Testing of piezoelectric pressure sensors

For Momo Medical

B.K.D. Koele
T.H.T. Leferink
T.L.P. Nguyen

TU Delft

December 17, 2018

Testing of piezoelectric pressure sensors

For Momo Medical

by

B.K.D. Koele
T.H.T. Leferink
T.L.P. Nguyen

to obtain the degree of Bachelor of Science

at the Delft University of Technology,

to be defended on Friday December 21, 2018 at 10:00 AM.

Student number:	4212762, 4098560, 4377273	
Project duration:	October 8, 2018 – December 21, 2018	
Thesis committee:	Ing. J. Bastemeijer,	TU Delft, project supervisor
	Dr. ing. I. E. Lager,	TU Delft, BAP coordinator
	Dr. J. Hoekstra,	TU Delft, jury member
	M. Gravemaker, B. Sc.,	Momo Medical, co-founder

This thesis is confidential and cannot be made public until December 21, 2023.

Abstract

In this thesis, a test device is designed for the company Momo Medical. Momo Medical is developing a system that can be used to prevent pressure ulcers and adding more functionality is being investigated. For this system they need a device to test the functionality of the six piezoelectric sensors that are used in their product. The sensors' response to a known pulse needs to be tested in order to give a pass/fail indication of the sensor quality.

First, different ways of testing the sensors are investigated. Based on the results of the investigation, a final test setup is chosen and characterized.

The test system developed in this thesis is based on a pneumatic setup, using a solenoid valve to control well-defined air pulses directed towards the sensors. Due to the addition of a reference load cell with custom designed read-out electronics, the device is able to test all six sensors one at the time and provides detailed feedback about the individual sensor quality. The test is performed using GUI-based software written in MATLAB, connected to a microcontroller. The software offers a broad variety of settings and can be configured according to Momo Medical's wishes. After configuration, the test can be performed at the click of a button.

The standard deviation of the device precision over three hours is $\sigma = 4.16$, which equals a variation of $c_v = 0.82\%$ of the mean $\mu = 504.8$. This easily satisfies the requirements set by Momo Medical.

Preface

This thesis is written in December 2018 during a ten-week period as part of the final project of the Electrical Engineering Bachelor's degree at the Delft University of Technology. When our graduation project started, we had some difficulty coming up with a good development idea. After some thought we figured we would take a look at smart chairs, to help people sit better.

Thanks to dr. ing. Ioan Lager, we quickly came in contact with Momo Medical. His efforts have greatly helped us in getting a quick start with our bachelor graduation project. We would like to thank him for putting in the effort to organize the bachelor graduation project for only three people, his great insights and many tough but helpfull questions, allowing us to properly motivate all choices made in this thesis.

We had not heard of Momo Medical before and were very keen to set a meeting to discuss possible graduation projects. As they were focusing on their pressure ulcer prevention tool, they could not start the development of a smart chair. They did however offer us a great opportunity to help them with improving their prevention tool and as our interest lied in medical applications, this became the subject of our thesis.

Less than a week later, we could start our work for Momo Medical. This was in large part thanks to Menno Gravemaker, co-founder of Momo Medical. His great support and helpful ideas have enabled us to reach further and develop ourselves on a personal as well as a professional level. When working at Momo, the communal lunches were great fun; ideas could be pitched to any colleague and great discussions arose. We would like to thank all of Momo Medical's staff for their input and ideas; in particular we would like to thank Roel van der Plas for his input on our final design.

From our faculty, we had great supervision from ing. Jeroen Bastemeijer. His quick responses and open door policy with no rushed meetings greatly helped us to ask detailed questions and consider our problems as a team. His practical ideas and approaches allowed us to quickly gather data from 'proof of concept' setups. His knowledge of mechanical systems was of great help and he even supplied us with parts when he thought it would advance our project.

B.K.D. Koele
T.H.T. Leferink
T.L.P. Nguyen
Delft, December 17, 2018

Contents

1

Introduction

In this chapter, an introduction will be given about Momo Medical, the product they are developing and what problem they are facing.

1.1. Pressure ulcers and Momo Medical

Pressure ulcers are localized damages to the skin and/or underlying tissue. They are caused by pressure to the skin over an extended period of time. This affects the upper skin and underlying tissue. These wounds are painful and take a long time to heal [1]. Pressure ulcers cause patient suffering, high expenses and increased workload for health care staff [2].

Momo Medical (hereafter also called Momo) strives to prevent these wounds from occurring. They developed a smart technology that provides continuous insight in the posture and movements of the patient. Their solution consists of a sensor plate and matching control unit (seen in figure 1.1), providing feedback to nurses and other staff. If a patient has not repositioned him/herself after a set amount of time, a nurse can be alerted.

The sensor plate is a thin plate (approximately 65x12x1 cm) placed underneath the mattress. It is covered by a sleeve to protect it against moisture and dust (not shown in the figure). The sensor plate provides a non-intrusive way to detect the position of the patient using a variety of sensors.

The control unit is a small box attached to the wall or the bed frame and reads the signals from the sensor plate. The signals are analyzed to determine the position of the patient. A circle of RGB LEDs indicates how long the patient has been in his/her current position (see figure 1.1). As soon as a patient lies down in the bed, a timer starts in the control unit. When a patient repositions him/herself, or a nurse repositions the patient, the timer is reset. If a repositioning has not occurred after a set time (e.g. 3 hours), a nurse is notified.

In future versions they would also like to be able to detect heart rate and breathing patterns with the same device [3].

1.2. The sensor plate in detail

In this thesis version 5 of Momo's sensor plate is considered. The sensor plate contains 6 dynamic force sensors and 8 static force sensors, as well as an accelerometer to detect the bed angle. Piezoelectric (PE) sensors are used as the dynamic force sensors, and force-sensing resistors (FSRs) are used as the static force sensors. All sensors are connected to conditioning circuits, after which the signal passes through to analog-to-digital converters (ADCs). The ADCs are then connected to the control unit using I^2C.

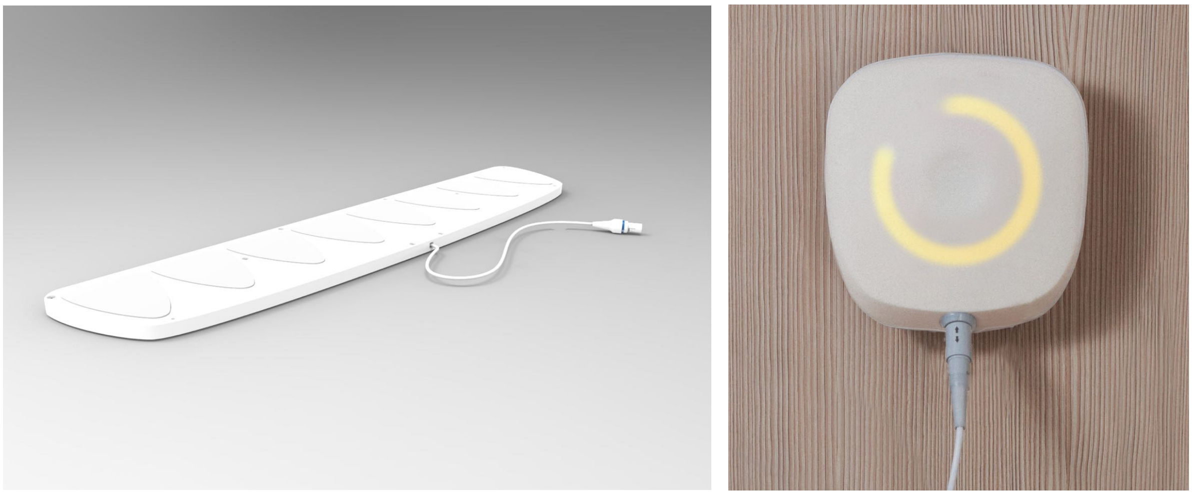


Figure 1.1: Sensor plate (left) and control unit (right). Pictures provided by Momo Medical

1.2.1. Domes and pucks

Momo refers to the ‘triangle’ shaped parts (seen in figure 1.2) as domes. The PE sensors are taped to the bottom of the six middle domes. On the narrow end there is a so called ‘puck’ (see figure 1.3), where the dome rests on the FSRs, on the opposite side the puck rests on the frame. The PE sensor thus measures how much the dome deforms.

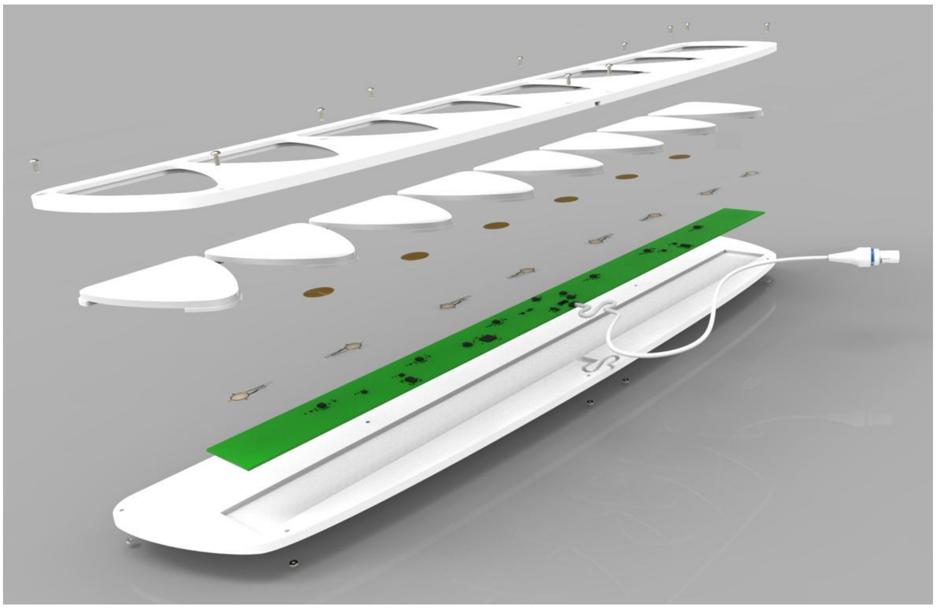


Figure 1.2: Sensor plate "exploded view". Picture provided by Momo Medical

1.2.2. PE sensor used by Momo Medical

Momo uses piezoelectric sensors to detect small vibrations through the mattress.

A piezoelectric sensor is based on the piezoelectric effect, found by Pierre and Jacques Curie in 1890. A piezoelectric material consists of a crystal lattice with electric dipole moments. These dipoles are generally oriented randomly throughout the material, so no net polarization is exhibited [4]. When a strong electric field is applied, the dipoles orient themselves according to the applied field.

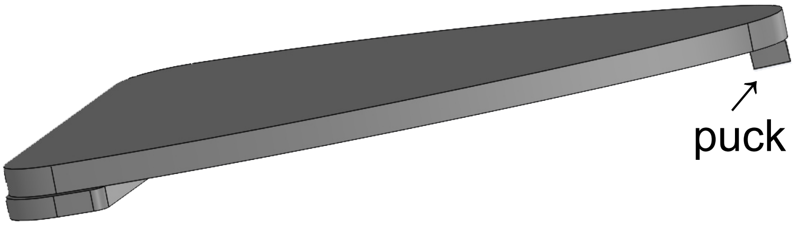


Figure 1.3: Dome with puck. Picture provided by Momo Medical and edited by the authors

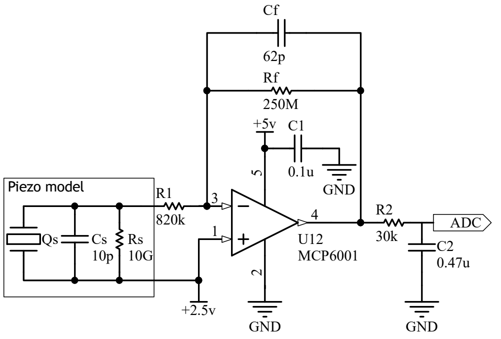


Figure 1.4: Conditioning circuit used in the current sensor plate for reading out the piezoelectric sensors

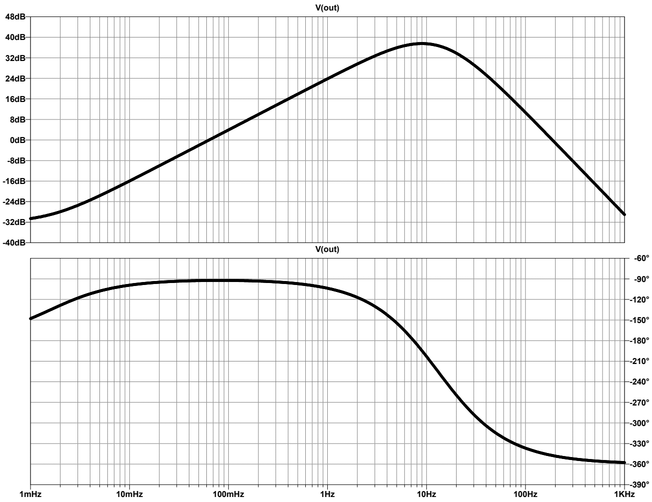


Figure 1.5: AC analysis in LTspice of the conditioning circuit shown in figure 1.4

When an external mechanical stress is induced, the dipoles change orientation. To the outside, this appears as a variation of surface charge density upon the different faces of the lattice [5].

The PE sensors used by Momo are piezoelectric diaphragms manufactured by Murata, type 7BB-20-6L0. Even though these diaphragms are designed to be used as buzzers, they can be used in reverse as pressure sensors. The diaphragms belong to the family of lead zirconate titanate (PZT) ceramics. PZT ceramics “exhibit very high dielectric and piezoelectric properties and find wide applications as sensor and actuator devices" [6]. The specific diaphragms used by Momo are 20 mm in diameter, have a resonance frequency $f_{res} = 6.3 \pm 0.6\,\text{kHz}$ and a capacitance of $10\,\text{nF} \pm 30\,\%$ [7].

1.2.3. Schematic of the PE sensor conditioning circuit

In order to properly read the data coming from the sensor, Momo Medical uses a conditioning circuit as seen in figure 1.4. As a model for the piezoelectric sensor a charge source is used in parallel with a resistor $R = 10\,\text{G}\Omega$ (see [8]) and a capacitor $C = 10\,\text{nF}$. A simulation in LTspice is performed and the frequency response can be seen in figure 1.5. The entire circuit can be viewed as having a first order high-pass filter with a cut-off frequency $f_{hpf} = 4\,\text{Hz}$ and a second order low-pass filter with a cut-off frequency of $f_{lpf} = 18.4\,\text{Hz}$.

1.2.4. Analog-to-digital conversion

After the conditioning circuit, the signals are fed to an ADC. The ADCs used in Momo’s current system are the ADS1015 made by Texas Instruments. These ADCs are 12-bit with a programmable gain amplifier (PGA) of 1, 2, 4, 8 or 16x. This PGA can amplify the signal before it enters the A/D converter itself. With a gain of 1, the full-scale range of the ADS1015 is $\pm 6.144\,\text{V}$ since it has an internal voltage reference. This means the least significant bit size equals

$$\frac{2 \cdot \frac{6.144}{16}}{2^{12}} = 0.125\,\mathrm{mV} \tag{1.1}$$

The smallest output voltage of the conditioning circuit as can be measured by the ADC is therefore 0.125 mV[9]. On the other hand we have the maximum value of the supply voltage (5V).

1.3. Problem overview

At this moment, the data from the piezoelectric sensors is unusable for a proper algorithm. The difference in sensitivity is more than 100% from sensor to sensor, even in a single sensor plate. As Momo's product will be put into production in the next few months, there is an urgency to address this problem. The goal of this thesis is to design a device which tests the sensors before they are shipped out, to make sure they are able to detect a heart rate and breathing pattern whilst placed under a mattress. After this, they would like to be able extract some values so the data can then be trimmed when data is read from the sensor plate.

1.3.1. Problem definition

Momo Medical wishes to have their piezoelectric sensors in their sensor plate tested before they are shipped out. For a sensor plate to be usable in the field, it is necessary that every sensor can be considered as equally sensitive. This means that every sensor must respond in a similar manner to a certain applied pressure. Sensor plates that pass can be shipped to the end users, plates that fail go back to the production area to be fixed. This test is done at the production facility and should be done in the same time it takes the production company to build one sensor plate (30 min). The test ensures that faulty sensors are replaced before the sensor plate is shipped out to the customers. Momo Medical would like to make use of the piezoelectric sensors to accurately measure heart rate and breathing pattern, as well as improving patient detection and to offer extra functionality to the customer. The sensor plate is the device-under-test (DUT).

1.3.2. System requirements

Momo Medical has provided certain requirements for the proposed testing device. These requirements are classified using the MoSCoW method [10] into Must haves, Should haves, Could haves and Won't haves tables (see table 1.1, 1.2, 1.3 and 1.4, respectively). The requirements placed in each section are numbered arbitrary and do not reflect their priority.

Table 1.1: Must-have requirements. Requirements from both the resources provided by Momo and the system design requirements. These requirements have a high priority and must be fulfilled in order to achieve the goal; designing a testing machine for the sensor plate.

Must Haves

Requirement	Description
M1	The testing must be done on the assembled sensor plate without protecting sleeve
M2	The maximum weight on the single sensor is 5 kg
M3	The minimum sampling frequency used for reading out the sensors is 50 Hz
M4	The dimensions of the testing system must be smaller than 2x2x2m (l x w x h)
M5	A production worker with moderate technical knowledge must know how to use the testing device at the production location, after a maximum of 8 hours of instructions
M6	All calculations must be done in software (external PC or embedded firmware) with a pass/fail indication per sensor
M7	All measurements must be stored in an external device
M8	The testing system must use 230 V AC as input (<16 A, 50 Hz)
M9	User documentation must be provided for operating and testing the system
M10	The sensor plate values must be read-out via the existing I^2C interface
M11	The total price of the materials used for the testing system must be lower than €10,000 (ex. VAT)
M12	The testing of a single sensor plate must be done within 30 minutes

Table 1.2: Should-have requirements. Requirements that would make the system better, but are not needed to achieve the goal

Should Haves

Requirement	Description
S1	When starting the software, the user should only have to press one start button in order to start testing a sensor plate
S2	The reference sensor should give equal responses within a ±5% margin in amplitude, 3 hours after testing
S3	The testing device should test all 6 sensors without manually repositioning the system
S4	Compensation coefficients should be exported from the software

Table 1.3: Could-have requirements. Requirements would be desirable for the system, but will only be touched upon if there is enough available time.

Could Haves

Requirement	Description
C1	The force-sensing resistors could be calibrated absolute with a 5% error margin
C2	The static and dynamic testing systems could be integrated into a single system
C3	The system could be made portable and simple so it is usable for Momo clients (e.g. hospitals, nursing homes) to have a test unit in-house
C4	The testing device could have a self-test option
C5	The testing device could meet CE and RoHS requirements

Table 1.4: Won't-have requirements. Requirements that will not be part of the current schedule, but may be interesting in the future to work on by another team

Won't Haves

Requirement	Description
W1	Calibration data is stored in the Cloud

2

Design Concepts

In this chapter different solutions to the problem described previously are put to the test and a final setup is chosen.

2.1. State-of-the-art analysis

The state-of-the-art analysis focuses on comparing similar solutions for a certain problem. Applications that use dynamic pressure sensors are looked into to get a better understanding of the currently used methods for calibrating dynamic forces.

2.1.1. Calibration systems

Calibration is a fundamental process for instruments that require high accuracy. During the calibration process, measurements are done on the instrument which are compared to values from a standard reference device. In case the measurements differ from each other, the instrument can be adjusted to ensure the results comply with the standard reference.

In modern processes, the standard reference is based on the SI units and some of their derived units. As the characteristics of devices change over time due to material property or the different environments it is used in, it is necessary to calibrate devices frequently to ensure they are still accurate.

Different kinds of sensors are used to perform measurements. Force can be measured using a wide variety of sensors [11]. For each of these sensors, calibration is necessary if accurate measurements are needed. Calibration systems are used to perform these calibrations and must comply to the standard reference for that quantity. The standard reference in the case of calibrating a force sensor is a known applied force. For each known force, the sensor read out should result in the same value as the calibrated system. If this is not the case, the instrument is not accurate and should not be used for applications which require high accuracy.

2.1.2. Dynamically calibrating force sensors

Systems which calibrate force sensors are common. However, the amount of force applied to an instrument is different. Tekscan makes force-sensing resistors called FlexiForce sensors [12]. These sensors have a range between 0 N and 111 N (0 - 25 lb). One of the possible ways of calibrating these sensors is written by Somer et al. [13]. In this paper, static calibration is done with the use of static weights and dynamic calibration is done through the inertial force of the mass. In order to transfer static force to the sensor, a lever mechanism is used in combination with a digital weighing device. Dynamic force is transferred through an oscillating mass. Different masses are used for this calibration process. In order to determine the amount of pressure applied to the sensor, a relation between the pressure applied to the sensor and the inertial force is made. As the mass is known and the acceleration is measured through an accelerometer, the inertial force can be calculated.

Unfortunately, the calibration system used to measure these FlexiForce sensors cannot be used to calibrate Momo's sensor plate, since the sensors should be calibrated while they are in the sensor plate. This method to mount these sensors to the plate can affect the sensitivity. Finally, the surface contact of the mass to the domes on the sensor plate may affect the measurement. In order to solve the issue with the surface contact, a medium must be chosen which is able to transfer the force from the mass onto the dome.

Another commonly used method for calibrating force sensors is by means of shock tubes. A shock tube consists of a closed tube with two compartments separated by a diaphragm. One compartment is filled with a low pressure gas (driven gas) with the force transducer on the end of the tube. The other compartment is filled with a high pressure gas (driver gas). When the pressure of the driver gas is increased, the diaphragm will rupture at a predetermined pressure. The high pressure of the driver gas expands in the direction of the low pressure size and increases the temperature of the driven gas as a result of the shock wave. The shock can be measured to calibrate the force transducer [14].

The frequency range of interest for the piezoelectric sensors is below 20 Hz. Since the shock tube produces an impulse (the rise time of the produced shock is in the order of nanoseconds, see [15]), frequency components from DC to the MHz-range are present.

The main disadvantages for using a shock tube is that the piezoelectric sensors Momo uses, have to be tested in the complete sensor plate after production. This means that it is not possible to use a shock tube, since that requires a single sensor to be mounted on the end of the tube. Mounting a complete sensor plate in the tube won't be a feasible solution.

2.2. Morphological chart

In order to make a selection out of the possible options for a dynamic force measurement, a morphological chart is created. In table 2.1 it can be seen that several options are compared with each other. The best concepts will be put to the test in this thesis. This chart is based on assumptions and only the chosen options will be investigated in detail.

From this table the following options are chosen: steel beam, dome-on-dome and the pneumatic setup. The first two options can be easily tested, as many of the parts necessary are readily available. At the same time the last option is being investigated, and all parts needed are ordered.

