisioneng.2010.01.003. URL http://www.sciencedirect.com/science/article/pii/S0141635910000048.

[33] Richard G. Lyons. *Understanding Digital Signal Processing*. Prentice Hall, United States of America, 3rd edition edition, 2010. ISBN ISBN-13: 978-0-13-702741-5.

[34] Eberhard Manske, Gerd Jäger, Tino Hausotte, and Roland Füßl. Recent developments and challenges of nanopositioning and nanomeasuring technology. *Measurement Science and Technology*, 23(7):074001, 2012. ISSN 0957-0233 1361-6501. doi: 10.1088/0957-0233/23/7/074001. URL http://dx.doi.org/10.1088/0957-0233/23/7/074001.

[35] G. Mok. *The design of a planar precision stage using cost effective optical mouse sensors.* Master thesis, 2015.

[36] S. Odenbach. *Recent progress in magnetic fluid research*, volume 16. 2004. doi: 10.1088/0953-8984/16/32/R02.

[37] A. Rankers R. Munnig Schmidt, G. Schitter and J. van Eijk. *The Design of High Performance Mechatronics.* Delft University Press, Netherlands, 2nd edition edition, 2014. ISBN 978-1-61499-367-4. doi: 10.3233/978-1-61499-368-1-i.

[38] Eugene I. Rivin. Vibration isolation of precision equipment. *Precision Engineering*, 17(1):41–56, 1995. ISSN 0141-6359. doi: https://doi.org/10.1016/0141-6359(94)00006-L. URL `http://www.sciencedirect.com/science/article/pii/014163599400006L`.

[39] Guanqiao Shan, Yingzi Li, Liwen Zhang, Zhenyu Wang, Yingxu Zhang, and Jianqiang Qian. Contributed review: Application of voice coil motors in high-precision positioning stages with large travel ranges. *Review of Scientific Instruments*, 86(10):101501, 2015. ISSN 0034-6748. doi: 10.1063/1.4932580. URL `https://doi.org/10.1063/1.4932580`.

[40] M. Torralba, M. Valenzuela, J. A. Yagüe-Fabra, J. A. Albajez, and J. J. Aguilar. Large range nanopositioning stage design: A three-layer and two-stage platform. *Measurement*, 89:55–71, 2016. ISSN 0263-2241. doi: https://doi.org/10.1016/j.measurement.2016.03.075. URL `http://www.sciencedirect.com/science/article/pii/S0263224116300537`.

[41] L. van Moorsel. *A planar precision stage using a single image sensor.* Master of science thesis, 2017.

[42] J.K. van Seggelen. *NanoCMM: A 3D Coordinate Measuring Machine with Low Moving Mass for Measuring Small Products in Array with Nanometer Uncertainty.* Technische Universiteit, 2007. ISBN 9789038626291. URL `https://books.google.nl/books?id=YZPcswEACAAJ`.

[43] Simon van Veen. *Planar ferrofluid bearings for precision stages.* Msc, 2013.

[44] PhD Wei Gao. *Precision Nanometrology - Sensors and Measuring Systems for Nanomanufacturing.* Advanced Manufacturing. Springer, London, 2010. ISBN 978-1-84996-253-7. doi: 10.1007/978-1-84996-254-4.

[45] Hai Yu, Qiuhua Wan, Xinran Lu, Yingcai Du, and Shouwang Yang. Small-size, high-resolution angular displacement measurement technology based on an imaging detector. *Applied Optics*, 56(3):755–760, 2017. doi: 10.1364/AO.56.000755. URL `http://ao.osa.org/abstract.cfm?URI=ao-56-3-755`.