Table 2.1: Morphological chart used to determine the best concepts

	Pneumatic		Mechanical (vibrating motor)		Dome-on-Dome		Electromagnet		Speaker		Dropping weight	
Price of non loanable components	Valve: €40, Tubing: €20, couplers €15, frame €20	+	Steel beam €20	+	Dome, piezoelement €5	++	Electromagnet, frame	+	Large speaker, cabinet, amplifier €2500	–	Weight, dropping mechanism, lifting mechanism €100	+
Feasible in time	Yes, doable	+	Yes, easy	++	Yes, easy	++	Yes, doable	+	Yes but costs play a role	+	No, large time investment in weight retrieval mechanism	-
Control	24V, ±4 watt	++	PWM (stepper) motor driver	++	Function generator: 0-20Vpp	0	MCU	+	Amplifier	+	MCU controlling drop, lift up, etc.	–
Size	Compressor + some tubing and valves in a frame needed	0	Steel beam of min. 80cm needed	0	Small, but shielded case important	++	Small	+	Large, >24" inch driver needed for low freqs.	–	Small	+
Frequency span	Max. freq. solenoid valves: 15Hz	+	Sub-Hertz difficult	+	Very low amplitudes with low frequencies	+	Impulse	-	Audio amplifier	++	Only impulse, so high frequencies	- - -
Coupling between vibration and sensor plate	No extra interface needed with air on dome	+++	Some kind of rubber pads needed + vibration not everywhere the same	–	Hard, rubber pads of some kind of gel needed	-	Same as other contact options. EM-field could play extra role	–	Air pressure not the same over the entire sensor plate	—	Good	+
Simultaneously measure all sensors	No, interference between measurements	0	With a single beam yes	+	No, interference between measurements	0	No, interference between measurements	-	Yes	+	No, interference between measurements	0
Lifespan	Solenoid valve only moving part	++	Motor could fail easy with weight	0	Short, piezo's excited with 20V decreases lifespan	-	Little moving parts	+	Large conus with low amplitude, large movement	0	Multiple servo's (i.e. moving parts) needed	-
Self-test functionality	Fixed force sensor under actuator	++	Gyroscoop/ accelerometer on the beam	+	Fixed force sensor under actuator but very small force	+	Fixed force sensor under weight	++	Microphone	+	Fixed force sensor under weight	++
Reproducible	Air nozzle position dependent	-	Yes	+	Very position dependent	-	Moving pin can get magnetized	-	Not that position dependent	+	Air pressure, position dependent	–
Speed single measurement	Fast, couple of seconds per measurement	+	Fast, couple of seconds per sensor plate	++	Fast, couple of seconds per measurement	+	Fast, couple of seconds per measurement	+	Fast, couple of seconds per sensor plate	++	Slow	0
Practicality	High, easy setup	+	High, but placement is crucial	+	Low, shielded environment needed	-	High, easy setup	+	Large, heavy	0	No bounces difficult, projectile retrieval difficult	–
Pro's		14		12		9		9		9		5
Con's		1		2		4		5		7		8
Net		13		10		5		4		2		-3

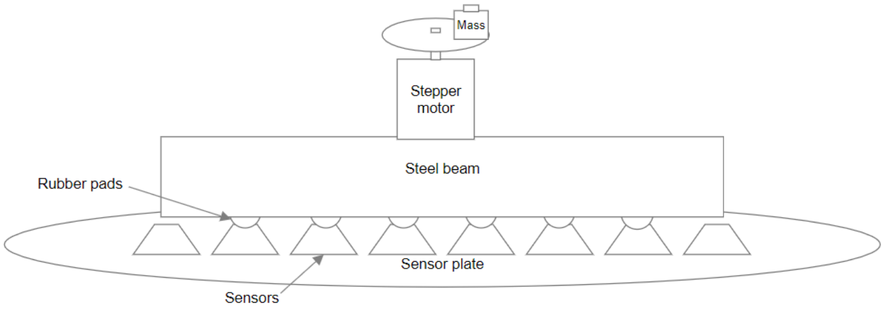


Figure 2.1: Schematic overview of the steel beam setup

2.3. Setup

The setup used for measuring the piezoelectric sensors consists of a microcontroller connected to the DUT through an I^2C interface. The microcontroller (LPC1768, made by NXP) uses Mbed as its software platform. The microcontroller reads out values from the ADC and can modify the ADC's PGA. Values received from the ADC are sent through a UART connection to a program made in MATLAB App Designer. The values received by MATLAB are processed and plotted in the GUI. By means of the program, measurement options can be selected which are sent to the microcontroller. These measurement options include which sensor is read out, the duration of the measurement and the gain factor of the PGA.

2.4. Steel beam with stepper motor setup

Since a stepper motor with driver is available at Momo Medical, this setup is considered first. This stepper motor is attached to a steel beam. The aim is to get a general idea of the problem at hand and to see if this setup proves to be a good candidate to be used in a final design.

2.4.1. Test Setup

A stepper motor with a small eccentric weight (mass $m = 50\,\mathrm{g}$, radius axis to weight $r = 1\,\mathrm{cm}$) is attached to a steel beam (60 mm × 60 mm × 1 m, 5.1kg). To ensure proper contact between the steel and the domes, soft plastic adhesive pads are carefully stuck on the steel to ensure that each pad is equally spaced. The steel beam is positioned on top of the DUT. An overview of the setup can be seen in figure 2.1. A microcontroller is used to generate a square wave with variable frequency, which controls a stepper motor driver (DRV8825) [16]. When the motor is spinning at a certain frequency (i.e. rotations per second) the unbalanced weight creates a vibration on the beam. The measurements are performed with test frequencies of 1, 2, 3, 4, 5, 6, 7, 8, 9 and 10 Hz.

2.4.2. Results

The unfiltered output for four different frequencies (1 Hz, 2 Hz, 3 Hz and 4 Hz) is plotted in figure 2.3. In figure 2.4 the frequency response for an input frequency of 5 Hz is shown to give an indication of the individual sensor responses. A clear difference between sensor responses can be seen. Appendix D.1

To see if standing waves in the steel beam play a role in the deflection of the beam, a simple calculation is made. If the standing waves in the beam turn out to have a significant impact, it would render the measurements useless, since the beam itself would move differently at every location along the beam. The speed of sound in steel is 4880 to 5050 $\mathrm{m\,s^{-1}}$ [17]. The maximum used test frequency is 10 Hz. This results in a wavelength of minimally $\lambda = \frac{v}{f} = \frac{4880}{10} = 488\mathrm{m}$. This is much larger than the 1 meter beam used in this experiment and therefor any deflection due to standing waves is negligible.

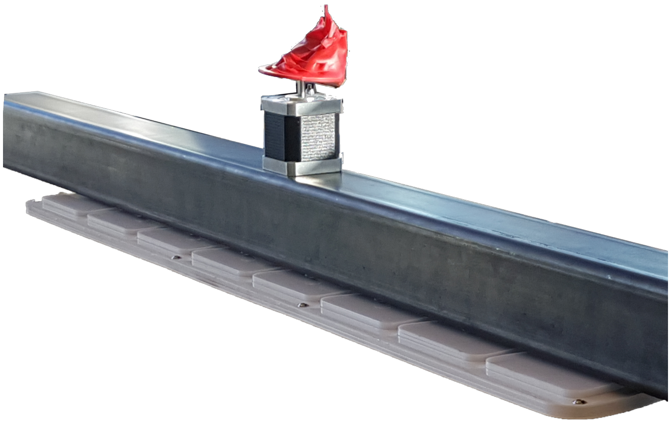


Figure 2.2: Actual steel beam setup

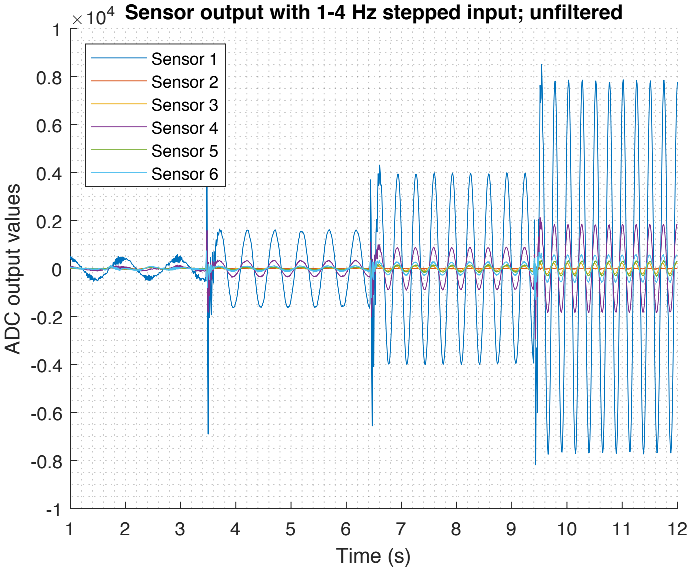


Figure 2.3: Sensor output with a 1 Hz (1-3.5s), 2 Hz (3.5-6.5 s), 3 Hz (6.5 - 9.5 s) and 4 Hz stepped input

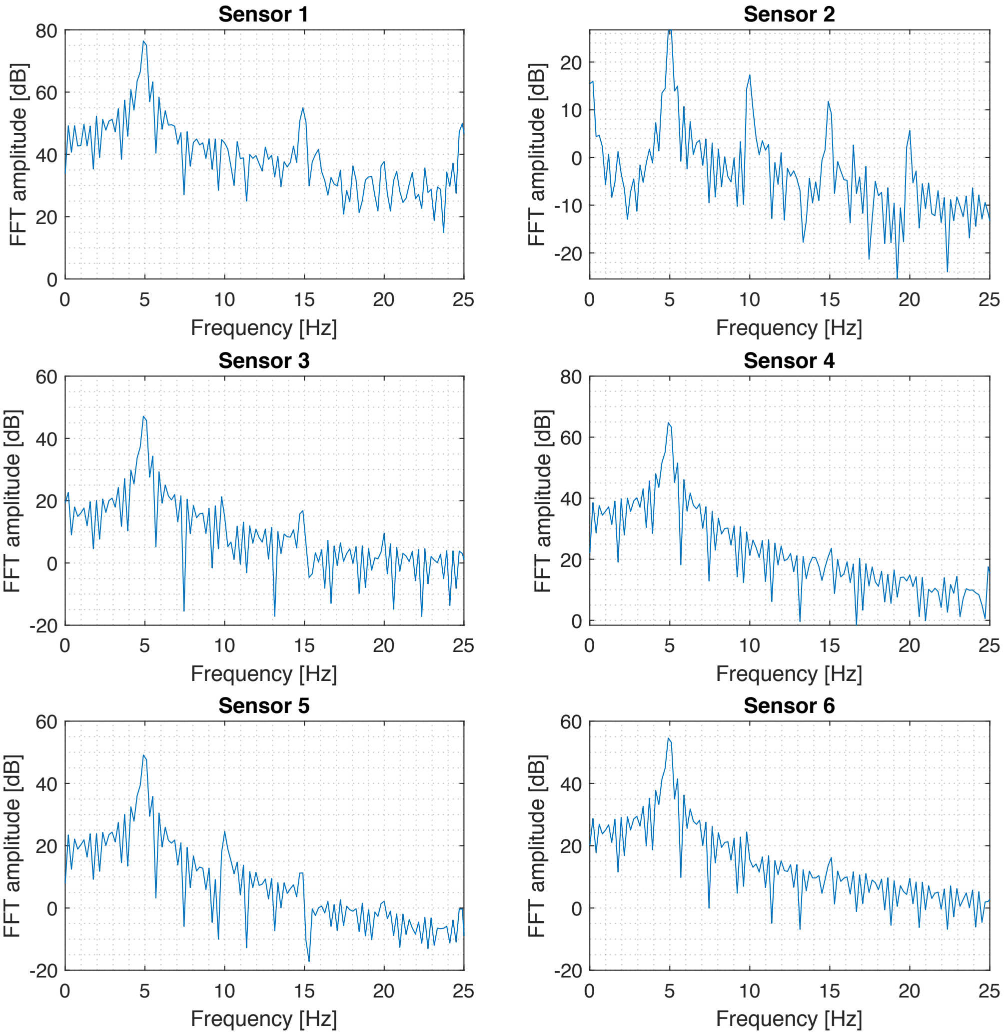


Figure 2.4: Frequency response of six PE sensors at a frequency of 5 Hz using the steel beam setup

2.4.3. Conclusion

The results shown confirm the expectation by Momo that the sensors' sensitivity varies greatly. When testing using the current setup, it is difficult to determine the amount of force the motor with its weight induces on the six sensors. Even when this force is known, it cannot be said with certainty that each sensor is exerted with 1/6th of the total force. Since all domes are not at the exact same height, no proper conclusions can be drawn from the sensor responses.

In a production environment this setup also has some flaws, as the motor used is not made for an eccentric load this could cause the motor to fail quickly.

2.5. Dome-on-dome

In order to get some information on the possibility of using a PE sensor as a sort of actuator, one dome is placed upside down on another dome. The bottom dome is excited with a function generator with a frequency of 1, 2, 4, 8 and 16Hz. As an additional advantage, this setup allows for an easy comparison of different binding materials of the PE sensor to the dome. This is an import aspect, since Momo raised questions about the currently used double sides tape.

The tests are performed with 6 domes, pairs of two, with the PE sensors fixated with the following binding materials:

- Super glue (cyanoacrylate)
- Epoxy
- Double sided tape

2.5.1. Test Setup

The test setup consists of two domes placed back-to-back, with one being driven by a function generator ($V_{pp} = 20\,\mathrm{V}$ and the other being read out by Momo's sensor plate PCB. The same conditioning circuit is used for all dome-on-dome tests. In order to minimize the influence of external electromagnetic fields, both domes are placed in a shielded container, with all ground connections made to mains earth. See figure 2.5 for a schematic overview of the used system.

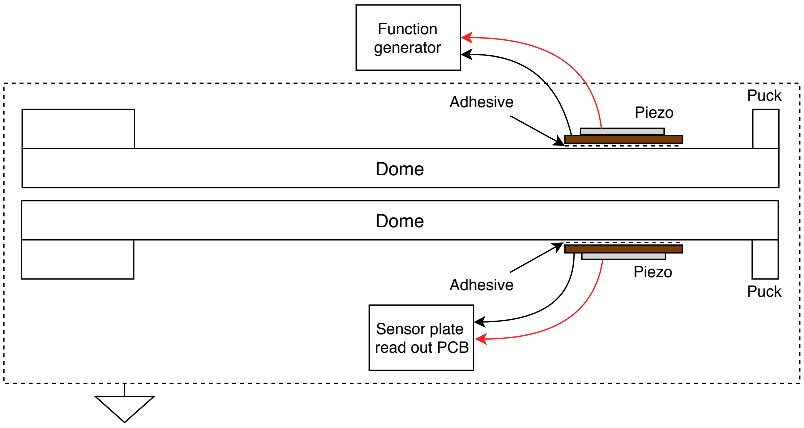


Figure 2.5: Schematic overview of the dome-on-dome setup. The gap between the two domes is only drawn for clarity. In practice the two domes are place on top of each other. The whole system is placed in a shielded environment to eliminate any interference

2.5.2. Results

The results of the test are shown in figure 2.6. In every measurement the excitation frequency can be distinguished, however the variation in adhesive materials seem to have a large impact on the results. Note that these results are not conclusive, but at least give a hint about the best of these three adhesive options.

2.5.3. Conclusion

The results are consistent with the expectation from a material standpoint, tape being the least rigid, then (still curing) epoxy and lastly the super glue. The sensor fixated with super glue shows the largest amplitude and the best consistency over various drive frequencies. Momo Medical was informed about these results and from this data, Momo determined that new versions will be fixated with super glue.

The main problem with using this setup for a testing device is that the surfaces need to be completely flat for a proper pressure transfer from one dome to another. Momo's current generation of sensor plates is

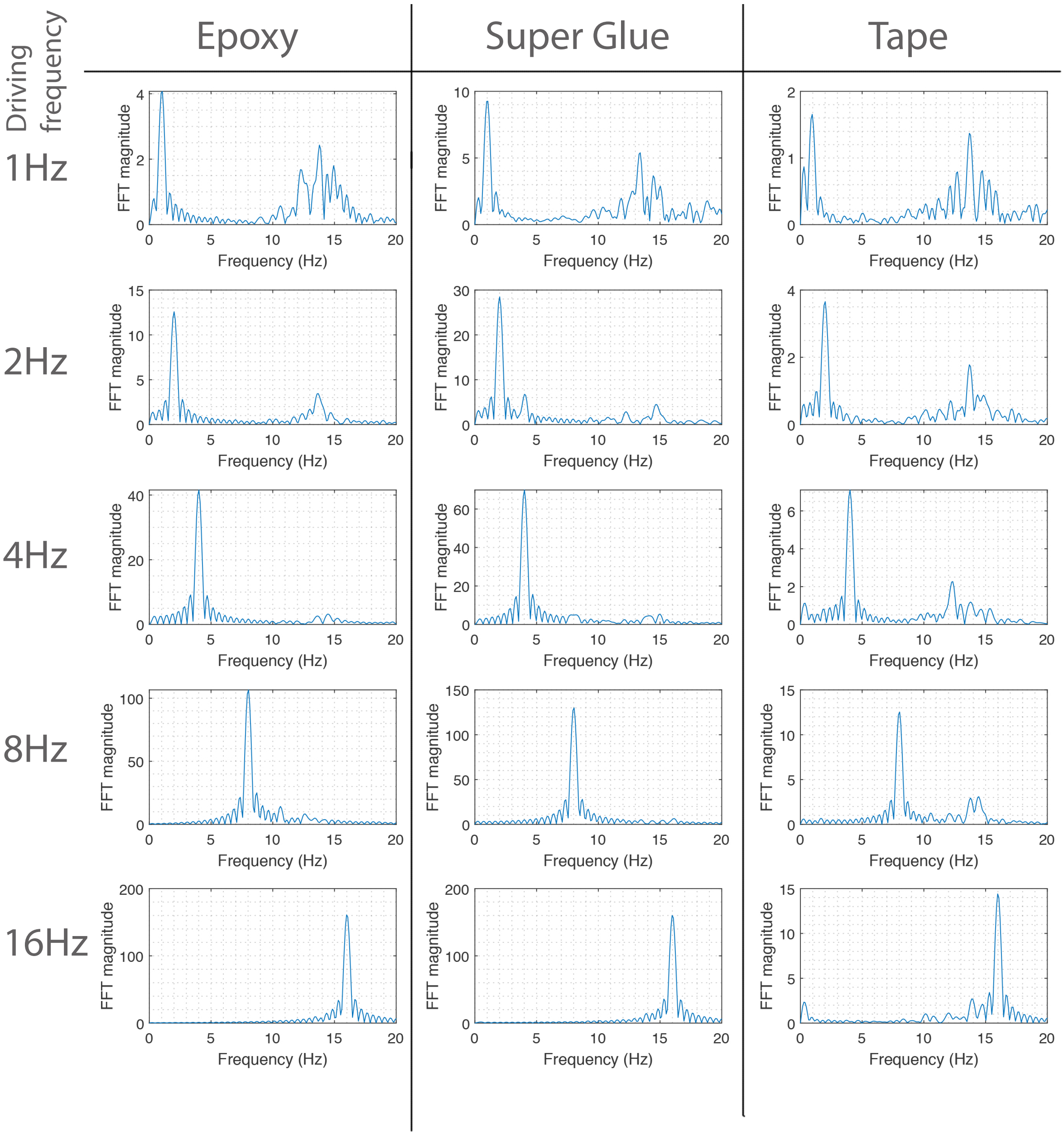


Figure 2.6: Frequency response for three different materials: Epoxy, Super glue and Tape. Each material is tested with 5 frequencies: 1, 2, 4, 8 and 16 Hz. A clear difference in amplitudes can be seen for the different binding materials and frequencies

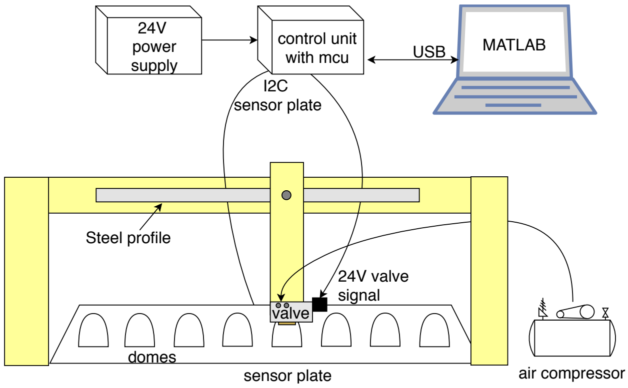


Figure 2.7: Schematic overview of the compressed air setup

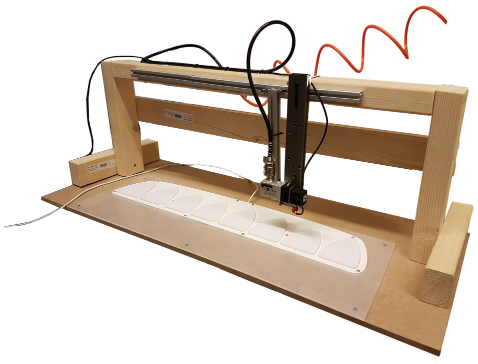


Figure 2.8: Setup with compressor connected to orange tube

made out of lasercut Polymethylmethacrylate (PMMA), which is not perfectly flat. If the upper dome does not contact the lower dome fully, proper testing cannot be done. Furthermore, using the piezo as a speaker as well as a sensor, it is unknown if the differences in response are the result of the speaker or the sensor. Therefore this method will not be developed further.

2.6. Compressed air

To deal with the repeating problem of a proper force transfer from an actuator to the dome, compressed air is considered next.

2.6.1. Test Setup

Using a test setup that can be seen in figure 2.8 pulses of air are shot towards a dome, and the output is shown in MATLAB. The compressor (Gamma CP-6 [18]) is connected to the orange tube (top right). The steady state pressure of the output pressure regulator is set to 3 bar. This output is then connected to a solenoid valve (Festo VUVS-LK25-M32C-AD-G14-1B2-S [19]). The valve has an opening time of 16 ms and closing time of 20 ms. In order to have a thin jet of air, an end cap with a 2.5 mm drilled hole in the middle, is screwed into the valve output connection. The valve is mounted on a rod, with an extra rod for support. For now, the valve is in a fixed position, above one sensor.

Schematically this setup is depicted in figure 2.10. The left part of the schematic shows the electrical system, consisting of the 24 V power input, the solenoid valve S1 between points 1 and 3, and the switch between points 3 an 4 for turning on and off the valve. The right part of the schematic shows the pneumatic system. From bottom to top one can see: pressure ‘ground’ (i.e. atmospheric pressure), the air compressor, the external pressure regulator with pressure gauge, the solenoid valve S1 and on top the air output nozzle.

The before mentioned microcontroller and MATLAB software is modified so it can open and close the valve from the microcontroller. In order to drive the 24 V solenoid a PCB was soldered containing a PN2222A NPN transistor connected to a MOSFET (IRFZ44N). A fly-back diode is added across the solenoid to eliminate induced voltage spikes when the circuit is switched, which could otherwise damage the MOSFET. A schematic of the circuit can be found in figure 2.9. The circuit is powered from a lab power supply set to 24 V. The test starts by the user pressing the run button, the parameters are sent to the microcontroller which controls the valve with the use of a digital output pin. The microcontroller waits one second before the valve is toggled. For all tests we used 5 Hz as the valve frequency with a total of 20 pulses.

2.6.2. Results

Multiple measurements were performed with this setup, many of them performed very well, yet some did not. This is different from the expected result, as it is expected that the air pulse and sensor do not vary greatly over time. In figure 2.11 three interesting results can be found, first a measurement that looks as expected. All

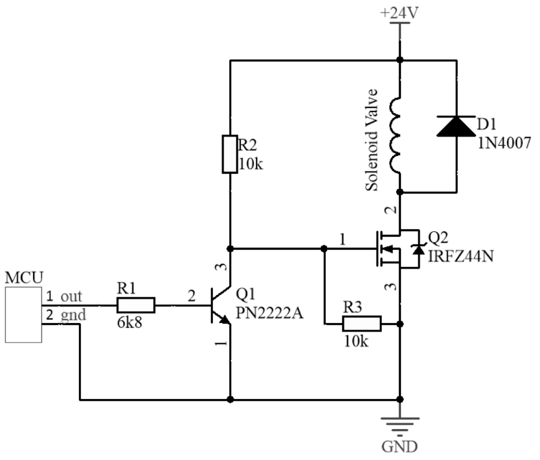


Figure 2.9: Schematic of 24 V solenoid valve driver, controlled from a microcontroller

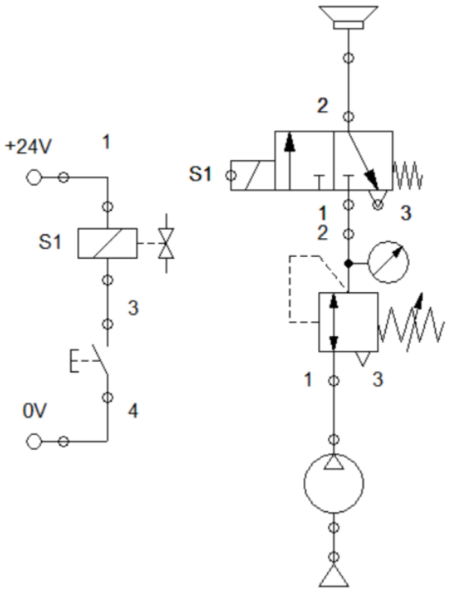


Figure 2.10: Electric (left); pneumatic (right) schematic. Picture provided by Roel van der Plas

periods are more or less equal and in the corresponding FFT the test frequency can be detected clearly. In the second measurement shown, a problem occurs at $t = 3.5$ s: suddenly many higher frequency components start to arise and the FFT becomes less clear. In the third measurement this problem persists, even when the measurements are performed multiple times and the FFT no longer shows its highest peak at the test frequency.

It has to be stated that the first and last results in figure 2.11 are repeatable ($n = 10$), whereas the second measurement was only observed one time. After this, we moved the valve and nozzle to another sensor in the same DUT, and the results are very similar to the first measurement in figure 2.11, only with a different amplitude. It is assumed that the tape holding the PE speaker in place has loosened or something similar, but the test setup seems to perform consistently.

2.6.3. Conclusion

The square wave frequency can clearly be seen from the FFT of the good measurements in figure 2.11. However, when the measurement shows a result similar to the bad measurement in figure 2.11, higher harmonics are present with higher amplitudes, so determining the test frequency is more difficult. The source of this problem has to be pinpointed exactly in order to do proper testing on the DUT. The repeatability of the good measurements however do seem to provide enough reliable data for a proper testing system.

2.7. Chosen concept

As mentioned in chapter 2, the steel beam and dome-on-dome setup have major drawbacks regarding the surface contact and the ability to use a reference. The compressed air setup however does not have this issue to the same extend. With this setup it is possible to place a different reference sensor under the nozzle in order to determine the consistency of the air exiting the system. The problems found using the setup are most likely problems with the sensor plate itself, but this will have the first priority for further testing. Therefor the compressed air setup is developed further for the final system.

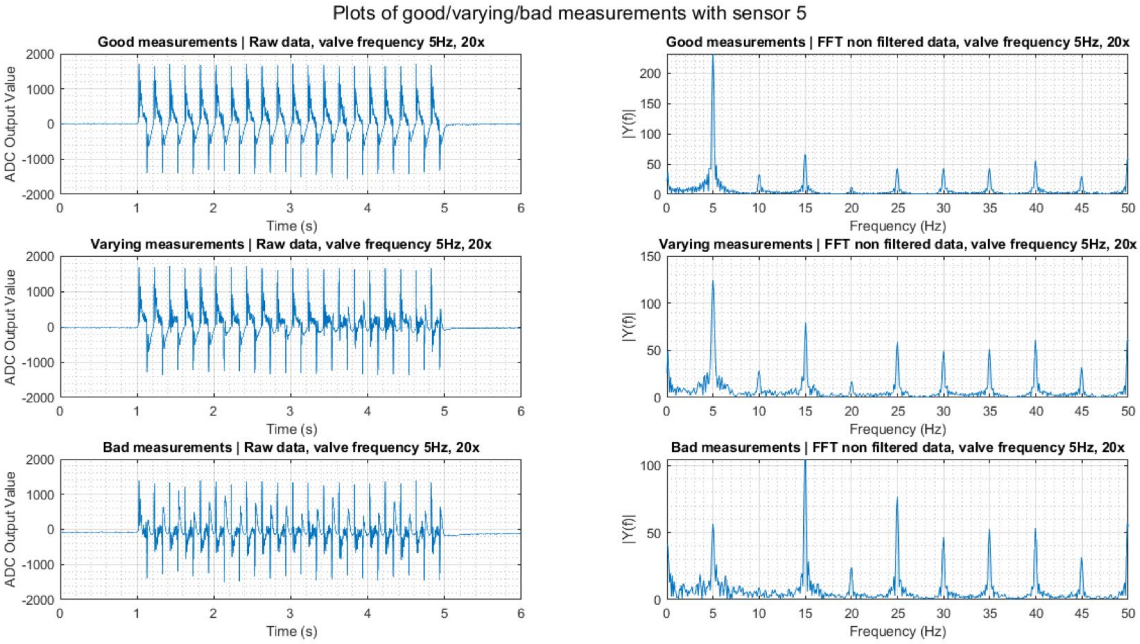


Figure 2.11: Measurement with compressor setup

3

Final Design

In this chapter the development of the final design will be explained.

3.1. Needed improvements

As mentioned in chapter 2, a pneumatic setup is chosen as the basis for the final design. The concept version had some major flaws, that lead to many needed improvements.

A general idea of the fluid dynamics involved is needed in order to determine if the system is stable from a mechanical standpoint. It should be determined if the data shown from the measurements is consistent with its theory.

When performing tests, the compressor sometimes switches on in order to provide enough air pressure for the system. This causes a slight ripple in the air pressure going to the valve. In order to reduce this issue, an extra pressure regulator (Festo LR-1/4-D-7-MINI) is connected to the system.

Before developing this idea further, the consistency of the generated air pulses has to be determined. For this reason a load cell will be placed underneath the valve nozzle and a 3 hour long test will be performed.

When it turns out these air pulses are consistent, it is possible to conclude that the DUT in itself causes the issue. Momo has called back their sensor plates in order to glue the PE sensors instead of using tape. Therefor it is more accurate to start testing with a sensor plate with glued sensors. As these sensor plates are a factor 10 more sensitive (according to Momo), the nozzle diameter is reduced to 0.7 mm and the air pressure to 1.5 bar. This version of the DUT is still considered version 5.

Lastly some major modifications have to be made to the structure of the test setup. All six sensors have to be tested, so either the DUT has to be moved, or the valve. Our colleague Roel van de Plas, a Mechatronics engineering student, will consider different concepts and together a final design will be chosen.

Testing can be done at any frequency between 0.5 and 10Hz, but the actual test frequency could be a single frequency. The frequency characteristics need to be determined for all sensors of the DUT in order to test the frequency characteristics of the sensors used by Momo. A test frequency or range can then be selected.

3.2. Theory

In order to get a better understanding of the amount of force applied to the DUT by the air pulse, fluid mechanics theory from [20] and [21] is applied to the system.

As a first approach, the nozzle on the output of the solenoid valve is modelled as a minimal Laval nozzle. The diameter of the nozzle is 0.7 mm. To see if the air escaping the nozzle is choked, the downstream pressure p^*, which is the normal atmospheric pressure of 0.1 MPa, should be no less than the critical ratio seen in equation 3.1. The pressure P_0 equals the upstream pressure, which is the 2.5 bar (0.25 MPa) pressure applied to the valve. For dry air, the heat capacity ratio $\gamma = 1.4$.

$$\frac{p^*}{P_0} = \left(\frac{2}{\gamma+1}\right)^{\frac{\gamma}{\gamma-1}} = 0.528 \tag{3.1}$$

Since $p^* \leq 0.528P_0$, choked flow occurs when the air escapes the nozzle of the valve. This means the air is choked at Mach 1. Equation 3.2 is used to determine an approximated air velocity. In this equation, T_0 equals room temperature (293K) and the gas constant $R = 287\,\mathrm{J\,kg^{-1}\,K^{-1}}$.

$$v = \sqrt{\gamma R T_0} = \sqrt{1.4 \cdot 287 \cdot 293} = 343\,\mathrm{m\,s^{-1}} \tag{3.2}$$

For the mass flow rate of the air escaping the nozzle, equation 3.3 is used, where the discharge coefficient $C_d = 0.8$ is chosen to compensate for any irregularities in the nozzle hole. The nozzle area $A = \pi r^2 = 3.85 \times 10^{-7}\,\mathrm{m}^2$, air pressure $P_0 = 0.25\mathrm{MPa}$ and compressed air density $\rho_0 = \sqrt{P_0/(RT)} = 3\mathrm{kg\,m^{-3}}$.

$$\dot{m} = C_d A \sqrt{\gamma \rho_0 P_0 \left(\frac{2}{\gamma+1}\right)^{\frac{\gamma+1}{\gamma-1}}} \tag{3.3}$$

This results in a mass flow rate $\dot{m} = 1.83 \times 10^{-4}\,\mathrm{kg\,s^{-1}}$.

To calculate the force, we use the relationship $F_{calc} = \dot{m}v = 1.83 \times 10^{-4} \cdot 343 = 62.6 \times 10^{-3}\,\mathrm{N}$. To compare this calculated force against the used setup, a digital weighing scale is placed underneath the air nozzle and a constant air flow is released. The scale measured a constant 6.6 g, which equals a force of $F_{meas} = mg = 0.0066 \cdot 9.81 = 64.7 \times 10^{-3}\,\mathrm{N}$. The air pressure will not be changed during the rest of the measurements in this thesis. This also means that system requirement **M2** is satisfied (see table 1.1).

3.3. Reference measurements

3.3.1. Reference setup

As mentioned in section 3.1 a load cell is added to the system in order to obtain a good reference measurement. A load cell based scale is constructed out of PMMA and 4mm MDF and can be seen in figure 3.2. The load cell used is the TAL221 made by HTC-Sensor [22]. It is bolted together using M3 screws and bolts. The relevant specifications for the load cell can be found in table 3.1.

Table 3.1: TAL221 load cell specifications

Capacity	500 g
Rated output	0.7 ± 0.15mV/V
Combined error	±0.05%FS

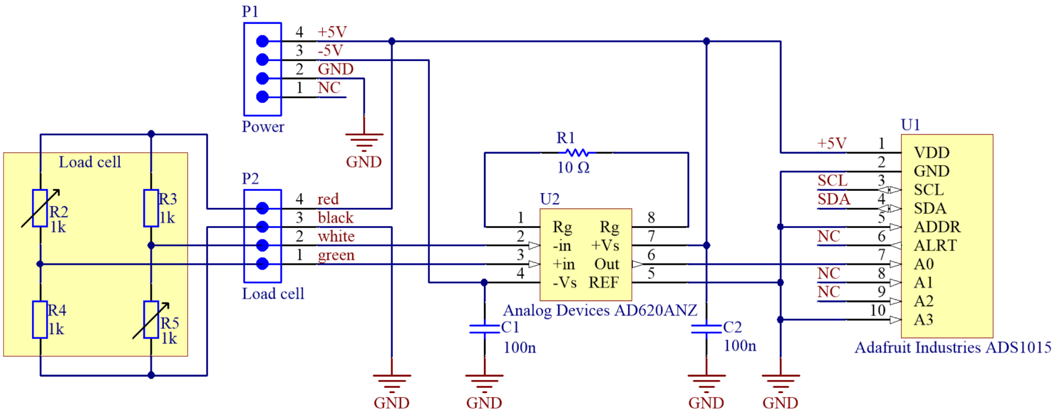


Figure 3.1: Schematic of load cell, read-out with the AD620 instrumentation amplifier and connected to the ADS1015 analog-to-digital converter. The I^{2}C lines from the ADC are connected to the microcontroller (not shown in this schematic)

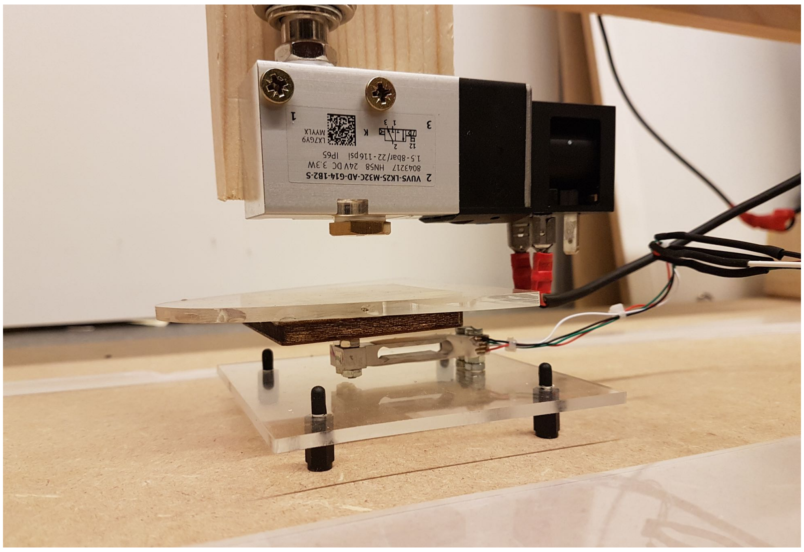


Figure 3.2: Load cell sensor mounted on a PMMA base and with an MDF and PMMA top in the same shape as a single dome on the DUT. The brass nozzle with the 0.7mm air hole can be seen underneath the Festo valve

3.3.2. Load cell linearity

In order to use the load cell as a reference for the final setup, it is necessary to determine the linearity of the load cell. To test this, different weights are placed on the load cell and the ADC values are read out. Weight from 5g to 100g are used, and with Newton's second law $F = mg$, where the standard gravitation acceleration $g = 9.81\,\mathrm{m\,s^{-2}}$. Figure 3.3 shows the output ADC values for the different applied weights. It can be seen that the load cell acts very linear. This means the load cell can be used to convert the ADC values to a force.

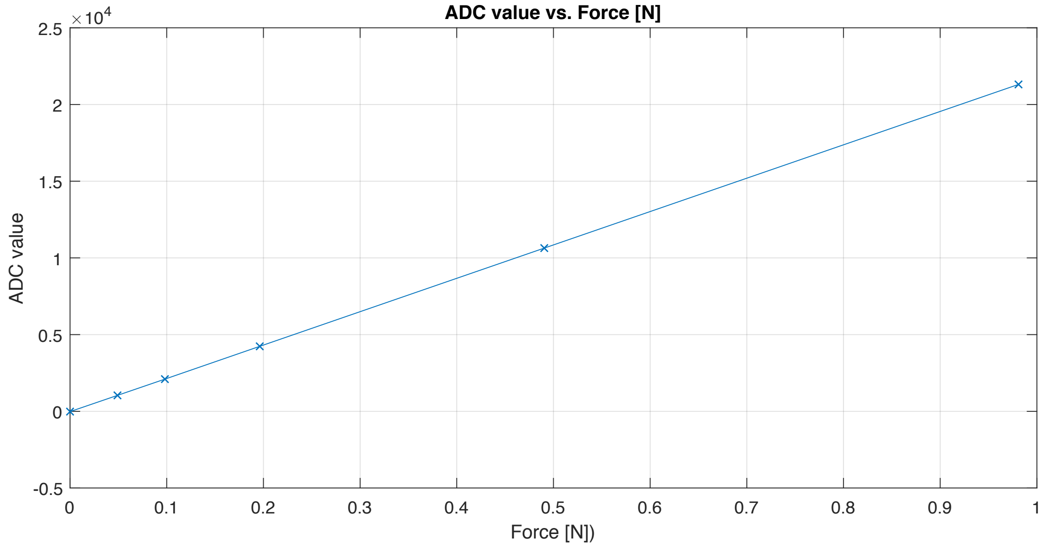


Figure 3.3: Mean ADC output values versus static force on the load cell. The measured values are shown as crosses

3.3.3. First measurements with the load cell

A first measurement is shown in figure 3.4 (top left). The result has many higher frequency components and does not seem to be the an approximated square wave.

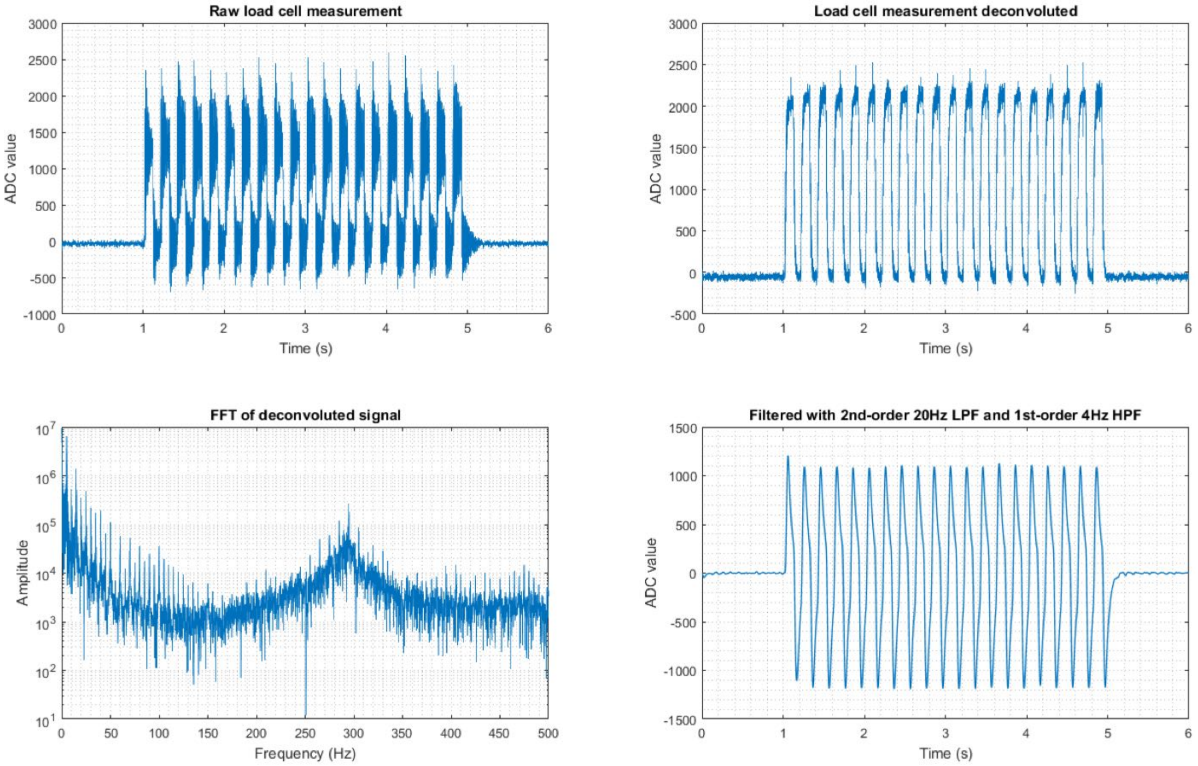


Figure 3.4: Results of the load cell test. Upper left: raw load cell measurement with a 5 Hz input. Upper right: deconvoluted measurement using the curve fitted impulse response from figure 3.5. Lower left: FFT taken from the deconvoluted measurement. Lower right: filtered deconvoluted measurement taken with the same filter used in the DUT

When looking closely at these higher frequency components, ringing can be seen at $t = 5\,\mathrm{s}$. This ringing could come from the mechanical system of the load cell and/or the air pulse itself. The ringing frequency is determined at $f_d = 99\,\mathrm{Hz}$. The load cell system can be modelled as a damped mass-spring system. This is a damped oscillator with a frequency determined by the spring constant k [$\mathrm{N\,m^{-1}}$], its mass m [kg] and the damping ratio ζ. The damped natural frequency ω_d can be calculated via $\omega_d = \sqrt{\frac{k(1-\zeta^2)}{m}}$. The impulse response of an ideal damped oscillator is of the form

$$y = \mathrm{A}e^{-t/\tau}\sin(\omega_d t) \tag{3.4}$$

To test if this load cell mass-spring system is responsible for the ringing seen in the data, a weight of 50 g is placed on the load cell and removed very quickly when a measurement is running. The resulting ringing is multiplied by -1 to obtain an approximated step response, so it can be differentiated in order to find the impulse response. This ringing can be curve fitted using equation 3.4. In figure 3.5 the measured ringing and the curve fitted damped sine are plotted (script in appendix D.2). The used parameters for equation 3.4 are $\mathrm{A} = 2000$, $\tau = 0.05\,\mathrm{s}$ and $\omega_d = 2\pi \cdot 99 = 622\,\mathrm{rad\,s^{-1}}$.

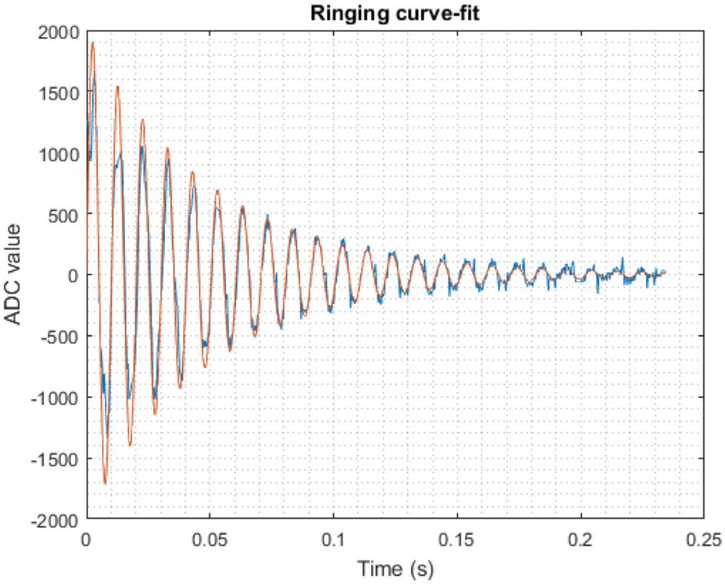


Figure 3.5: Curve fitting the ringing. The blue line represents the calculated impulse response from the measured step response. The orange line is the curve fitted damped sine used for the deconvolution

The measured signal can be seen as the result of a convolution between the load cell system and the air pulse system. Mathematically this is described by equation 3.5.

$$y(t) = h_{lc}(t) * f(t) \tag{3.5}$$

In this equation, $y(t)$ is the measured output from the load cell, $h_{lc}(t)$ is the impulse response of the load cell mass-spring system and $f(t)$ of the air pulse system.

It is now possible to deconvolute the measurement of the load cell with the impulse response of the load cell to obtain data without the ringing of the load cell. Since a deconvolution is very sensitive to noise ([23]), the curve fitted sine wave is an ideal approximation of the load cell system impulse response. The result of this deconvolution can be seen in figure 3.4 on the top right. When looking at the FFT of this deconvoluted signal, a peak at 294.8 Hz can be detected (script in appendix D.6). It turns out higher frequency components are still present, which can originate from either the air or inaccuracies in the modeling of the mass-spring system.

Fortunately, as mentioned in section 1.2.3, the DUT has internal filtering. When modeling the piezo speaker and its signal conditioning circuit, it was determined that the system has a first order high-pass filter around 4 Hz and a second order low-pass filter around 18 Hz. When filtering the deconvoluted data using these conditions, the bottom right plot in figure 3.4 can be shown.

Comparing this to the upper measurement in figure 2.11, it is very similar. It is therefor assumed that air pulses are consistent enough to be able to do proper testing.

Lastly, the damped natural frequency $f_d = 99\,\text{Hz}$ and the used test frequency is $f_{test} = 5\,\text{Hz}$. Since $\frac{f_{test}}{f_d} = 0.05 \ll 1$, we can neglect the frequency behaviour of the load cell.

3.3.4. Load cell repeatability

For approximately 3 hours, every 90 seconds a test is performed to determine the repeatability of the load cell, 128 measurements in total. From every measurement, the peak FFT value at 5 Hz test frequency is taken and plotted in figure 3.6. In this figure the mean value and the standard deviation are shown to give an indication about the consistency of the FFT amplitude. The used script can be found in D.4.

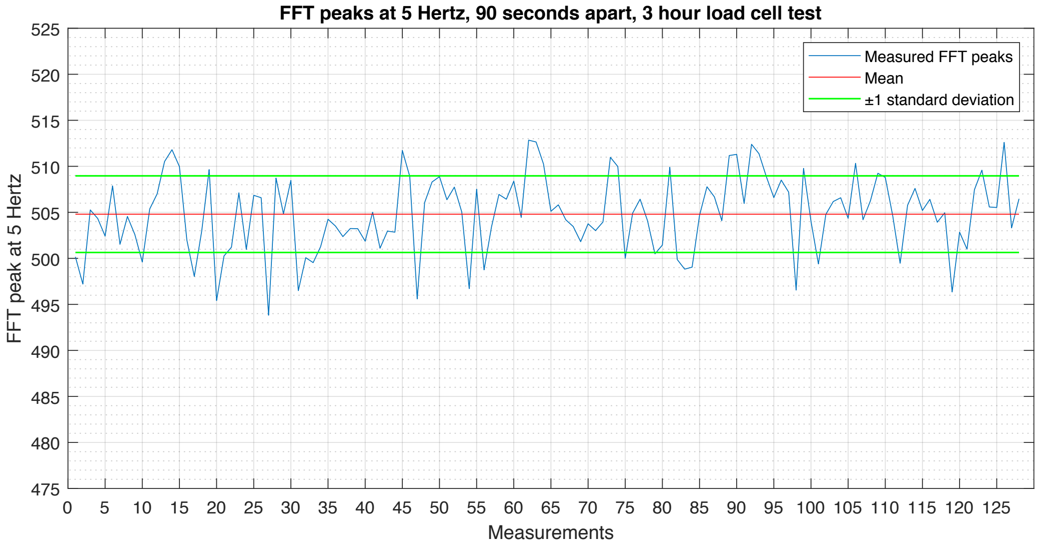


Figure 3.6: FFT peak values at 5 Hz during a ±3h test

3.3.5. Load cell frequency response

For testing purposes it is convenient to use a single test frequency. In order to be able to tell if a sensor is ‘good’ or ‘bad’ based on one test frequency, a test is performed on all sensors of the DUT.

First the frequency characteristics of the air pulse are examined using the load cell. Each test consists of a measurement with 20 pulses of a certain frequency. The frequencies used go from 0.5 Hz to 12 Hz. Lower frequencies might take too much time during production and higher frequencies are not reachable with the current pneumatic valve. As the pulse is not a perfect square wave, not all frequency components are present at the same amplitude. In table A.1 the peak FFT values for every test frequency are shown. The test is repeated 5 times and the measurements are averaged for further analysis. These averages can also be seen in table A.1 together with the percent deviation of the single measurements. All deviations are within 2% of the average. It can be seen that there are large differences in the response for the different test frequencies. The FFT peak at 12 Hz is $(\frac{735.1}{411.6} - 1) \cdot 100\% = 79\%$ higher than the peak at 7 Hz. To compensate for these differences, a scaling factor is calculated where all averages are scaled relative to the FFT value at 0.5 Hz. These scaling factors can be seen in table 3.2.

Table 3.2: Frequency scaling factors used to compensate for the frequency characteristics of the air pulse

Frequency (Hz)	Scale factor
0.5	1.000
1	1.016
2	1.029
3	1.560
4	1.042
5	1.361
6	1.577
7	0.949
8	1.063
9	1.408
10	1.453
11	1.531
12	1.695

3.4. DUT measurements

3.4.1. Schematic setup

In figure 3.7 the setup used for the final testing system can be seen. In a wooden beam, holes are drilled exactly above each sensor. Since only the inner 6 domes contain piezoelectric sensors, no holes are drilled above the first and the last dome. An extra hole is drilled above the load cell, so the reference measurements can be incorporated in the same system. The DUT can be placed underneath the wooden frame and its position is fixed by a raised edge. To have the least amount of variables, the protecting sleeve which is normally fitted around the DUT when used in the field will be removed. This also satisfies requirement **M1**.

The pneumatic solenoid valve is attached to a wooden rail where a hole with the same diameter is drilled on the top side. With bolts the rail with the valve can be positioned above each sensor or above the load cell. The darker yellow wooden slats are used to fix the position of the valve, so the air won't be blowing on the domes under an angle. The nozzle height is 1.4 cm and this distance is the same for all PE sensors and for the load cell. The load cell is fixed to the base plate to ensure no extra vibrations are induced in this system.

The control unit forms the core of the system. In the control unit, the load cell amplifier circuit (shown in figure 3.1), the solenoid valve driver (shown in figure 2.9) and the LPC1768 microcontroller are placed. A bench power supply is used to supply the +5 V and −5 V to power the load cell and load cell amplifier, and a power adapter provides the 24 V which is used to drive the solenoid valve. MATLAB is used to control the microcontroller via USB.

The air compressor has an internal pressure regulator, and the air hose is connected from this pressure regulator to a second pressure regulator to remove any pressure fluctuations when the pressure of the compressor drops.

The complete setup could be placed in an area of 1.1x0.5x0.5m. This is excluding the air compressor, since compressed air is usually available at production sites where the system will be placed. Requirement **M4** is satisfied with this area. Since the setup is a concept, no care is taken to meet CE and RoHS requirements (**C5**).

A small calculation is done to provide some information about the amount of power used by the complete setup. The solenoid valve in on-state consumes 3.3 W according to the datasheet. The microcontroller and ADCs only use a couple of milliampères at 5 V, so the consumed power is in the order of tens of milliwatts. The same holds for the load cell and its amplifier circuitry. The used laptop has a rated maximum power of 90 W. The biggest power consumer is the air compressor when turned on, which has 1100 W as its rated maximum power. Even with the air compressor taken as a part of the setup, the total power still falls below the maximum requirement given by **M8**.

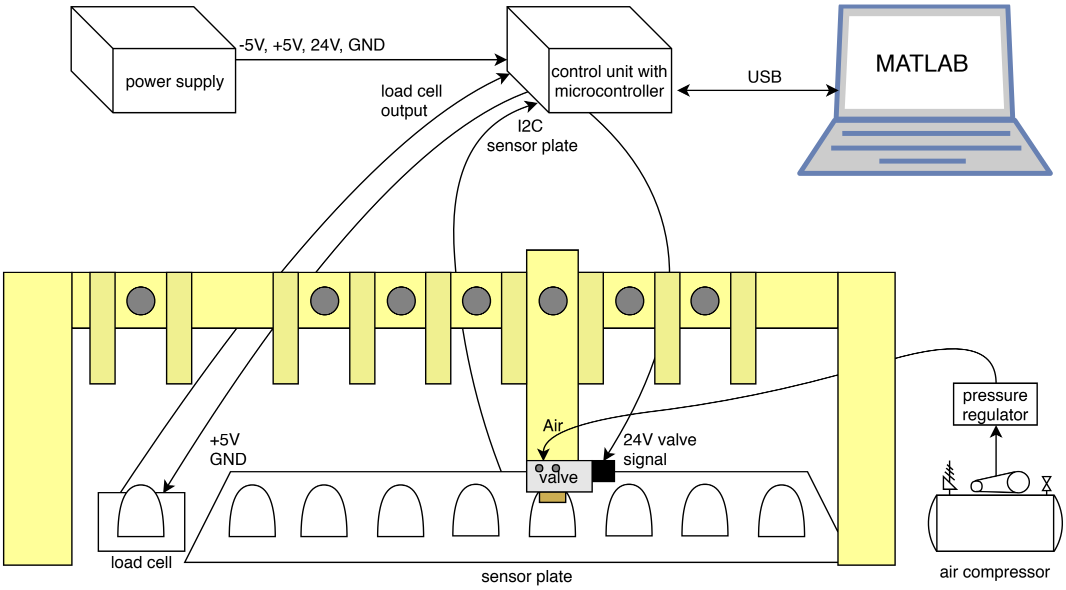


Figure 3.7: Complete overview of the setup used

3.4.2. Measurement software

The measurement software has as its main tasks to regulate the amount of air to the sensors, reading out the sensor values from the DUT and to determine if a sensor is deemed to be unsuitable for use. The software allows the user to change certain parameters through an interface. Based on the measurement type, results are shown in graphs or tables, or both.

The software used to measure the sensors are MATLAB App Designer and Mbed as aforementioned in section 2.3. The program (using MATLAB App Designer) controls certain configurations inside the microcontroller and processes the data which it receives from the microcontroller. The microcontroller (using Mbed platform) communicates with the PC to receive the user configuration settings. These configuration settings will be used to control the valve and the DUT. The code of the MATLAB program can be found in E, the code used for the microcontroller is found in F. A Nassi-Shneiderman diagram is made of the MATLAB program and shown in figure 3.8.

The microcontroller communicates through I^2C with three ADCs. Two ADCs are from the DUT and one is used to read out the load cell. Furthermore a pin is used to control the valve circuit. For each ADC, the gain factor of the PGA is set at the start of a measurement based on the user configuration. The sample frequency argument is used to set the frequency to which the ADC is reading out. The sensor number is used to determine which ADC and its channel is read out. The duration parameter is used to determine for how long the microcontroller is reading the ADC for the sensor values. The valve frequency and the amount of times it should open and close is used to determine when the pin out to the valve circuit should be high or low. At last the choice can be made to have a pulse or a step as output signal, which is useful for the load cell.

The main program has three measurement types. A single test is used to measure a single sensor once. The ‘continuously’ test is used to continuously measure a certain sensor for the purpose of consistency testing. The last measurement type is the full test, which will be the main test used by Momo Medical.

The full test starts with measuring a sensor once to determine a suitable gain factor, after which the program will start a measurement with the determined gain factor. Once the measurement is done, the next sensor is read out. As a reference, the load cell is measured at the start and at the end of a session. A constant amount of air is applied to the load cell, whereas pulses are applied to the piezoelectric sensors based on the valve frequency and the amount of times it should open and close. The result between the measurement of the load cell at the start and at the end should result in a similar plot if the air is indeed constant. If there are any differences in these two measurements, it means the sensor values from the measurement session are inaccurate and the measurements should be redone. If the results of the two load cell measurements are similar, the program will analyze the data based on the frequency response of the signal. The dominant frequency and its value of each sensor is determined and placed in a table. The value at the dominant frequency is compared to a certain threshold value to determine if a sensor is acceptable or should be replaced. The threshold value is adjustable by Momo Medical. Along with a pass/fail indication, scaling factors are determined to compensate for the differences in sensitivity. Using these scaling factors, the ADC value of each sensor will be brought closer to each, but the accuracy will drop when the scale factor is too high.

In the final design of the program, only the full test is of importance. A separate panel in the program accessible by Momo Medical, contains the other two tests in case additional measurements are needed, along with the configuration settings and a log system to follow each major step inside the program. Depending on Momo Medical's wishes, they can decide to include this into the version used by the production worker.

In order to comply with the system requirements, several features are added into the program. The sampling frequency is adjustable by the user, with a range between 50 to 2500Hz (thus **M3** is satisfied). The main program only consists of three buttons, for ease of use. As the production worker must understand what the buttons do, additional information will be provided in the user manual of the program, which can be found in appendix C (thus satisfying **M5** and **M9**). Pressing the 'Run Full Test' button starts the test, the user is not required to change configuration settings (satisfying **S1**). Besides buttons, visual items are included in the program such as colored circles and a status bar, to follow the progress of the test. Data is received from the microcontroller and processed in the program. At the end of a full test a pass/fail indication is shown for each sensor through a red colored circle if a sensor is rejected, or a green colored circle in the case it passed the test (satisfying **M6**). The measurement data and scaling factors are saved after each full test in a specified location, such as an external USB storage device (satisfying **M7** and **S4**). Data is not saved to a cloud based solution (satisfying **W1**).

The duration of a full test depends on the duration of each measurement. The minimum amount of measurements in one session is sixteen, as each sensor is measured once to determine the gain factor, and once to measure the signal with the determined gain. As it may happen that the maximum value of the signal is close to the full scale range of the ADC with the used gain factor. A higher gain factor is chosen in that situation. On average a full session requires eighteen measurement. Moving the valve to the next sensor takes around fifteen seconds with seven movements, resulting in 105 seconds in total for moving the valve. The duration for each measurement depends on the default value. Accounting for the data processing and pressing the buttons, a total average of eight seconds (six seconds for the measurement itself, two seconds for the data processing and pressing on buttons if necessary) results in 144 seconds for the program to run a session. Combining these two results in 249 seconds, in other words four minutes and nine seconds. This amount is less than the required thirty minutes, making it satisfy **M12**.

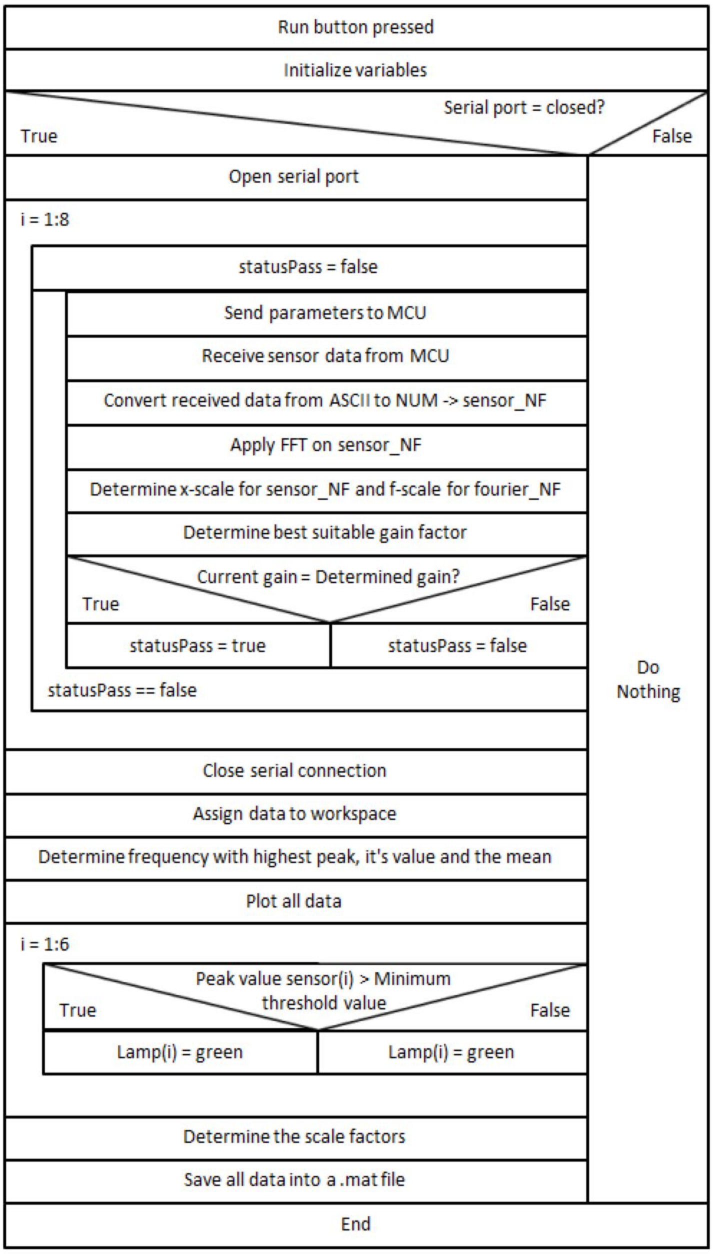


Figure 3.8: Nassi-Shneiderman Diagram of the MATLAB program

3.4.3. DUT frequency response

As the frequency characteristics of the air pulse are known, a complete DUT can be tested over the frequency range of interest. The results are plotted in figure 3.9 using D.5. Apart from a difference in amplitude, the frequency response looks very similar for all 6 sensors. Because of the band-pass filter in the conditioning circuit of the piezoelectric sensors (shown in figure 1.4), the high-pass characteristic seen in the results are to be expected. A zoomed-in version of figure 1.5 is shown in figure 3.10.

As all sensors respond very similar to all frequencies, 5 Hz is chosen as the testing frequency for the system.

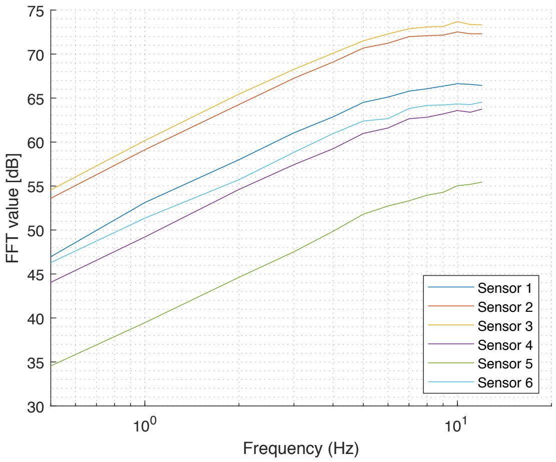


Figure 3.9: Frequency response of a complete sensor plate (6 PE sensors)

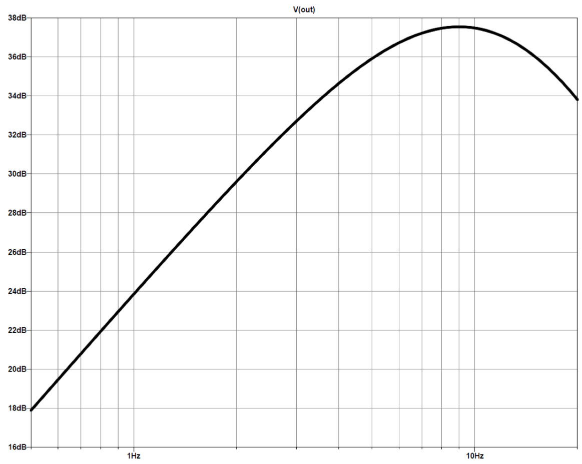


Figure 3.10: Zoomed in version of the frequency response of the conditioning circuit shown in figure 1.4

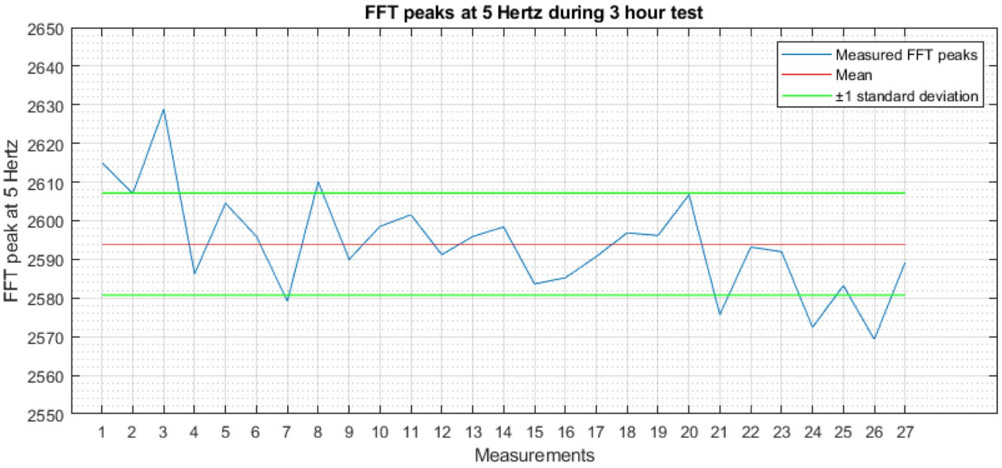


Figure 3.11: FFT peaks from sensor 3 of the DUT at the test frequency $f_{test} = 5\,\text{Hz}$ measured every 5 minutes for 3 hours

3.4.4. DUT repeatability

A long term test is performed to determine the repeatability of testing the DUT. It makes no sense if the DUT has passed the test but reacts very differently a few moments later. A single sensor of the DUT is tested every 5 minutes for 3 hours to see if the output drifts over time. Sensor 3 is chosen for this test since that sensor has the highest output for the same test frequencies (as can be seen in figure 3.9). To compare every measurement, the FFT peak at the 5 Hz test frequency is taken and plotted in figure 3.11.

A small downward drift can be seen over the complete time span. The mean value $\mu = 2594$ is also drawn in the figure, together with the standard deviation $\sigma = 13.2$ to give a good indication of the maximum deviation from the mean to the measured peak values. The maximum deviation from the mean is with measurement 3. The percent deviation in this case is $\frac{2629-2594}{2594} \cdot 100\% = 1.4\%$.

4

Results

In this chapter the device precision is derived and a full test is performed on the DUT.

4.1. Device precision

To determine the precision of the the total system, first the precision of the air pulse on the load cell is taken. From figure 3.6 the mean and the standard deviation can be calculated. The mean $\mu = 504.8$ with a standard deviation $\sigma = 4.16$. This results in a coefficient of variation of $c_v = \frac{\sigma}{\mu} \cdot 100\% = \frac{4.16}{504.8} \cdot 100\% = 0.82\%$.

The maximum deviation from the mean is during measurement 27, where the measured FFT peak is at 493.8. This results in a percent deviation $|\frac{493.8-504.8}{504.8}| \cdot 100\% = 2.2\%$. This means requirement **S2** is met.

4.2. Full DUT test

On December 13th at 22:43, a full system test is performed using the DUT.

The result can be seen in figure 4.1. Through individual sensor tests, the response of each sensor to an applied force is clearly observable; there are large differences in amplitude between the sensors. In order to get an idea of the sensitivity of each sensor, the measurement of each sensor is plotted next to the other in order to make a good comparison. Sensor 5 is rejected because of poor performance using the arbitrary FFT peak threshold (set at 800).

The data collected during the measurement is used to analyze the DUT. The results are then shown in the analyze panel in the program depicted in figure 4.2.

4.3. Device costs

The costs for the device are approximately €3186,92. An overview of the different costs can be found in table B.1. If the production company has an available laptop and/or MATLAB license, the costs of these can be omitted. The same goes for the air compressor if a compressed air line is available at the production facility. Any way, the cost is less than €10.000 and thus **M11** is satisfied.

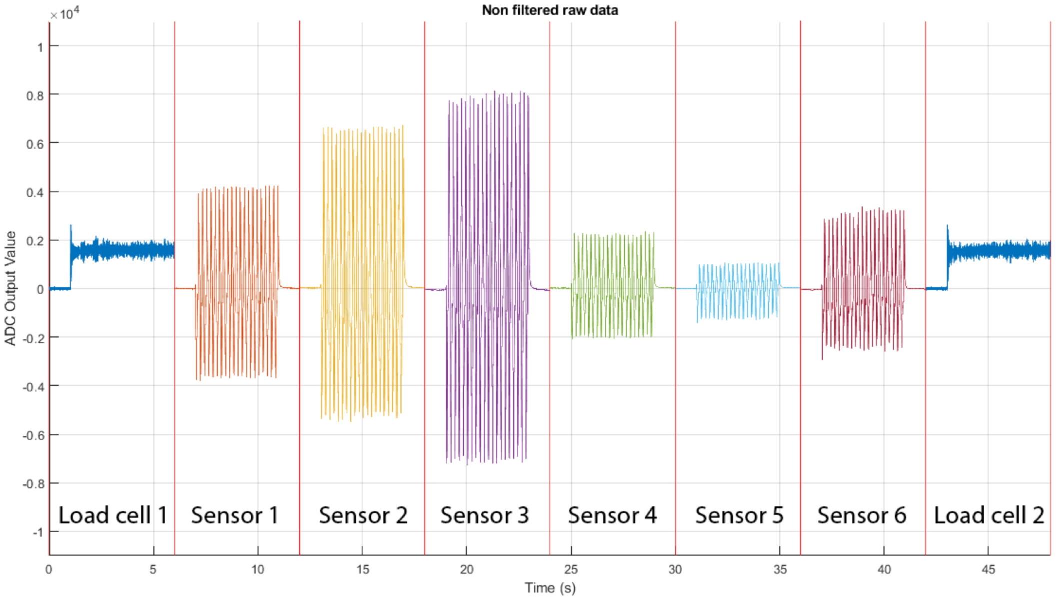
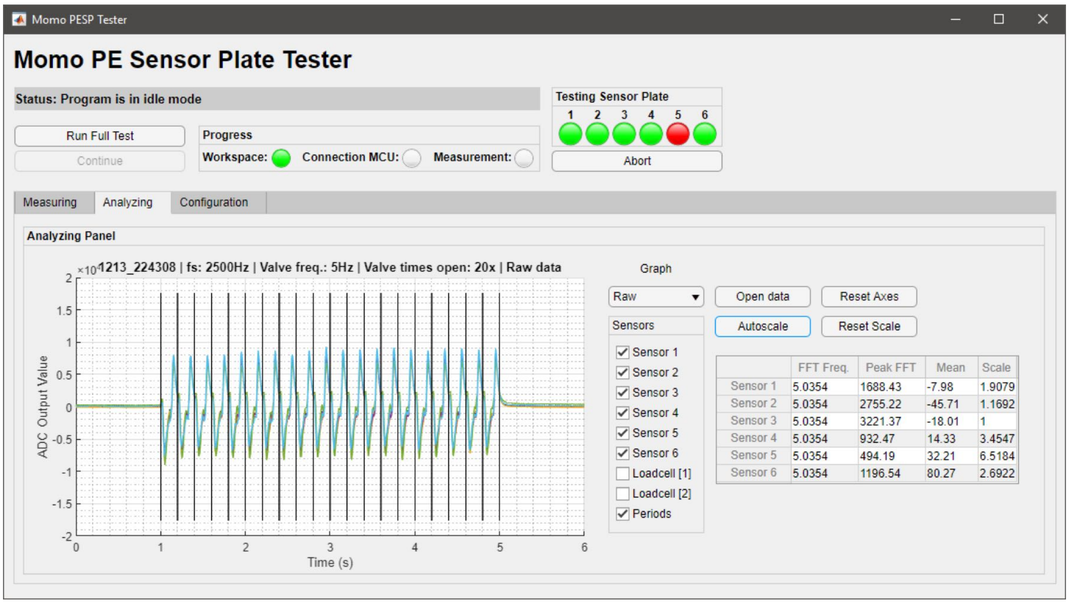


Figure 4.1: Plot of a full test, the measurements of each sensor are placed next to each other



Status: Program is in idle mode

Workspace: ● Connection MCU: ○ Measurement: ○

Testing Sensor Plate

1 ● 2 ● 3 ● 4 ● 5 ● 6 ●

1213_224308 | fs: 2500Hz | Valve freq.: 5Hz | Valve times open: 20x | Raw data

Raw ▾

Sensors

☑ Sensor 1
☑ Sensor 2
☑ Sensor 3
☑ Sensor 4
☑ Sensor 5
☑ Sensor 6
☐ Loadcell [1]
☐ Loadcell [2]
☑ Periods

	FFT Freq.	Peak FFT	Mean	Scale
Sensor 1	5.0354	1688.43	-7.98	1.9079
Sensor 2	5.0354	2755.22	-45.71	1.1692
Sensor 3	5.0354	3221.37	-18.01	1
Sensor 4	5.0354	932.47	14.33	3.4547
Sensor 5	5.0354	494.19	32.21	6.5184
Sensor 6	5.0354	1196.54	80.27	2.6922

Figure 4.2: Analyzing the measurement data in the MATLAB program

5

Conclusion

From the three concepts seen in chapter 2, the compressed air method is chosen to be developed in detail in chapter 3. This method does not have the issue regarding surface contact between the actuator and the DUT, and can be used with a reference.

From the measurements in chapter 4 it is concluded that the device is able to properly measure the DUT. The precision of the measurement device has a coefficient of variation $c_v = 0.82\%$ which is well within the system limits.

The test devices performs as intended, and all parts work together nicely. Testing is performed on the sensor plate without its protecting sleeve as intended, no weight limits are exceeded, and the test does not exceed the testing time of 30 minutes. The sampling rate is set to $F_s = 2500\,\mathrm{Hz}$ and is therefor more than enough for proper testing. The devices stayed within its size and cost limit, and is operable by a production worker. Testing starts by pressing one button, all calculations are performed in software and a pass/fail indicator is displayed on the screen. Raw data is exported for Momo to be able to do a more detailed analysis later on. The device is powered from one power cord and does not draw too much (>16 A) current. User documentation is provided for users who wish to know more about the software. All tests were performed using version 5 of Momo's sensor plate using the existing I^2C connection, although the piezo sensor fixation was altered in chapter 3. The device performs a self-test when the first measurement is started. In short, most important requirements have been met.

6

Discussion and Recommendation

Nothing is perfect, and discussion and improvements are always possible. This is also the case with this project and its limited time span. In this chapter some possible improvements are discussed.

6.1. Discussion

As the air pulse is noisy, measurements at frequencies much higher than 10 Hz could result in an inaccurate system. Moreover, at these frequencies the rise and fall time of the valve become a major part of the total period, so frequencies higher than 12 Hz cannot be tested at all. If the system should be able to test using higher frequencies, more investigation is needed into the air characteristics of the pulse and a faster valve is needed.

For our current setup, it is assumed the production worker plugs in the DUT to the microcontroller before starting any tests. The software is written based on reading out the ADC values from the sensor plate. If no sensor plate is connected to the microcontroller, the 'read' action still happens. Of course the microcontroller is not able to read any data, but this is not known by the MATLAB program. This issue can be resolved by implementing a check function in the microcontroller, by not allowing any connection to MATLAB before the sensor plate is connected.

The main program has an abort button, used to terminate the current running process. However, this abort button only terminates the processes within MATLAB. The valve control inside the microcontroller still operates whenever this button is pressed. During our tests, the workaround used for this issue is unplugging the USB cable of the microcontroller and plugging it back into the laptop if the test should be stopped. The software can be improved by sending a stop signal to the microcontroller if the abort button is pressed. The stop signal should halt any actions until the system is reset.

6.2. Recommendations

Currently, MATLAB is used for the testing program. This requires a MATLAB license and a dedicated PC. These items are expensive and do not provide a neat embedded solution. In order to realize this, the software that is currently written in MATLAB, could be programmed into a microcontroller. In addition to this, the buttons currently used inside the program could be replaced by physical buttons.

The device could be modified in order to simplify the operation. A linear movement system could move the valve in one direction (impression in figure 6.1), or multiple valves could be connected. The first option has more moving parts, the second option requires a movement of the load cell between each nozzle. This last problem could be solved if the load cell (or other sensor) is integrated into the valve suspension. Thus, at this point in time, requirement **S3** is not satisfied. In both cases, the software needs to be updated to support these new features.

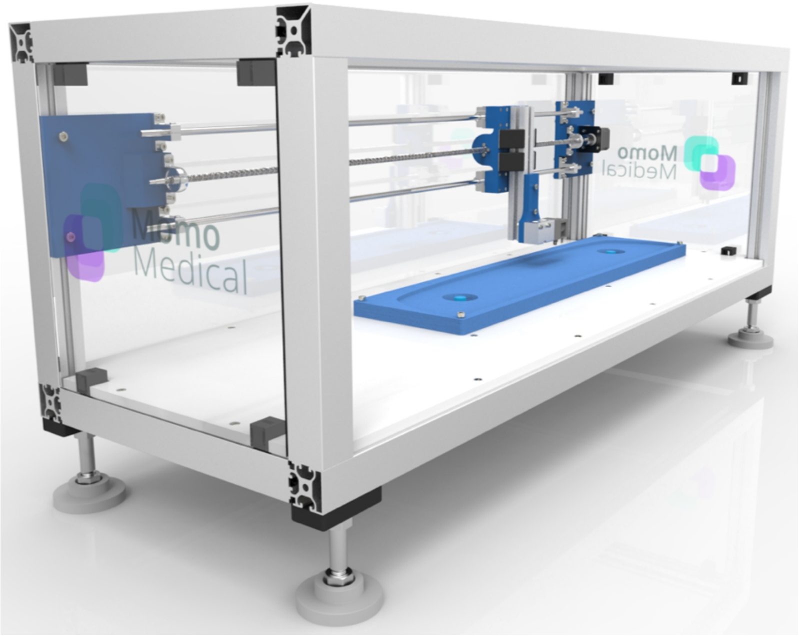


Figure 6.1: Device with linear movement system. Picture provided by Roel van der Plas

If the FSRs also need basic testing, multiple pressures could be selected when using a configurable pressure regulator. As long as the averages are taken over a longer period of time, the calibration should be able to reach a 5% error margin. This would mean the system can calibrate both types of sensors, and only one calibration device is needed. As this is not yet implemented, requirement **C1** and **C2** are not satisfied.

In order to provide real-time feedback of the air pulse to the system, a sensor could be mounted to a new valve mount to sense the air as it is protruded from the nozzle (schematically shown in figure 6.2). The difficulty with this is the fact that the valve switching also produces a relatively large amount of force that is measured as well. A first attempt at this was made, as can be seen in figure 6.3. Unfortunately the results showed a large peak from the mechanical switching of the valve, but no significant output of the force of the air pulse.

In the final setup in this thesis, no static force is exerted on the sensors (i.e. a preload). When a small preload is present on a PZT PE sensor, the sensors' sensitivity could improve and the impact of this change could be investigated [5].

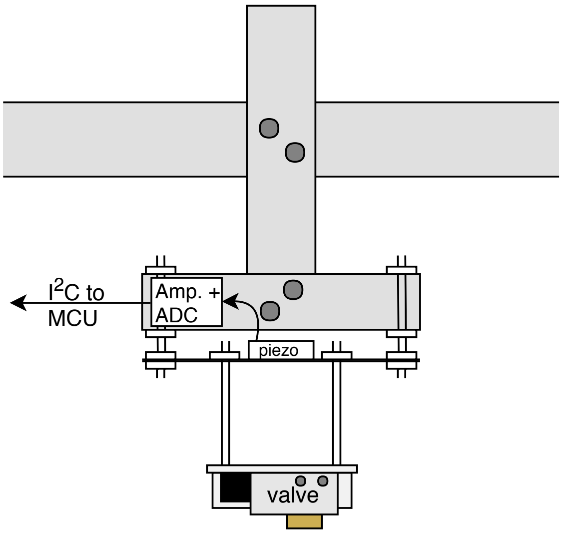


Figure 6.2: Schematic setup using a piezoelectric sensor attached to the construction as a reference

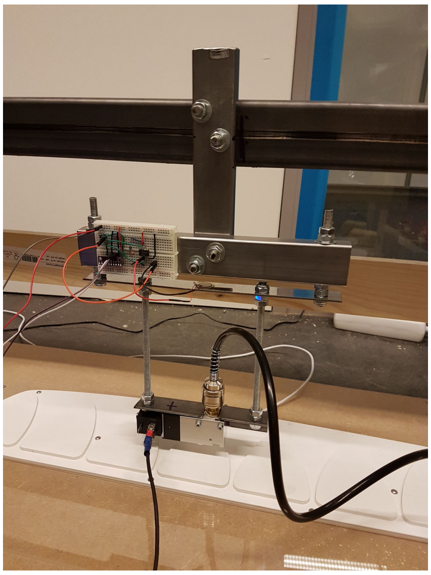


Figure 6.3: Steel setup with piezoelectric sensor as a reference attached to the construction

References

[1] J. Posnett and P. Frank, “The burden of chronic wounds in the uk,” *Nursing Times*, vol. 104, no. 3, pp. 44–45, 2008.

[2] R. Halfens, E. Meesterberends, J. Neyens, A. Rondas, S. Rijcken, S. Wolters, and J. Schols, “Landelijke prevalentiemeting zorgproblemen rapportage - resultaten 2015,” Tech. Rep., 2015. [Online]. Available: https://nl.lpz-um.eu/Content/Public/NL/Publications/LPZ%20Rapport%202015.pdf

[3] (2018) Momo medical solutions. [Online]. Available: https://momomedical.com/solutions/

[4] J. Tichy, J. Erhart, E. Kittinger, and J. Privratska, *Fundamentals of Piezoelectric Sensorics*, 1st ed. Springer, 2010.

[5] G. Gautschi, *Piezoelectric sensorics: force, strain, pressure, acceleration and acoustic emission sensors, materials and amplifiers*, 1st ed. Springer-Verlag, 2002.

[6] A. Jain, P. K. J., A. K. Sharma, A. Jain, and R. P.N, “Dielectric and piezoelectric properties of pvdf/pzt composites: A review,” *Polymer Engineering & Science*, vol. 55, no. 7, pp. 1589–1616, 2015. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/pen.24088

[7] *Murata 7BB-20-6L0 Data sheet*, Murata Manufacturing Co. Ltd. [Online]. Available: https://www.murata.com/en-us/api/pdfdownloadapi?cate=&partno=7BB-20-6L0

[8] N. Aupetit, “Signal conditioning for shock sensors,” *ST Application Note - AN4708*, 2015.

[9] *ADS101x Data sheet*, Texas Instruments. [Online]. Available: http://www.ti.com/lit/ds/symlink/ads1015.pdf

[10] H. van Vliet, *Software Engineering: Principles and Practice*, 3rd ed. Chichester, 2008.

[11] J. Fraden, *Handbook of Modern Sensors*, 5th ed. Springer, 2016.

[12] (2018) Flexiforce a301 sensor. [Online]. Available: https://www.tekscan.com/products-solutions/force-sensors/a301

[13] M. A. Somer. M. Nacy and I. A. Baqer, “Static and dynamic calibration for flexiforce sensor using a special purpose apparatus,” *Innovative Systems Design and Engineering*, vol. 4, no. 1, 2013.

[14] J. Fourmann, A. Coustou, H. Aubert, P. Pons, J. Luc, A. Lefrançois, M. Lavayssière, and A. Osmont, “Wireless pressure measurement in air blast using pvdf sensors,” *2016 IEEE SENSORS*, pp. 1–3, Oct 2016.

[15] S. Downes, A. Knott, and I. Robinson, “Towards a shock tube method for the dynamic calibration of pressure sensor,” *Philosophical transactions. Series A, Mathematical, physical, and engineering sciences*, vol. 372, Aug 2014.

[16] *DRV8825 Data sheet*, Texas Instruments. [Online]. Available: http://www.ti.com/lit/ds/symlink/drv8825.pdf

[17] (2018) Speed of sound in some common solids. [Online]. Available: https://www.engineeringtoolbox.com/sound-speed-solids-d_713.html

[18] *GAMMA compressor 6 liter*, 2018. [Online]. Available: https://www.gamma.nl/assortiment/gamma-compressor-6-liter/p/B534705

[19] *Solenoid valves VUVS/valve manifold VTUS*, Festo, April 2016. [Online]. Available: https://docs-emea.rs-online.com/webdocs/14f9/0900766b814f9ddb.pdf

[20] D. Elger, B. Williams, C. Crowe, and J. Roberson, *Engineering Fluid Mechanics - SI Version*, 10th ed. Wiley, 2010.

[21] F. White, *Fluid Mechanics*, 7th ed. McGraw-Hill, 2011.

[22] *TAL221 Data sheet*, HTC-Sensor. [Online]. Available: https://cdn.sparkfun.com/assets/9/9/a/f/3/TAL221.pdf

[23] O. T. (2008) Intro to signal processing - deconvolution. [Online]. Available: http://terpconnect.umd.edu/~toh/spectrum/Deconvolution.html

A

Load cell measurement data

F (Hz)	average	t1	%dev	t2	%dev	t3	%dev	t4	%dev	t5	%dev
0.5	697.7	699.1	0.2	696.3	0.2	688.6	1.3	705.8	1.2	698.8	0.2
1	686.5	685.1	0.2	682.4	0.6	679.8	1.0	695.6	1.3	689.5	0.4
2	678.3	677	0.2	672.3	0.9	674.7	0.5	685.3	1.0	682.1	0.6
3	447.4	445.6	0.4	442	1.2	443.3	0.9	455.1	1.7	450.9	0.8
4	669.9	670	0.0	663.8	0.9	668.9	0.1	675.9	0.9	670.9	0.1
5	512.5	511	0.3	508.4	0.8	516.4	0.8	514.5	0.4	512.2	0.1
6	442.5	442	0.1	444.2	0.4	435.4	1.6	443	0.1	447.7	1.2
7	735.1	729.6	0.8	733.9	0.2	741.9	0.9	727.1	1.1	743.1	1.1
8	656.2	661	0.7	651.1	0.8	657.9	0.3	653.4	0.4	657.4	0.2
9	495.6	495.5	0.0	491	0.9	503	1.5	489.8	1.2	498.5	0.6
10	480.1	470.6	2.0	479.5	0.1	477.8	0.5	488.3	1.7	484.5	0.9
11	455.7	453.8	0.4	456.5	0.2	456.2	0.1	455.4	0.1	456.4	0.2
12	411.6	412.5	0.2	412.7	0.3	406.1	1.3	414.4	0.7	412.2	0.2

Table A.1: Average FFT peak values for different frequencies. The measured values (t1, t2, ...) are shown together with the percent deviation between the average and the measured result.

B

Total system costs

Brand	Type	Type number	Unit price	Amount	Total price
Pneumatics					
Festo	Pressure regulator	LR-1/4-D-7-MINI	42,06	1	42,06
Festo	Valve	VUVS-LK25-M32C-AD-G14-1B2-S	50,22	1	50,22
Stanley	Air tube	Spiral hose 6x8mm , 5m	10,70	2	21,40
Stanley	Quick Connector (F) <> 1/4" (M)	UNI 1/4M	5,37	2	10,74
Stanley	Quick Connector (M) <> 1/4" (M)	1/4M	3,72	2	7,44
Optional					
Gamma	Air compressor	CP-6	81,82	1	81,82
Electronics					
Temna	Power supply	72-10500	138,72	1	138,72
NXP	Microcontroller	MBED NXP LPC1768	49,95	1	49,95
TI	ADC	EVAL BOARD ADS1015 12-BIT ADC	8,72	1	8,72
POWERPAX	Power supply	SW4309	15,72	1	15,72
Infinion	MOSFET	IRFZ44NPBF	0,70	1	0,70
Analog Devices	Instrumentation amplifier	AD620ANZ	9,55	1	9,55
Connectors	ODU	5P F	6,00	1	6,00
Case	Case	MB6W	18,03	1	18,03
Various	Wires	-	2,00	1	2,00
Various	Various passives (diodes/caps/resistors)	-	2,00	1	2,00
Construction					
Konsta	Board	MDF 122x61 12mm	4,29	1	4,29
Konsta	Beam	210x60x40mm	3,30	1	3,30
Konsta	Beam	210x44x18mm	2,70	1	2,70
Konsta	Beam	210x12x12mm	2,29	1	2,29
Gamma	Bolts	M10x120mm 4pcs	5,78	1	5,78
Gamma	Nuts	M10 4pcs	2,14	1	2,14
Gamma	Screws	Various	3,00	1	3,00
Laserbeest	PMMA	3mm clear	53,94	1	53,94
Control system					
MATLAB	License	2018b	2000,00	1	2000,00
HP	Computer	EliteBook 745 G3 P4T40EA	698,35	1	698,35
Total (ex. VAT)					3186,92

Table B.1: Total system cost (date of creation: 12-12-2018)

C

User Manual

C.1. Before Starting

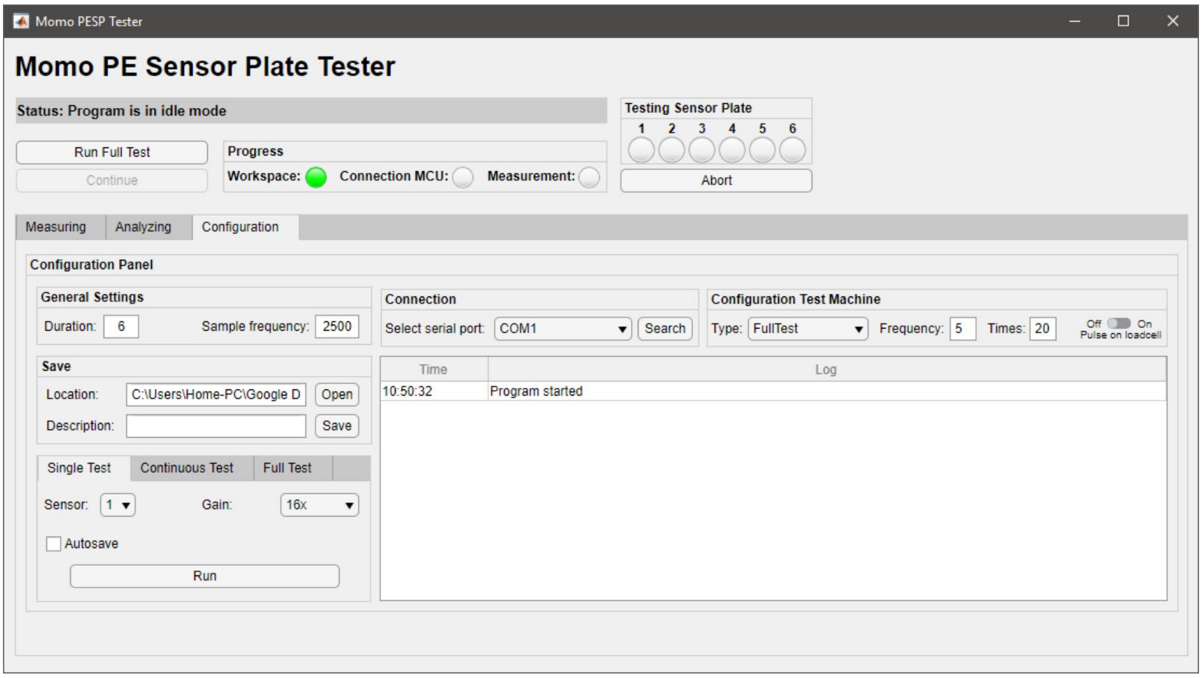
Ensure the following cables are plugged into the right position before attempting to start a measurement. The connector position are labeled on the case of the microcontroller:

- Sensor plate connection
- Load cell connector
- Valve connector
- Mini-USB connector
- 24V power connector
- -5V, +5V power connector

C.2. Quick Start Guide

- Start the program **Momo_PESP_Tester.mlapp**
- Make sure the workspace circle is green before continuing. In the case the circle remains red, refer to the troubleshooting page
- Make sure the valve is positioned on the load cell position, using the lower of the two holes in the mount
- Press the **Run Full Test** button
- Once the status bar states that the measurement is done for a sensor, the valve should be repositioned to the next sensor. Use the upper hole in the mount for the piezoelectric sensors.
- Press the continue button to continue the full test
- At the end of the full test, each pass/fail indicator will light up either green or red along with the scaling factor
- The measurement data and its scaling factor are automatically saved by the program

C.3. Program Overview


Momo PE Sensor Plate Tester

Status: Program is in idle mode

Workspace: ●

Duration: 6

Sample frequency: 2500

Select serial port: COM1

Type: FullTest

Frequency: 5

Times: 20

Location: C:\Users\Home-PC\Google D

Description:

Sensor: 1

Gain: 16x

☐ Autosave

Time	Log
10:50:32	Program started

Figure C.1: The configuration panel inside the MATLAB program

- Main window

  Three buttons:
  - **Run Full Test**: Run the full test
  - **Continue**: Continue the full test after the valve is put on the right position
  - **Abort**: Abort all actions and close the connection

  Progress indicators:
  - **Workspace**: If the default save location is found, the circle will be green, else it will be red
  - **Connection MCU**: If the connection is being made with the microcontroller, the circle is orange. If the connection is successfully made , the circle will be green. If the connection is failed, the circle will be red
  - **Measurement**: If the program is busy doing the data processing, the circle will be orange. If the data is successfully processed, the circle will be green
  - **Testing Sensor Plate**: After the full test, each circle (representing the piezoelectric sensors on the DUT), will be either green if the sensor passed the test, or red if the sensor failed the test
  - **Status bar**: The current status of the program is shown in the status bar
- Tabs
  - **Measuring**: There are three plots which can be used to look at the current measurement. A plot of the raw data, a plot of the filtered data and the frequency response of the raw data.
  - **Analyzing**: This tab is used to analyze a full test. The tab is automatically opened after a full test or a data set can be imported manually after which a plot is shown with all the measurements. The autoscale button can be pressed to automatically scale all the sensors. This can be undone by resetting the autoscale. Resetting the axis results in a zoomed out plot of the data, in case the user has zoomed into the measurement.
  - **Configuration Panel** This tab is used to modify the default settings. The duration of each measurement and the sampling frequency used can be changed in the general settings field. In the

case multiple UART connections are made, the right serial connection to the microcontroller must be chosen. The save location and the description can be set in the save field. Additional configuration to the test system is made by selecting the type of test, the valve frequency, amount of times the valve needs to open and close and the option to change the input signal to the loadcell to either a step or a pulse. The single or continues test can be run, its parameters can be changed in their respective window. The log window can be used to detect any errors or follow the progress of the program.

C.4. Troubleshoot

- **E101: No serial port found in the list**
  Open the configuration panel. If there is no serial port selected, try to remove the USB cable to the microcontroller and plug it into another USB port. Click on the **Search** button to look for new serial ports. In case no serial ports are found, consider installing the newest serial port drivers for your operating system

- **E102: Could not start the connection**
  Remove the USB cable to the microcontroller and close the program. Reconnect the USB cable and start the program

- **E103: Error with communication with the MCU**
  Remove the USB cable to the microcontroller and close the program. Reconnect the USB cable and start the program

- **E105: Error with saving data**
  Open the configuration panel. Change the save directory by clicking on the **Open** button found in the save field

- **E107: Error with reading data from the MCU**
  Remove the USB cable to the microcontroller and close the program. Reconnect the USB cable and start the program

If any of the problems persists or if other errors are showing up, please contact the developer of the software in order to solve the problem.

D

MATLAB code

D.1. Steel beam FFT plots

```
% Plot FFT's of 6 sensors using steel beam setup
close all;
Fs = 100;   % Sampling frequency

% Create new arrays containing only the 5Hz part
sensor_F_1_part = sensor_NF_1(1290:1572);
sensor_F_2_part = sensor_NF_2(1290:1572);
sensor_F_3_part = sensor_NF_3(1290:1572);
sensor_F_4_part = sensor_NF_4(1290:1572);
sensor_F_5_part = sensor_NF_5(1290:1572);
sensor_F_6_part = sensor_NF_6(1290:1572);

Np = numel(sensor_F_1_part);
zeropad = 2^(nextpow2(Np)); % Zero−pad to nearest power of 2

% Calculate FFT of all 6 sensors
fftF_1 = abs(fft(sensor_F_1_part, zeropad)/zeropad);
fftF_1 = fftF_1(1:zeropad/2);
fftF_1(2:end−1) = 2∗fftF_1(2:end−1);

fftF_2 = abs(fft(sensor_F_2_part, zeropad)/zeropad);
fftF_2 = fftF_2(1:zeropad/2);
fftF_2(2:end−1) = 2∗fftF_2(2:end−1);

fftF_3 = abs(fft(sensor_F_3_part, zeropad)/zeropad);
fftF_3 = fftF_3(1:zeropad/2);
fftF_3(2:end−1) = 2∗fftF_3(2:end−1);

fftF_4 = abs(fft(sensor_F_4_part, zeropad)/zeropad);
fftF_4 = fftF_4(1:zeropad/2);
fftF_4(2:end−1) = 2∗fftF_4(2:end−1);

fftF_5 = abs(fft(sensor_F_5_part, zeropad)/zeropad);
fftF_5 = fftF_5(1:zeropad/2);
fftF_5(2:end−1) = 2∗fftF_5(2:end−1);

```

```
fftF_6 = abs(fft(sensor_F_6_part, zeropad)/zeropad);
fftF_6 = fftF_6(1:zeropad/2);
fftF_6(2:end-1) = 2*fftF_6(2:end-1);

% Create frequency axis
f_axis = linspace(0, Fs/2, zeropad/2);

figure;
subplot(321)
plot(f_axis, 20*log10(fftF_1));
title('Sensor 1')
xlabel('Frequency [Hz]')
ylabel('FFT amplitude [dB]')
xlim([0 25])
grid minor
subplot(322)
plot(f_axis, 20*log10(fftF_2));
title('Sensor 2')
xlabel('Frequency [Hz]')
ylabel('FFT amplitude [dB]')
xlim([0 25])
grid minor
subplot(323)
plot(f_axis, 20*log10(fftF_3));
title('Sensor 3')
xlabel('Frequency [Hz]')
ylabel('FFT amplitude [dB]')
xlim([0 25])
grid minor
subplot(324)
plot(f_axis, 20*log10(fftF_4));
title('Sensor 4')
xlabel('Frequency [Hz]')
ylabel('FFT amplitude [dB]')
xlim([0 25])
grid minor
subplot(325)
plot(f_axis, 20*log10(fftF_5));
title('Sensor 5')
xlabel('Frequency [Hz]')
ylabel('FFT amplitude [dB]')
xlim([0 25])
grid minor
subplot(326)
plot(f_axis, 20*log10(fftF_6));
title('Sensor 6')
xlabel('Frequency [Hz]')
ylabel('FFT amplitude [dB]')
xlim([0 25])
grid minor
```

D.2. Curve fitting

```
%% Curve fitting

```

```
clear all;
close all;
clc;

load('deconvolutieunit12_50gram.mat');

deconv_raw = diff(sensor_NF);
% manualy set sample values with visible ringing
deconv_raw = -deconv_raw(1614:2200);

% manually determine f and tau from raw data plot
dt = 1/2500;
N = length(deconv_raw);
t = (0:N-1)*dt;
N1 = N;
tau = 0.05;
f = 99;

figure();
plot(t, deconv_raw)
hold on;

impulse_fit = 2000*sin(2*pi*f*t).*exp(-t/tau);
plot(t, impulse_fit);
grid minor
xlabel('Time (s)')
ylabel('ADC value')
title('Ringing curve-fit')
```

D.3. Dome on dome result

```
clear;
close all;
% Create arrays of measurements per type fixation
epoxy = [load('epoxy_1hz.mat', 'sensor_NF_1'); load('epoxy_2hz.mat',...
    'sensor_NF_1'); load('epoxy_4hz.mat', 'sensor_NF_1'); ...
    load('epoxy_8hz.mat', 'sensor_NF_1'); ...
    load('epoxy_16hz.mat', 'sensor_NF_1')];

lijm = [load('lijm_1hz.mat', 'sensor_NF_1'); load('lijm_2hz.mat',...
    'sensor_NF_1'); load('lijm_4hz.mat', 'sensor_NF_1'); ...
    load('lijm_8hz.mat', 'sensor_NF_1'); ...
    load('lijm_16hz.mat', 'sensor_NF_1')];

tape = [load('tape_1hz.mat', 'sensor_NF_1'); load('tape_2hz.mat',...
    'sensor_NF_1'); load('tape_4hz.mat', 'sensor_NF_1'); ...
    load('tape_8hz.mat', 'sensor_NF_1'); ...
    load('tape_16hz.mat', 'sensor_NF_1')];

% Array with fixation names
materials = [epoxy lijm tape];
mats_str = ["Epoxy" "Super glue" "Tape"];

% Test frequencies
freqs = [1 2 4 8 16];
```

```

x = linspace(0, 2.125, 255);
padding = 1000;
Fs = 120;

ticks = -25:5:25;
for i = 1:3
    for j = 1:5
        % Remove DC bias from data
        materials(j, i).dcbias = mean(materials(j, i).sensor_NF_1);

        % Zeropad data to obtain better FFT resolution
        sensor_padded = [materials(j, i).sensor_NF_1 - materials(j, i).dcbias; zeros(padding,1)];

        Np = numel(sensor_padded);
        if mod(Np, 2) == 1
            Np = Np+1;
        end
        % Take single sided FFT of measurements
        fft1 = abs(fft(sensor_padded)/Np);
        fft1 = fft1(1:Np/2);
        fft1(2:end-1) = 2*fft1(2:end-1);

        % Put FFTs in a single array
        materials(j, i).ffts = fft1;

    end
end

% Initialize array to hold subplot handle
h = zeros(15,1);

% Frequency axis
f = linspace(0, Fs/2, Np/2);

% Hard way to determine the order in the subplots
k = [1 4 7 10 13 2 5 8 11 14 3 6 9 12 15];
m = 1;
figure
for i = 1:3
    for j = 1:5
        % Create multiple subplots
        h(m) = subplot(5, 3, k(m));
        plot(f, materials(j, i).ffts)
        xlim([0 20])
        grid minor;
        title(['Material: ', num2str(mats_str(i)),'; Frequency: ', num2str(freqs(j)), ' Hz']);
        xlabel('Frequency (Hz)')
        ylabel('FFT magnitude')
        m = m + 1;
    end
end
```

D.4. FFT peak finder

```
clear all;
close all;
clc;
% Load measurement files
[filename,folder] = uigetfile('*.mat',...
        'Select One or More Files', ...
        'MultiSelect', 'on');

        filename = char(filename);
[row, col] = size(filename);

% Put all measurements in one matrix
data = [];
for n=1:row
    load([folder filename(n, :)]);
    data = [data, fourier_NF];
end

% Find peaks of measuments
pks = [];
for n=1:row
    [pks_tmp, locs_tmp] = findpeaks(data(:,n), 'MinPeakHeight', 300);
    pks = [pks, pks_tmp];

end

figure;
plot(pks)
ax = gca;
title('FFT peaks at 5 Hertz, 90 seconds apart, 3 hour load cell test')
ax.XGrid = 'on';
ax.YGrid = 'on';
ax.YMinorGrid = 'on';
xlabel('Measurements')
ylabel('FFT peak at 5 Hertz')
ylim([475 525]); xlim([0 130]); xticks([0:5:row])
hold on
% Find mean and standard deviation
y = mean(pks);
sd = std(pks);
upper_sd = y+sd; lower_sd = y-sd;
% Plot horizontal lines of mean and std. dev.
line([1, row],[y,y], 'Color', 'r')
line([1, row], [upper_sd, upper_sd], 'Color', 'g', 'LineWidth', 0.9);
line([1, row], [lower_sd, lower_sd], 'Color', 'g', 'LineWidth', 0.9);
legend('Measured FFT peaks', 'Mean', [char(177) '1 standard deviation'])
```

D.5. Sensor plate frequency response

```
close all;

% Test frequencies
freqs = [0.5 1 2 3 4 5 6 7 8 9 10 11 12];
```

```

% Sensor plate result corrected with the calculated ratios
d1 = sensors_freqs_d1 .* ratios;

figure;
hold on
set(gca, 'XScale', 'log');
xlabel('Frequency (Hz)')
ylabel('FFT value [dB]')
xlim([0.5 20])
% Plot result in dB on a logarithmic axis
for i = 1:6
    semilogx(freqs, 20*log10(d1(:,i)));
end
grid minor
legend('Sensor 1', 'Sensor 2', 'Sensor 3', 'Sensor 4', 'Sensor 5', 'Sensor 6', 'Location', 'SouthEast')
```

D.6. Deconvolution

```
%% Manual deconvolution

% load measurement
load('C:\Users\tlefe\Google Drive\BAP\Code en Metingen\Matlab\Metingen\Loadcell/1211_093728_Loadcell_fs2500_Sloadcell_G8x_f5_n20.mat')

% plot raw measurement
figure();
N = length(sensor_NF);
t = (0:N-1)*dt;
subplot(221);
plot(t, sensor_NF);
ylim([-750 2750])
grid minor
hold on;
xlabel('Time (s)')
ylabel('ADC value')
title('Raw load cell measurement')

% deconvolve via fft
impulse_fit = cos(2*pi*99*t).*exp(-t/0.05);
deconv = ifft(fft(sensor_NF) ./ fft(impulse_fit'));
N = length(deconv);
t = (0:N-1)*dt;
subplot(222);
plot(t, real(deconv));
ylim([-250 3500])
grid minor
xlabel('Time (s)')
ylabel('ADC value')
title('Load cell measurement deconvoluted')

% fft of deconvolved signal
dN = length(deconv);
f_res = 1/max(N*dt);
```

```matlab
f = (0:N-1) * f_res - 1250;
subplot(223);
plot(f, 20*log10(abs(fftshift(fft(deconv)))));
xlim([0 500]);
grid minor
xlabel('Frequency (Hz)')
ylabel('Amplitude (dB)')
title('FFT of deconvoluted signal')

% Filter deconvoluted signal
 fs = 2500;
 fc = [4, 18]; %from LTSpice simulation
 [B1,A1] = butter(2, fc(2)/(fs/2), 'low');
 [B2,A2] = butter(1, fc(1)/(fs/2), 'high');

 data_filtered = filter(B2,A2,(filter(B1,A1,deconv)));
subplot(224)
 plot(t,data_filtered, 'linewidth',1);
 grid minor
 title('Filtered with 2nd-order 20Hz LPF and 1st-order 4Hz HPF')
 xlabel('Time (s)')
 ylabel('ADC value')
```

E

Momo PE Sensor Plate Tester - MATLAB code

```matlab
%% Global properties to be used and to be part of the struct 'app'
properties (Access = public)
    % Properties for connection
    s                   % Serial Object
    nDatabits = 8;      % 2+(n*6) with n the amount of sensors reading simultaneously

    % Properties for storing measurement data
    tempData            % Temporary place to store data, which will be put in incomingData
    incomingData        % Data that is coming in from the serial USB port
    rawData             % Used for doing a full test to store each sensor measurement
    allMeasurements     % Used for doing a full test to store all sensor measurement
    sensor_NF           % Used to store all the data processed non filtered data
    sensor_F            % Used to store all the data processed filtered data
    fourier_NF          % Used to store all the FFT of the non filtered data
    x_scale             % Time axis for the data plots
    f_scale             % Frequency axis for Fourier plots

    % Properties to handle stops/continues between each function
    sensorReady         % If the nozzle is aligned with the dome
    stopContinuousTest  % Stops the continuous test
    emergencyStop       % Stop full test

    % Temporary struct to load the .mat into, so won't lose the last measurement
    temp

    % Properties to save all data from analyzing into seperate properties to avoid overwrite issues
    a_enableSensor = '1111111'    % On/Off for each sensor, default is all on
    a_incomingData
    a_description
    a_x_scale
    a_f_scale
```

```matlab
        a_scaleFactor
        a_sensor_NF
        a_sensor_F
        a_fourier_NF
        a_Fvalve
        a_Nvalve
        a_fs
        a_date
        a_folder
    end

    % Private Functions
    methods (Access = private)

        %%  Initializes the basic parameters for a connection:
        %   - buffersize based on the sample frequency and the duration
        %   - baudrate at 921600 to ensure a sample frequency of 2500 is possible
        function bufferSize = initializeConnection(app, Fs, duration)
            bufferSize = ceil(duration * Fs * app.nDatabits);

            app.s = serial(app.SelectserialportDropDown.Value);
            app.s.BaudRate = 921600;
            app.s.InputBuffersize = bufferSize*8;
        end

        %%  Initializes all basic properties and indicators
        function initializeVariables(app, bufferSize)
            app.stopContinuousTest = 0;
            app.sensorReady = 0;
            app.emergencyStop = 0;
            app.incomingData = [];
            app.rawData = zeros(bufferSize, 8);
            app.sensor_NF = [];
            app.sensor_F = [];
            app.fourier_NF = [];
            app.f_scale = [];
            app.x_scale = [];
            app.allMeasurements = [];
            app.a_scaleFactor = [1, 1, 1, 1, 1, 1];
            app.ConnectionMCULamp.Color = [1 0.65 0];
            LampArray = [app.Test_Lamp1, app.Test_Lamp2, app.Test_Lamp3, ...
                app.Test_Lamp4, app.Test_Lamp5, app.Test_Lamp6];
            for i = 1:6
                set(LampArray(i), 'Color', [0.94 0.94 0.94]);
            end
            app.MeasurementLamp.Color = [1 0.65 0];
        end

        %%  Determines the gain parameter based on user input [based on MCU]
        function gain = determineGain(app, gainValue)
            if(strcmp(gainValue, '16x'))
                gain = 0;
            elseif(strcmp(gainValue, '8x'))
                gain = 1;
            elseif(strcmp(gainValue, '4x'))
                gain = 2;
```

```
            elseif(strcmp(gainValue, '2x'))
                gain = 3;
            elseif(strcmp(gainValue, '1x'))
                gain = 4;
            else
                gain = 10; % If there were any errors with the user input
            end
        end

        %% Before starting a connection, all the necessary parameters/settings need to be valid
        %   Gain parameter to ensure a valid gain factor in the MCU
        %   A valid serial port connected before attempting to start a connection
        %   A valid save location in the case autosave is selected (full test and continuous are true by default)
        %   Abort measurement in the case the gain could not be determined, or if no serial port was selected
        function userConfigStatus = checkUserConfig(app, determinedGain, selectedAutoSave, selectedFolder)
            userConfigStatus = true;

            if(determinedGain == 10)
                updateLog(app, '[E100]|Measurement has been aborted: Gain could not be determined');
                app.StatusLabel.Text = 'Status: [E100]|Measurement has been aborted: Gain could not be determined';
                userConfigStatus = false;
                enableRunButtons(app);
            end

            if(isempty(app.SelectserialportDropDown.Value))
                updateLog(app, '[E101]|Measurement has been aborted: Serial port not found');
                app.StatusLabel.Text = 'Status: [E101]|Measurement has been aborted: Serial port not found';
                userConfigStatus = false;
                enableRunButtons(app);
            end

            if(selectedAutoSave == true && selectedFolder == "")
                updateLog(app, '[E105]|Error with save variables to the selected folder');
                app.StatusLabel.Text = 'Status: [E105]|Error with save variables to the selected folder';
                app.WorkspaceLamp.Color = 'red';
                userConfigStatus = false;
                enableRunButtons(app);
            end

            if(~exist(selectedFolder, 'dir'))
                updateLog(app, '[E106]|Measurement has been aborted: Could not find save location');
                app.StatusLabel.Text = 'Status: [E106]|Measurement has been aborted: Could not find save location';
                app.WorkspaceLamp.Color = 'red';
                userConfigStatus = false;
```

```matlab
                enableRunButtons(app);
            end
        end

        %% Tries to open the serial port
        function serialOpenStatus = openSerialPort(app)
            try
                fopen(app.s);
                serialOpenStatus = true;
                app.ConnectionMCULamp.Color = 'green';
            catch e
                updateLog(app, '[E102]|Measurement has been aborted: Could not connect to serial port, try again');
                updateLog(app, e.message);
                enableRunButtons(app);
                serialOpenStatus = false;
                instrreset;
                p = instrhwinfo('serial');
                app.SelectserialportDropDown.Items = p.AvailableSerialPorts;
                app.ConnectionMCULamp.Color = [1 0.65 0];
            end
        end

        %% Once the connection with the serial port is made, the input buffer is being emptied
        %% after which the parameters are send to the MCU and the MCU will start reading the ADC values
        %% the values are then send to MATLAB and stored
        function communicateMCU(app, bufferSize, fs, sensor, duration, gain, valveFrequency, valveTimes, loadcellPulse)
            app.MeasurementLamp.Color = [1 0.65 0];
            updateLog(app, ['MCU: Sensor: ', num2str(sensor), ', Duration: ', num2str(duration), ...
                ', Gain: ' num2str(2^(4-gain)) ' (', num2str(gain), '), Frequency: ', num2str(valveFrequency), ', Times: ', num2str(valveTimes)]);
            updateLog(app, ['MATLAB: inputData size is: ' num2str(bufferSize) ', buffer size: ' num2str(bufferSize*8) ' bytes, pulse: ' num2str(loadcellPulse)]);
            updateLog(app, 'Write variables:');

            % Flush input buffer before reading again
            while(app.s.BytesAvailable ~=0)
                bytesAvailable1 = app.s.BytesAvailable;
                flushinput(app.s);
                bytesAvailable2 = app.s.BytesAvailable;
                updateLog(app, [num2str(bytesAvailable1) 'bytes still in input buffer, buffer flush, still ' num2str(bytesAvailable2) ' in the input buffer']);
            end

            % Writing arguments
            data = [fs str2num(sensor) duration gain valveFrequency valveTimes loadcellPulse];
            fprintf(app.s, '%d %d %f %d %f %d %d\n', data, 'async');

```

```
            % Reading data (Current maxDataStream value is arbitrary chosen, based on max values of 2000/3000Hz)
            maxDataStream = 30000; % Amount of data the buffer needs to hold
            nMaxTimes = floor(bufferSize/maxDataStream);
            remainderBufferSize = bufferSize-nMaxTimes*maxDataStream;
            app.incomingData = [];

            updateLog(app, ['maxDataStream is: ' num2str(maxDataStream) ', ' num2str(ceil(bufferSize/maxDataStream)) 'x, remainder ' num2str(remainderBufferSize)])
            updateLog(app, 'Read variables:')

            for m = 1:nMaxTimes
                tempData = fread(app.s, maxDataStream);
                app.incomingData = [app.incomingData;tempData];
                updateLog(app, ['Amount of data in iteration ' num2str(m) ' is: ' num2str(numel(tempData))])
            end

            if(remainderBufferSize ~= 0)
                tempData = fread(app.s, remainderBufferSize);
                app.incomingData = [app.incomingData;tempData];
                updateLog(app, ['Amount of data in iteration ' num2str(m+1) ' is: ' num2str(numel(tempData))])
            end

            updateLog(app, 'Done with reading data MCU')
        end

        %% If an error occurs during any communication with the MCU, status information will be given
        function errorCommunication(app, e)
            disp(e.message);
            updateLog(app, '[E103]|Error with communication with the MCU');
            updateLog(app, e.message);
            app.StatusLabel.Text = 'Status: [E102]|Check log for more details';
            fclose(app.s);
            enableRunButtons(app);
            app.ConnectionMCULamp.Color = 'red';
        end

        %% Processes all the data [ASCII] to [NUM], applies a low pass filter and FFT
        function processData(app, measurementType, sampleFrequency, incomingData, iteration, Fvalve, Nvalve)
            app.rawData(:,iteration) = incomingData;
            % Each datastream starts with a 10 (defined newline '\n'), look for the first index of the data stream
            % Take second data stream as the first one still contains the value stored in the ADC register from previous time
            for startIndex = 1:10
                if incomingData(startIndex) == 10
                    break
                end
            end

```

```
            % Determine how many datastreams there are by dividing the amount of data in the array, by the amount of data per datastream
            % Two edge cases: datastream suddenly starts and datastream suddenly ends
            % Deal with these two cases by subtracting 2 datastreams (or use a floor and subtract 1)
            nDataRange = floor(numel(incomingData)/app.nDatabits)-1;
            sensor_NF = zeros(nDataRange,1);

            % ASCII to numbers, base of ASCII is 48 (ASCII 48 = NUM 0), check ASCII table for more information
            % Datastream always ends with |13|10| -> used as reference
            % Index 0: '10', Index 1: sign +/-, Index 2-6: data, Index 7: carrier return
            for dataStream = 1:(numel(sensor_NF))
                % First sensor
                sensor_NF(dataStream) = (incomingData(startIndex+2)-48)*10000+(incomingData(startIndex+3)-48)*1000+(incomingData(startIndex+4)-48)*100+(incomingData(startIndex+5)-48)*10+(incomingData(startIndex+6)-48)*1;
                if incomingData(startIndex+1) == '-'
                    sensor_NF(dataStream) = sensor_NF(dataStream) * -1;
                end

                %Go to the next data stream
                startIndex = startIndex + app.nDatabits;
            end

            % Creating the x-axis for the graph
            x_scale = linspace(0,(nDataRange/sampleFrequency),nDataRange);

            % Applying a filter to each sensor, with passband frequency 25Hz
            sensor_F = lowpass(sensor_NF,25,sampleFrequency);

            % Display the peak of the sample frequency in the FFT
            startSample = 1*sampleFrequency;
            endSample = min((Nvalve/Fvalve+1)*sampleFrequency, size(sensor_NF,1));

            % Applying FFT on the non filtered signal
            T = 1/sampleFrequency;    % Sampling period
            L = numel(sensor_NF(startSample:endSample));  % Length of signal

            % FFT function gives the double sides spectrum, convert it into single spectrum
            DSFourierData = fft(sensor_NF(startSample:endSample), 2^nextpow2(L)); % Double sided FFT
            DSFourierData = abs(DSFourierData/numel(DSFourierData)); %Normalized, positive
            fourier_NF = DSFourierData(1:numel(DSFourierData)/2+1); %Single sided FFT
            fourier_NF(2:end-1) = 2*fourier_NF(2:end-1);
            f_scale = sampleFrequency*(0:(numel(DSFourierData)/2))/numel(DSFourierData);

            % Determines the most dominant frequency and its peak, and the second most dominant frequency and its peak
            maxValue = zeros(2,1);
            indexMaxValue = maxValue;
```

```
            temp = fourier_NF;
            for i=1:2
                [maxValue(i) indexMaxValue(i)] = max(temp);
                temp(indexMaxValue(i)) = 0;
            end

            updateLog(app, ['Frequency with highest peak ' num2str(f_scale(
                indexMaxValue(1))) 'Hz, with a peak of ' ...
                num2str(maxValue(1))]);
            updateLog(app, ['Frequency with second highest peak ' num2str(f_scale(
                indexMaxValue(2))) 'Hz, with a peak of ' ...
                num2str(maxValue(2))]);

            % Assign variables based on type of measurement
            if(strcmp(measurementType, 'Single'))
                assignSingleVariablest(app, x_scale, sensor_NF, sensor_F, f_scale,
                    fourier_NF);
            elseif(strcmp(measurementType, 'Full'))
                assignMultipleVariables(app, iteration, x_scale, sensor_NF, sensor_F,
                    f_scale, fourier_NF);
            else
                updateLog(app, '[E104]|Could not save variables, error in assigning
                    variables');
            end

            % Plot the variables
            plotVariablesMeasurement(app, x_scale, sensor_NF, sensor_F, f_scale,
                fourier_NF)
            app.ConnectionMCULamp.Color = 'green';
        end

        %%  When a new .mat full test file is loaded, all sensor checkboxes are checked
        function enableSensorCheckboxes(app)
            app.Sensor1CheckBox.Enable = true;
            app.Sensor2CheckBox.Enable = true;
            app.Sensor3CheckBox.Enable = true;
            app.Sensor4CheckBox.Enable = true;
            app.Sensor5CheckBox.Enable = true;
            app.Sensor6CheckBox.Enable = true;
            app.Loadcell1CheckBox.Enable = true;
            app.Loadcell2CheckBox.Enable = true;
            app.GraphDropDown.Enable = true;
            app.PeriodsCheckBox.Enable = true;
        end

        %% Assigns variables from single and continuous test
        function assignSingleVariablest(app, x_scale, sensor_NF, sensor_F, f_scale,
            fourier_NF)
            % Put the variables in the struct so it can be used across functions
            app.x_scale = x_scale;
            app.sensor_NF = sensor_NF;
            app.sensor_F = sensor_F;
            app.f_scale = f_scale;
            app.fourier_NF = fourier_NF;
            app.allMeasurements = [app.allMeasurements, sensor_NF];

```

```
            % Assigning a variable for each property in the struct so it can be read out in the workspace
            assignin('base', 'incomingData', app.incomingData);
            assignin('base', 'allMeasurements', app.allMeasurements);
            assignin('base', 'x_scale', app.x_scale);
            assignin('base', 'sensor_NF', app.sensor_NF);
            assignin('base', 'sensor_F', app.sensor_F);
            assignin('base', 'fourier_NF', app.fourier_NF);
            assignin('base', 'f_scale', app.f_scale);
        end

        %% Assigns variables from full test
        function assignMultipleVariables(app, iteration, x_scale, sensor_NF, sensor_F, f_scale, fourier_NF)
            % Put the variables in the struct so it can be used across functions
            app.x_scale = x_scale;
            app.sensor_NF(:,iteration) = sensor_NF;
            app.sensor_F(:,iteration) = sensor_F;
            app.f_scale = f_scale;
            app.fourier_NF(:,iteration) = fourier_NF;

            % Assigning a variable for each property in the struct so it can be read out in the workspace
            assignin('base', 'incomingData', app.rawData);
            assignin('base', 'x_scale', app.x_scale);
            assignin('base', 'sensor_NF', app.sensor_NF);
            assignin('base', 'sensor_F', app.sensor_F);
            assignin('base', 'fourier_NF', app.fourier_NF);
            assignin('base', 'f_scale', app.f_scale);
        end

        %% Assigns variables from analyzing
        function assignAnalyzeVariables(app)
            % Put the variables in the struct so it can be used across functions
            assignin('base', 'a_x_scale', app.a_x_scale);
            assignin('base', 'a_f_scale', app.a_f_scale);
            assignin('base', 'a_sensor_NF', app.a_sensor_NF);
            assignin('base', 'a_sensor_F', app.a_sensor_F);
            assignin('base', 'a_fourier_NF', app.a_fourier_NF);
        end

        %% Plot the data from each measurement to the GUI
        function plotVariablesMeasurement(app, x_scale, sensor_NF, sensor_F, f_scale, fourier_NF)
            % Plotting all the graphs
            plot(app.UIAxes_NF,x_scale,sensor_NF)
            xlim(app.UIAxes_NF, 'auto')
            ylim(app.UIAxes_NF, 'auto')
            plot(app.UIAxes_F,x_scale,sensor_F)
            xlim(app.UIAxes_F, 'auto')
            ylim(app.UIAxes_F, 'auto')
            plot(app.UIAxes_FFT, f_scale, fourier_NF)
            xlim(app.UIAxes_FFT, [0 20])
            ylim(app.UIAxes_FFT, 'auto')
            app.MeasurementLamp.Color = 'green';
        end
```

```

        %% Determines the best gain based on the highest value from the last measurement
        function gain = determineBestGain(app, data)
            factor = 1.1;
            bits = ceil(log(max(data)*factor)/log(2));
            gain = bits - 11;
            if(gain<0)
                gain = 0;
            end
            updateLog(app, ['Highest peak has value ' num2str(max(data)) ', based on this and factor 1.1, the best gain is ' num2str(2^(4-gain))]);
        end

        %% Analyzes the full data, either after finishing the full test or when the user loads a full test .mat file
        function analyzeFullTestData(app, sampleFrequency, sensorNumber, gain, Fvalve, ...
                    Nvalve, folder, type, today)
            maxFFTValue = ones(6,1);

            % Process data to put into table
            for i = 1:6
                [maxFFTValue(i), indexMaxFFTValue] = max(app.a_fourier_NF(:,i));
                frequencyFFTMaxValue = app.a_f_scale(indexMaxFFTValue);
                if(i == 1)
                    app.UITable2.Data = [{sprintf('%0.4f', frequencyFFTMaxValue) ...
                        sprintf('%0.2f', maxFFTValue(i)) sprintf('%0.2f', mean(app.a_sensor_NF(:,i))) '1'}];
                else
                    app.UITable2.Data = [app.UITable2.Data;{sprintf('%0.4f', frequencyFFTMaxValue) ...
                        sprintf('%0.2f', maxFFTValue(i)) sprintf('%0.2f', mean(app.a_sensor_NF(:,i))) '1'}];
                end
            end

            % Assign data
            assignAnalyzeVariables(app);

            % Process data into graph
            updateAnalyzeAxes(app);

            % Peak value of the FFT normalised to the amount of times the valve opens
            normalizedFFTMaxValue = maxFFTValue/app.a_Nvalve;
            testSensorsIndicators(app, normalizedFFTMaxValue);

            app.TabGroup2.SelectedTab = app.AnalyzingTab;
            enableSensorCheckboxes(app);

            % If no scale factor has been determined, it will be determined now
            if(app.a_scaleFactor == [1, 1, 1, 1, 1, 1])
                updateLog(app, 'Scale factor will be determined now');
                maxFFTValue = ones(6,1);
                app.a_scaleFactor = ones(6,1);

```

```
        for i = 1:6
            [maxFFTValue(i), ~] = max(app.a_fourier_NF(:,i));
        end

        referenceValue = max(maxFFTValue);

        for i = 1:6
            app.a_scaleFactor(i) = referenceValue/maxFFTValue(i);
        end

        scaleFactor = app.a_scaleFactor;
        incomingData = app.a_incomingData;
        x_scale = app.a_x_scale;
        sensor_NF = app.a_sensor_NF;
        sensor_F = app.a_sensor_F;
        f_scale = app.a_f_scale;
        fourier_NF = app.a_fourier_NF;
        description = app.a_description;

        % Filename
        fileName = [today '_' type '_fs' num2str(sampleFrequency) '_S' sensorNumber '_G' gain '_f' num2str(Fvalve) '_n' num2str(Nvalve) '_cmp.mat'];

        % Foldername
        if ~exist(folder, 'dir')
            app.stopContinuousTest = 1;
            enableRunButtons(app);
            disp(folder);
            updateLog(app, '[E105]|Error with auto saving variables to the selected folder, no variables are saved');
            app.StatusLabel.Text = 'Status: [E105]|Error with auto saving variables to the selected folder, no variables are saved';
            return;
        end

        if(path ~= 0)
            save(strcat(folder, '/',fileName), 'description', 'incomingData', 'x_scale', 'sensor_NF', 'sensor_F', 'f_scale', 'fourier_NF', 'scaleFactor');
            updateLog(app, '− − − Measurement saved − − −');
        end
    end
end

%% Saves all the important variables to a .mat file
function autoSaveVariables(app, sampleFrequency, sensorNumber, gain, Fvalve, ...
        Nvalve, folder, type, today)
    % Assigning
    incomingData = app.incomingData;
    x_scale = app.x_scale;
    sensor_NF = app.sensor_NF;
    sensor_F = app.sensor_F;
    f_scale = app.f_scale;
    fourier_NF = app.fourier_NF;
```

```
            description = app.DescriptionEditField.Value;

            % Filename
            fileName = [today '_' type '_fs' num2str(sampleFrequency) '_S' sensorNumber '_G' gain '_f' num2str(Fvalve) '_n' num2str(Nvalve) '.mat'];

            % Foldername
            if ~exist(folder, 'dir')
                app.stopContinuousTest = 1;
                enableRunButtons(app);
                disp(folder);
                updateLog(app, '[E105]|Error with auto saving variables to the selected folder, no variables are saved');
                app.StatusLabel.Text = 'Status: [E105]|Error with auto saving variables to the selected folder, no variables are saved';
                return;
            end

            if(path ~= 0)
                save(strcat(folder, '/',fileName), 'description', 'incomingData', 'x_scale', 'sensor_NF', 'sensor_F', 'f_scale', 'fourier_NF');
                updateLog(app, '- - - Measurement saved - - -');
            end
        end

        %% Creates a plot when the continuous test is finished each time and saves the plot as .png
        function savePlotContinuous(app, sensorNumber, Fvalve, ...
                Nvalve, type, Fs, gain, folder, today)
            % Make a figure of the plots and upload it
            % Raw data
            plotFigure = figure('units','normalized','outerposition',[0 0 1 1]);
            subplot(1,2,1)
            plot(app.x_scale, app.sensor_NF);
            title(['Raw data: sensor ' sensorNumber ', valve Frequency ' num2str(Fvalve) ', times ' num2str(Nvalve)])
            xlabel('Time (s)')
            ylabel('ADC Output Value')
            grid on;
            grid minor;

            subplot(1,2,2)
            plot(app.f_scale, app.fourier_NF)
            title('FFT of non filtered data')
            xlabel('Frequency (Hz)')
            ylabel('|Y(f)|')
            xlim([0 50]);
            grid on;
            grid minor;

            % Save figure
            fileName = [today '_' type '_fs' num2str(Fs) '_S' sensorNumber '_G' gain '_f' num2str(Fvalve) '_n' num2str(Nvalve)];
            saveas(plotFigure, [strcat(folder,'/',fileName) '.png']);
            close(plotFigure);
        end
```

```matlab

        %% Diable the following buttons if a test is started
        function disableRunButtons(app)
            app.Single_Run.Enable = false;
            app.Full_Run.Enable = false;
            app.Continuous_Run.Enable = false;
            app.Continuous_Stop.Enable = false;
            app.Full_Continue.Enable = false;
        end

        %% Enable/disable the following buttons when a test is finished
        function enableRunButtons(app)
            app.Single_Run.Enable = true;
            app.Full_Run.Enable = true;
            app.Continuous_Run.Enable = true;
            app.Continuous_Stop.Enable = false;
            app.Full_Continue.Enable = false;
        end

        %% Updates the log window
        function updateLog(app, message)
            try
                app.UITable.Data = [{datestr(now, 'HH:MM:SS') message}; app.UITable.Data;];
            catch e
                disp(e.message);
            end
        end

        % Updates the analyze axes depending on various settings
        function updateAnalyzeAxes(app)
            checkBoxes = [app.Sensor1CheckBox.Value app.Sensor2CheckBox.Value app.Sensor3CheckBox.Value app.Sensor4CheckBox.Value ...
                app.Sensor5CheckBox.Value app.Sensor6CheckBox.Value app.Loadcell1CheckBox.Value app.Loadcell2CheckBox.Value app.PeriodsCheckBox.Value];

            TableData = app.UITable2.Data;
            scaleFactor = ones(8,1);
            for i = 1:6
                scaleFactor(i) = str2double(TableData(i,4));
            end

            if (app.GraphDropDown.Value == "Raw")
                data = app.a_sensor_NF;
                xaxis = app.a_x_scale;
                xlim(app.UIAxes, 'auto')
                title(app.UIAxes, strcat(app.a_date, " | fs: ", num2str(app.a_fs), "Hz | Valve freq.: ", num2str(app.a_Fvalve), ...
                    "Hz | Valve times open: ", num2str(app.a_Nvalve), "x | Raw data"), 'Interpreter', 'none')
                maxLine = max(app.a_sensor_NF(:));
                xlabel(app.UIAxes, 'Time (s)')
                ylabel(app.UIAxes, 'ADC Output Value');
            elseif(app.GraphDropDown.Value == "Filtered");
                data = app.a_sensor_F;
```

```
                xaxis = app.a_x_scale;
                xlim(app.UIAxes, 'auto')
                title(app.UIAxes, strcat(app.a_date, " | fs: ", num2str(app.a_fs), "Hz | Valve freq.: ", num2str(app.a_Fvalve), ...
                    "Hz | Valve times open: ", num2str(app.a_Nvalve), "x | Filtered data"), 'Interpreter', 'none')
                maxLine = max(app.a_sensor_F(:));
                xlabel(app.UIAxes, 'Time (s)')
                ylabel(app.UIAxes, 'ADC Output Value');
            elseif(app.GraphDropDown.Value == "FFT");
                data = app.a_fourier_NF;
                xaxis = app.a_f_scale;
                xlim(app.UIAxes, [0 20])
                title(app.UIAxes, strcat(app.a_date, " | fs: ", num2str(app.a_fs), "Hz | Valve freq.: ", num2str(app.a_Fvalve), ...
                    "Hz | Valve times open: ", num2str(app.a_Nvalve), "x | FFT raw data"), 'Interpreter', 'none')
                maxLine = 0;
                xlabel(app.UIAxes, 'Frequency (Hz)');
                ylabel(app.UIAxes, '|Y(f)|');
            end

            cla(app.UIAxes);
            for i = 1:9
                if(checkBoxes(i) == 1)
                    if(i == 9)
                        peakPeriod = 1/app.a_Fvalve;
                        line(app.UIAxes, [1 1],[maxLine*1.1 -maxLine*1.1], 'Color','black');
                        hold(app.UIAxes, 'on');
                        for k = 1:app.a_Nvalve
                            try
                                line(app.UIAxes, [1+peakPeriod*k 1+peakPeriod*k],[maxLine*1.1 -maxLine*1.1], 'Color','black');
                            catch e
                                disp(e.message);
                            end
                        end
                    else
                        plot(app.UIAxes, xaxis, data(:,i)*scaleFactor(i));
                        hold(app.UIAxes, 'on');
                    end
                end
            end
            hold(app.UIAxes, 'off');
        end

        %%  Calculate the mean from 3 intervals, when it is 0, when it is 1 (first part and second part)
        %%   Difference between 0 and 1 must be substantial, and first part/second part should result in same mean
        function statusPass = verifyMeasurementLC(app, sampleFrequency, duration, rawData)
            xStart = 0; % [sec]
            xSwitch = 1; % [sec] switch from off to on
            xEnd = duration; % [sec]
```

```
            xMid = (xEnd-xSwitch)/2+xSwitch; % [sec]
            varRange = 0.1; % [sec]

            index1 = ceil((xStart+varRange)*sampleFrequency);   % Start of the measurement
            index2 = ceil((xSwitch-varRange)*sampleFrequency);  % Time the valve switches, min delta time
            index3 = ceil((xSwitch+varRange)*sampleFrequency);  % Time the valve switches, plus delta time
            index4 = ceil((xMid)*sampleFrequency);              % Mid period when the valve is on
            index5 = ceil((xEnd-varRange)*sampleFrequency);     % End of the measurement
            updateLog(app, ['Index1:5 : ' num2str(index1) ' ' num2str(index2) ' ' num2str(index3) ' ' num2str(index4) ' ' num2str(index5)]);

            meanT1 = mean(rawData(index1:index2));
            meanT2 = mean(rawData(index3:index4));
            meanT3 = mean(rawData(index4:index5));
            updateLog(app, ['Means: ' num2str(meanT1) ' ' num2str(meanT2) ' ' num2str(meanT3)]);

            conditionLimit1 = 100; % Difference between meanT2 and meanT3
            conditionLimit2 = 100; % Difference between meanT1 and av(meanT2,meanT3)

            if(abs(meanT2-meanT3)<conditionLimit1 && abs(meanT2-meanT1)>conditionLimit2)
                statusPass = true;
                updateLog(app, 'Both conditions are met, measurement will go on');
            else
                statusPass = false;
                updateLog(app, 'One of the conditions is not met, measurement will be redone');
            end
        end

        %%  Look if the FFT is 'good' enough, else redo the measurement
        %%  The dominant frequency should be close to the valve frequency
        function statusPass = verifyMeasurementPE(app, valveFrequency, f_scale, rawDataFFT)
            [maxFFTValue indexMaxFFTValue] = max(rawDataFFT);
            frequencyFFTMaxValue = f_scale(indexMaxFFTValue);
            varRange = 0.1; % [Hz]

            %   If measurement is wrong, redo the measurement, else save the data and go on
            if(abs(frequencyFFTMaxValue-valveFrequency)<varRange)
                statusPass = true;
                updateLog(app, 'Condition is met, measurement will go on');
            else
                statusPass = false;
                updateLog(app, 'Condition not met, measurement will be redone');
            end
        end

        %%  Color the sensor indicators either red or green
```

```
        function testSensorsIndicators(app, normalizedMaxFFTValue)
            LampArray = [app.Test_Lamp1, app.Test_Lamp2, app.Test_Lamp3, ...
                app.Test_Lamp4, app.Test_Lamp5, app.Test_Lamp6];
            minimumValueFFT = 40;
            for i = 1:6
                if(normalizedMaxFFTValue(i) > minimumValueFFT)
                    set(LampArray(i), 'Color', 'green');
                else
                    set(LampArray(i), 'Color', 'red');
                end
            end
        end
    end


    methods (Access = private)

        % Code that executes after component creation
        function startupFcn(app)
            %% This callback is executed at the program startup
            %% Disconnect and delete all instrument objects, looks for all available serial ports
            %% Determines if the workspace exists

            % Clean up the command window, workspace is not cleared to prevent any loss of data
            clc;

            % Disconnect and delete all instrument objects
            instrreset;

            % Status updates
            app.UITable.Data = [{datestr(now, 'HH:MM:SS') 'Program started'}];
            app.StatusLabel.Text = 'Status: Program is in idle mode';

            % Initializations
            % Instrument Control Toolbox and serial port drivers required
            warning off MATLAB:subscripting:noSubscriptsSpecified
            p = instrhwinfo('serial');
            app.SelectserialportDropDown.Items = p.AvailableSerialPorts;

            % Checks if the default save location is present, default save location is the 'Metingen' folder
            folder = ['Metingen'];
            if ~exist(folder, 'dir')
                updateLog(app, 'Default save location is not found, consider changing your workspace');
                app.WorkspaceLamp.Color = 'red';
            else
                app.LocationEditField.Value = [pwd '\' folder];
                app.WorkspaceLamp.Color = 'green';
            end
        end

        % Button pushed function: Single_Run
        function Single_RunPushed(app, event)
```

```
        %% This callback is executed when the user presses on the Single run button
        %% Does one test with the parameters in the configuration panel

        % Clean up the command window
        clc;

        % User configuration parameters
        selectedSensor = app.Single_Sensor.Value;
        selectedDuration = app.General_Duration.Value;
        selectedGain = app.Single_Gain.Value;
        selectedFs = app.General_SampleFrequency.Value;
        selectedFvalve = app.ValveFrequency.Value;
        selectedNvalve = app.ValveTimes.Value;
        selectedAutoSave = app.Single_AutoSave.Value;
        selectedType = app.TypeDropDown.Value;
        selectedFolder = app.LocationEditField.Value;
        if(app.PulseonloadcellSwitch.Value == "On")
            selectedLoadcellPulse = 1;
        else
            selectedLoadcellPulse = 0;
        end

        % Standard initialisations
        updateLog(app, '- - - Running single measurement now - - -');
        app.StatusLabel.Text = 'Status: Running single measurement now';
        disableRunButtons(app);
        determinedBufferSize = initializeConnection(app, selectedFs, selectedDuration);
        initializeVariables(app, determinedBufferSize);
        determinedGain = determineGain(app, selectedGain);
        connectionStatus = openSerialPort(app);
        today = datestr(now, 'mmdd_HHMMSS');

        % Checks if all the configuration parameters are correct and if the connection with the MCU could be made
        if(checkUserConfig(app, determinedGain, selectedAutoSave, selectedFolder) == false ...
                || connectionStatus == false)
            return;
        end

        % First part of the code is about the communication with the MCU and data transfer
        % Asynchronous writing arguments, synchronous reading data
        try
            communicateMCU(app, determinedBufferSize, selectedFs, selectedSensor, selectedDuration, ...
                determinedGain, selectedFvalve, selectedNvalve, selectedLoadcellPulse);
            fclose(app.s);
        catch e
            errorCommunication(app, e);
            return;
        end

        % Second part of the code is about processing the data
```

```
            % First checks if data was sent to the program before processing it
            if(size(app.incomingData) == 0)
                updateLog(app, '[E107]|Error receiving data from the MCU, received no data');
                app.StatusLabel.Text = 'Status: [E107]|Error receiving data from the MCU, received no data';
                enableRunButtons(app);
                app.MeasurementLamp.Color = 'red';
                return;
            end

            % Processing the raw data [ASCII] into values [NUM]
            processData(app, 'Single', selectedFs, app.incomingData, 1, selectedFvalve, selectedNvalve);

            % Autosave if selected, check if the directory is right
            if(selectedAutoSave == true)
                try
                    autoSaveVariables(app, selectedFs, selectedSensor, selectedGain, selectedFvalve, ...
                        selectedNvalve, selectedFolder, selectedType, today);
                catch e
                    updateLog(app, e.message);
                    enableRunButtons(app);
                    app.WorkspaceLamp.Color = 'red';
                    return;
                end
            end

            % End of measurement
            enableRunButtons(app);
            updateLog(app, '– – – Single test has finished – – –');
            app.StatusLabel.Text = 'Status: Single test has finished';
        end

        % Button pushed function: Full_Run
        function Full_RunPushed(app, event)
            %% This callback is executed when the user presses on the Full run button
            %% Does a test on all seven sensors (1LC, PE) with the parameters in the configuration panel

            % Clean up the command window
            clc;

            % User configuration parameters
            selectedDuration = app.General_Duration.Value;
            selectedFs = app.General_SampleFrequency.Value;
            selectedFvalve = app.ValveFrequency.Value;
            selectedNvalve = app.ValveTimes.Value;
            selectedType = app.TypeDropDown.Value;
            selectedFolder = app.LocationEditField.Value;
            selectedLoadcellPulse = 0;

            % Standard initialisations
            updateLog(app, '– – – Running full measurement now – – –');
            app.StatusLabel.Text = 'Status: Running full measurement now';
```

```matlab
            disableRunButtons(app);
            determinedBufferSize = initializeConnection(app, selectedFs, selectedDuration);
            initializeVariables(app, determinedBufferSize);
            connectionStatus = openSerialPort(app);
            today = datestr(now, 'mmdd_HHMMSS');

            % Checks if all the configuration parameters are correct and if the connection with the MCU could be made
            if(checkUserConfig(app, 1, true, selectedFolder) == false ...
                    || connectionStatus == false)
                return;
            end

            % First part of the code is about the communication with the MCU and data transfer
            % Predefined arrays to quickly determine settings for each iteration
            sensors = [app.Full_Loadcell.Value app.Full_Sensor1.Value app.Full_Sensor2.Value app.Full_Sensor3.Value ...
                app.Full_Sensor4.Value app.Full_Sensor5.Value app.Full_Sensor6.Value app.Full_Loadcell.Value];
            orderSensor = ['7', '1', '2', '3', '4', '5', '6', '7'];
            orderPlots = [7, 1, 2, 3, 4, 5, 6, 8];

            % Asynchronous writing arguments, synchronous reading data
            % Between each measurement, the sensor values are being processed and saved in the local workspace
            try
                % Goes through 8 possible iterations
                for i = 1:8
                    statusPass = false;
                    determinedGain = 4; % Start with a gain of 1, then look for the appropriate gain

                    % If the sensor is selected, it will go through this if statement
                    if (sensors(i) == true)
                        app.Full_Continue.Enable = true;
                        if(orderSensor(i) == '7')
                            updateLog(app, ['Press continue to start measuring the loadcell']);
                        else
                            updateLog(app, ['Press continue to start measuring sensor ' orderSensor(i)]);
                        end

                        % Waiting for the continue button to be pressed
                        % In the future the button will be replaced by a signal that the nozzle is on top of the dome
                        while(app.sensorReady ~= 1)
                            pause(1);
                            if(app.emergencyStop == 1)
                                enableRunButtons(app);
                                fclose(app.s);
                                updateLog(app, '– – – Full test has been aborted – – –');
                                return;
```

```
                end
            end

            % Start measuring, will continue to do so until all requirements have been met
            while(statusPass ~= true && app.emergencyStop ~= 1)
                % Information about the arguments and data
                communicateMCU(app, determinedBufferSize, selectedFs, orderSensor(i), selectedDuration, determinedGain, selectedFvalve, selectedNvalve, selectedLoadcellPulse);

                % Data processing of the incoming information
                if(size(app.incomingData) == 0)
                    updateLog(app, '[E107]|Error receiving data from the MCU, received no data');
                    app.StatusLabel.Text = 'Status: [E107]|Error receiving data from the MCU, received no data';
                    enableRunButtons(app);
                    fclose(app.s);
                    app.MeasurementLamp.Color = 'red';
                    return;
                end

                processData(app, 'Full', selectedFs, app.incomingData, orderPlots(i), selectedFvalve, selectedNvalve);

                % Check if the data received has a good shape and is not distorted, loadcell: check step function, PE: check FFT
                if(orderSensor(i) == '7')
                    statusPass = verifyMeasurementLC(app, selectedFs, selectedDuration, app.sensor_NF(:,orderPlots(i)));
                else
                    statusPass = verifyMeasurementPE(app, selectedFvalve, app.f_scale, app.fourier_NF(:,orderPlots(i)));
                end

                % Check if the selected gain is too high/too low, adjust if needed
                bestGain = determineBestGain(app, app.sensor_NF(:, orderPlots(i)));
                if(determinedGain == bestGain && statusPass == true)
                    % Done with reading, waiting for a signal to read the next sensor
                    app.sensorReady = 0;
                    if(orderSensor(i) == '7')
                        updateLog(app, ['Done reading loadcell']);
                    else
                        updateLog(app, ['Done reading sensor ' orderSensor(i)]);
                    end
                else
                    statusPass = false;
                    determinedGain = bestGain;
                    updateLog(app, ['Redoing the measurement, trying now with gain: ' num2str(2^(4-determinedGain))]);
                end
```

```matlab

                            % To make it interruptable to make a emergency stop
                            pause(1);
                            if(app.emergencyStop == 1)
                                enableRunButtons(app);
                                fclose(app.s);
                                updateLog(app, '- - - Full test has been aborted - - -');
                                return;
                            end
                        end
                    end
                end

                % Information about stopping the connection
                fclose(app.s);
            catch e
                errorCommunication(app, e);
                return;
            end

            % End of measurement
            enableRunButtons(app);
            updateLog(app, '- - - Full test has finished - - -');
            app.StatusLabel.Text = 'Status: Full test has finished';

            % Switches to the analyze tab and shows all the measurements
            % Saves all current variables as different ones to avoid overwriting
            % Uses these variables to do analyzing
            app.a_description = app.DescriptionEditField.Value;
            app.a_incomingData = app.incomingData;
            app.a_x_scale = app.x_scale;
            app.a_f_scale = app.f_scale;
            app.a_sensor_NF = app.sensor_NF;
            app.a_sensor_F = app.sensor_F;
            app.a_fourier_NF = app.fourier_NF;
            app.a_date = today;
            app.a_folder = selectedFolder;
            app.a_fs = selectedFs;
            app.a_Fvalve = selectedFvalve;
            app.a_Nvalve = selectedNvalve;
            app.a_scaleFactor = [1,1,1,1,1,1];
            assignin('base', 'scaleFactor', app.a_scaleFactor);

            % After the measurement and data processing is done, the data is being analyzed by the program
            analyzeFullTestData(app, selectedFs, '1234567', '1x', selectedFvalve, ...
                        selectedNvalve, selectedFolder, selectedType, today);
        end

        % Button pushed function: Full_Continue
        function Full_ContinuePushed(app, event)
            %% This emulates a ready signal when nozzle is on top of the dome
            app.sensorReady = 1;
            app.Full_Continue.Enable = false;
        end
```

```

        % Button pushed function: Continuous_Run
        function Continuous_RunPushed(app, event)
            %% This callback is executed when the user presses on the Continuous run button
            %% Keeps doing a test with the parameters in the configuration panel

            % Clean up the command window
            clc;

            % User configuration parameters
            selectedSensor = app.Continuous_Sensor.Value;
            selectedDuration = app.General_Duration.Value;
            selectedInterval = app.Continuous_Interval.Value;
            selectedGain = app.Continuous_Gain.Value;
            selectedFs = app.General_SampleFrequency.Value;
            selectedFvalve = app.ValveFrequency.Value;
            selectedNvalve = app.ValveTimes.Value;
            selectedType = app.TypeDropDown.Value;
            selectedFolder = app.LocationEditField.Value;
            if(app.PulseonloadcellSwitch.Value == "On")
                selectedLoadcellPulse = 1;
            else
                selectedLoadcellPulse = 0;
            end

            % Standard initialisations
            updateLog(app, '− − − Running continuous measurement now − − −');
            app.StatusLabel.Text = 'Status: Running continuous measurement now';
            disableRunButtons(app);
            app.Continuous_Stop.Enable = true;
            determinedBufferSize = initializeConnection(app, selectedFs, selectedDuration);
            initializeVariables(app, determinedBufferSize);
            determinedGain = determineGain(app, selectedGain);
            connectionStatus = openSerialPort(app);
            runtimeAmount = 0;

            % Checks if all the configuration parameters are correct and if the connection with the MCU could be made
            if(checkUserConfig(app, determinedGain, true, selectedFolder) == false ...
                    || connectionStatus == false)
                return;
            end

            % First part of the code is about the communication with the MCU and data transfer
            % Asynchronous writing arguments, synchronous reading data
            while(app.stopContinuousTest ~= 1)
                % Initialisations for each measurement
                initializeVariables(app, determinedBufferSize);
                today = datestr(now, 'mmdd_HHMMSS');
                runtimeAmount = runtimeAmount + 1;
                updateLog(app, ['− − − Continuous measurement ' num2str(runtimeAmount) ' − − −']);

```

```
        try
            communicateMCU(app, determinedBufferSize, selectedFs, selectedSensor, selectedDuration, ...
                determinedGain, selectedFvalve, selectedNvalve, selectedLoadcellPulse);

            % Second part of the code is about processing the data
            % First checks if data was sent to the program before processing it
            if(size(app.incomingData) == 0)
                updateLog(app, '[E107]|Error receiving data from the MCU, received no data');
                app.StatusLabel.Text = 'Status: [E107]|Error receiving data from the MCU, received no data';
                enableRunButtons(app);
                fclose(app.s);
                app.MeasurementLamp.Color = 'red';
                return;
            end

            % Processing the raw data [ASCII] into values [NUM]
            processData(app, 'Single', selectedFs, app.incomingData, 1, selectedFvalve, selectedNvalve);

            % Autosave if selected, check if the directory is right
            autoSaveVariables(app, selectedFs, selectedSensor, selectedGain, selectedFvalve, ...
                selectedNvalve, selectedFolder, selectedType, today);
            savePlotContinuous(app, selectedSensor, selectedFvalve, ...
                selectedNvalve, selectedType, selectedFs, selectedGain, selectedFolder, today)

            % Wait for the given interval, unless the stop button is pressed
            logHistory = app.UITable.Data;
            app.UITable.Data = [{datestr(now, 'HH:MM:SS') ['- - - Waiting for ' num2str(selectedInterval) ' seconds  - - -']};app.UITable.Data];

            for i = 1:ceil(selectedInterval)
                if(app.stopContinuousTest~=1)
                    pause(1);
                    app.UITable.Data = [{datestr(now, 'HH:MM:SS') ['- - - Waiting for ' num2str(selectedInterval-i) ' seconds  - - -']};logHistory];
                else
                    break;
                end
            end
        catch e
            errorCommunication(app, e);
            instrreset;
            initializeConnection(app, selectedFs, selectedDuration);
            app.UITable.Data = [{datestr(now, 'HH:MM:SS') '- - - Attempting to restart - - -'}; app.UITable.Data];
            fopen(app.s);
            disableRunButtons(app);
            app.Continuous_Stop.Enable = true;
```

```
                end
            end

            % End of measurement
            fclose(app.s);
            enableRunButtons(app);
            updateLog(app, '- - - Continuous test has finished - - -');
            app.StatusLabel.Text = 'Status: Continuous test has finished';
        end

        % Button pushed function: Continuous_Stop
        function Continuous_StopPushed(app, event)
            %% When the user presses on the stop button, the continuous test will stop as soon as possible
            app.stopContinuousTest = 1;
            disableRunButtons(app);
            updateLog(app, 'Stop button is pressed, measuring will stop soon');
        end

        % Button pushed function: OpendataButton
        function OpendataButtonPushed(app, event)
            %% This callback is executed when the user presses on the Open data button
            %% This is used to manually analyze full tests by opening the .mat file

            % Clean up the command window
            clc;

            % The user needs to select a .mat file
            app.MomoPESPTesterUIFigure.Visible = 'off';
            selectedFolder = app.LocationEditField.Value;
            if(selectedFolder == "")
                selectedFolder = pwd;
            end
            [filename,folder] = uigetfile([selectedFolder '\*.mat'],...
                'Select a full test');
            app.MomoPESPTesterUIFigure.Visible = 'on';

            % The program tries to read the .mat file and extracts all the necesssary information
            filename = char(filename);
            try
                app.temp = load([folder filename(1,:)]);
                [~, column] = size(app.temp.sensor_NF);
                if(column < 7)
                    updateLog(app, 'Error opening file, possibly not a full test');
                    return;
                end

                dateIndex1 = 1;
                dateIndex2 = 11;
                [fsIndex1, fsIndex2] = regexpi(filename(1,:),'_fs.*_S');
                [valveFrequencyIndex1, valveFrequencyIndex2] = regexpi(filename(1,:), 'x_f.*_n');
                date = filename(1, dateIndex1:dateIndex2);
                if(contains(filename, 'cmp') == 1)
                    [timesIndex1, timesIndex2] = regexpi(filename(1,:), '_n.*_cmp');
```

```
                else
                    [timesIndex1, timesIndex2] = regexpi(filename(1,:), '_n.*.mat');
                end

                app.a_incomingData = app.temp.incomingData;
                app.a_x_scale = app.temp.x_scale;
                app.a_f_scale = app.temp.f_scale;
                app.a_sensor_NF = app.temp.sensor_NF;
                app.a_sensor_F = app.temp.sensor_F;
                app.a_fourier_NF = app.temp.fourier_NF;
                app.a_date = date;
                app.a_folder = folder;
                app.a_fs = str2num(filename(1, fsIndex1+3:fsIndex2-2));
                app.a_Fvalve = str2num(filename(1, valveFrequencyIndex1+3:
                    valveFrequencyIndex2-2));
                app.a_Nvalve = str2num(filename(1, timesIndex1+2:timesIndex2-4));

                if(isfield(app.temp, 'scaleFactor') == 1)
                    app.a_scaleFactor = app.temp.scaleFactor;
                    updateLog(app, 'Scale factor found, it won''t be recalculated');
                else
                    app.a_scaleFactor = [1,1,1,1,1,1];
                    updateLog(app, 'Scale factor not found, all scale factors are
                        determined now');
                end

                if(exist('app.temp.description') == 1)
                    app.a_description = app.temp.description;
                else
                    app.a_description = '';
                end

                analyzeFullTestData(app, app.a_fs, '1234567', '1x', app.a_Fvalve, ...
                    app.a_Nvalve, folder, 'FullTest', date)
            catch e
                disp(e.message)
            end
        end

        % Value changed function: Sensor1CheckBox
        function Sensor1CheckBoxValueChanged(app, event)
            %% If the user unchecked/checked a sensor button in the analyzing tab,
            %% a new plot is made with the selected sensors
            updateAnalyzeAxes(app)
        end

        % Value changed function: Sensor2CheckBox
        function Sensor2CheckBoxValueChanged(app, event)
            %% If the user unchecked/checked a sensor button in the analyzing tab,
            %% a new plot is made with the selected sensors
            updateAnalyzeAxes(app)
        end

        % Value changed function: Sensor3CheckBox
        function Sensor3CheckBoxValueChanged(app, event)
            %% If the user unchecked/checked a sensor button in the analyzing tab,
```

```
            %% a new plot is made with the selected sensors
            updateAnalyzeAxes(app)
        end

        % Value changed function: Sensor4CheckBox
        function Sensor4CheckBoxValueChanged(app, event)
            %% If the user unchecked/checked a sensor button in the analyzing tab,
            %% a new plot is made with the selected sensors
            updateAnalyzeAxes(app)
        end

        % Value changed function: Sensor5CheckBox
        function Sensor5CheckBoxValueChanged(app, event)
            %% If the user unchecked/checked a sensor button in the analyzing tab,
            %% a new plot is made with the selected sensors
            updateAnalyzeAxes(app)
        end

        % Value changed function: Sensor6CheckBox
        function Sensor6CheckBoxValueChanged(app, event)
            %% If the user unchecked/checked a sensor button in the analyzing tab,
            %% a new plot is made with the selected sensors
            updateAnalyzeAxes(app)
        end

        % Value changed function: Loadcell1CheckBox
        function Loadcell1CheckBoxValueChanged(app, event)
            %% If the user unchecked/checked a sensor button in the analyzing tab,
            %% a new plot is made with the selected sensors
            updateAnalyzeAxes(app)
        end

        % Value changed function: GraphDropDown
        function GraphDropDownValueChanged(app, event)
            %% If the user unchecked/checked a sensor button in the analyzing tab,
            %% a new plot is made with the selected sensors
            updateAnalyzeAxes(app)
        end

        % Value changed function: PeriodsCheckBox
        function PeriodsCheckBoxValueChanged(app, event)
            %% If the user unchecked/checked a sensor button in the analyzing tab,
            %% a new plot is made with the selected sensors
            updateAnalyzeAxes(app)
        end

        % Button pushed function: ResetScaleButton
        function ResetScaleButtonPushed(app, event)
            %% If the user clicks on the reset scale button, the scale factors will be
                reset,
            %% a new plot is made with scale 1
            analyzeFullTestData(app, app.a_fs, '1234567', '1x', app.a_Fvalve, ...
                    app.a_Nvalve, app.a_folder, 'FullTest', app.a_date)
        end

        % Button pushed function: AutoscaleButton
```

```
        function AutoscaleButtonPushed(app, event)
            %% If the user clicks on the auto scale button, the scale factors will be automatically calculated,
            %% a new plot is made with calculated scales

            % Process data to put into table
            currentDataTable = app.UITable2.Data;
            maxFFTValue = ones(6,1);
            for i = 1:6
                [maxFFTValue(i), ~] = max(app.a_fourier_NF(:,i));
            end

            referenceValue = max(maxFFTValue);

            for i = 1:6
                currentDataTable(i,4) = cellstr(num2str(referenceValue/maxFFTValue(i)));
            end

            app.UITable2.Data = currentDataTable;

            % Process data into graph
            updateAnalyzeAxes(app)
        end

        % Button pushed function: ResetAxesButton
        function ResetAxesButtonPushed(app, event)
            %% If the user clicks on the reset axes button, the axes limit are reset to these values
            if(app.GraphDropDown.Value == "FFT")
                xlim(app.UIAxes, [0 20]);
                ylim(app.UIAxes, 'auto');
            else
                xlim(app.UIAxes, 'auto');
                ylim(app.UIAxes, 'auto');
            end
        end

        % Button pushed function: SaveButton
        function SaveButtonPushed(app, event)
            %% If the user clicks on the save button, all important variables are saved into a .mat file
            %% Each property is assigned a variable so it can be read out in the workspace

            % Assigning properties to variables
            incomingData = app.incomingData;
            x_scale = app.x_scale;
            sensor_NF = app.sensor_NF;
            sensor_F = app.sensor_F;
            f_scale = app.f_scale;
            fourier_NF = app.fourier_NF;
            selectedFs = app.General_SampleFrequency.Value;
            description = app.DescriptionEditField.Value;
            frequency = app.ValveFrequency.Value;
            openTimes = app.ValveTimes.Value;
```

```
            selectedFolder = app.LocationEditField.Value;

            % If no location is chosen or if it does not exist, the current directory is chosen
            if(selectedFolder == "" || ~exist(selectedFolder, 'dir'))
                selectedFolder = pwd;
            end

            % Check if the single test, continuous or full test tab is active to determine the sensors and gain
            if (app.TabGroup3.SelectedTab == app.SingleTestTab)
                gain = app.Single_Gain.Value;
                if(app.Single_Sensor.Value == '7')
                    sensorNumber = 'loadcell';
                else
                    sensorNumber = app.Single_Sensor.Value;
                end
            elseif(app.TabGroup3.SelectedTab == app.ContinuousTestTab)
                gain = app.Continuous_Gain.Value;
                if(app.Continuous_Sensor.Value == '7')
                    sensorNumber = 'loadcell';
                else
                    sensorNumber = app.Continuous_Sensor.Value;
                end
            elseif (app.TabGroup3.SelectedTab == app.FullTestTab)
                Sensors = [app.Full_Sensor1.Value app.Full_Sensor2.Value app.Full_Sensor3.Value app.Full_Sensor4.Value app.Full_Sensor5.Value app.Full_Sensor6.Value];
                sensorNumber = '';
                for i = 1:6
                    if (Sensors(i) == true)
                        sensorNumber = [sensorNumber num2str(i)];
                    end
                end
                gain = '1x';
            end

            % Filename
            today = datestr(now, 'mmdd_HHMMSS');
            type = app.TypeDropDown.Value;
            fileName = [today '_' type '_fs' num2str(selectedFs) '_S' sensorNumber '_G' gain '_f' num2str(frequency) '_n' num2str(openTimes)];

            % Opening dialog box to let the user chose their save location
            app.MomoPESPTesterUIFigure.Visible = 'off';
            [name, path] = uiputfile('*.mat', 'File Selection', [selectedFolder '/' fileName]);

            if(path ~= 0)
                save(strcat(path,name), 'description', 'incomingData', 'x_scale', 'sensor_NF', 'sensor_F', 'f_scale', 'fourier_NF');
                app.UITable.Data = [{datestr(now, 'HH:MM:SS') '--- Measurement saved ---'};app.UITable.Data];
                if(app.LocationEditField.Value == "")
                    app.LocationEditField.Value = path;
                    app.WorkspaceLamp.Color = 'green';
```

```matlab
                    updateLog(app, 'Save location has been updated to your most recent one');
                end
            end

            app.MomoPESPTesterUIFigure.Visible = 'on';
        end

        % Button pushed function: SearchButton
        function SearchButtonPushed(app, event)
            %% Button to search for serial ports in the case the MCU was disconnected while running this app

            % (Instrument Control Toolbox required and serial port drivers)
            p = instrhwinfo('serial');
            app.SelectserialportDropDown.Items = p.AvailableSerialPorts;
        end

        % Button pushed function: AbortButton
        function AbortButtonPushed(app, event)
            %% If the user presses on the abort button, the processes inside MATLAB will stop as soon as possible
            app.emergencyStop = 1;
            updateLog(app, 'Abort button has been pressed, measurement will stop soon');
            app.stopContinuousTest = 1;
            enableRunButtons(app);
            return;
        end

        % Value changed function: Loadcell2CheckBox
        function Loadcell2CheckBoxValueChanged(app, event)
            %% If the user unchecked/checked a sensor button in the analyzing tab,
            %% a new plot is made with the selected sensors
            updateAnalyzeAxes(app)
        end

        % Button pushed function: OpenButton
        function OpenButtonPushed(app, event)
            %% If the user clicks on the open button,
            %% a dialog box will open to change the default save location
            app.MomoPESPTesterUIFigure.Visible = 'off';
            if(app.LocationEditField.Value == "" || ~exist(app.LocationEditField.Value))
                folder = uigetdir(pwd);
            else
                folder = uigetdir(app.LocationEditField.Value);
            end

            app.MomoPESPTesterUIFigure.Visible = 'on';
            if(folder ~= 0)
                app.LocationEditField.Value = folder;
                app.WorkspaceLamp.Color = 'green';
                updateLog(app, 'Save directory is succesessfully changed');
            end
        end
```

```
    end
```

F

Momo PE Sensor Plate Tester - Mbed code

```cpp
#include "mbed.h"
#include "Adafruit_ADS1015.h"
#include "USBSerial.h"

#define SERIAL_BAUD_RATE    921600
#define I2C_RATE                400000

DigitalOut valve(p23); // Pin to control the valve opening/closing
I2C SP(p28, p27); // Sensor Plate, SDA - SCL
I2C LC(p9, p10); // Load cell, SDA - SCL
Serial pc(USBTX, USBRX); // tx, rx

// ADC
Adafruit_ADS1015 piezo_electric_adc(&SP, 0x4B); // SP ADC 1
Adafruit_ADS1015 piezo_electric_adc2(&SP, 0x4A); // SP ADC 2
Adafruit_ADS1015 loadcell_adc(&LC, 0x48); // LC ADC
adsGain_t pga_table[]= {GAIN_SIXTEEN,GAIN_EIGHT,GAIN_FOUR,GAIN_TWO,GAIN_ONE};
uint8_t scaleTable[] = {1, 2, 4, 8, 16};

// Sensor value and its scale factor index
int loadcellValue = 0;
int electricValue = 0;
uint8_t scaleFactor_LC = 1;
uint8_t scaleFactor_PE = 1;

// Read Configuration
float sampleFrequency = 2500;
float duration = 0.0;
uint8_t channel_electric = 0;
uint8_t sensorNumber = 0;
uint8_t variableGain = 0;

// Valve Configuration
float valveFrequency = 1;
int nValveOpen = 1;
uint8_t loadcellPulse = 0;

// Variables for periodic tasks
Ticker s_PE; // Task for PE
Ticker s_LC; // Task for LC
Timer t;
bool ready = false;

// Test Variables
float tempTimer = 0;;

// Reads the ADC from the sensor
void getSingleElectric()
```

```
{
    // Invalid input
    if (sensorNumber > 5) {
        return;
    }

    // 6 PE sensors are split between 2 ADC’s, 3 PE sensors for each ADC
    channel_electric = sensorNumber%3;

    if (sensorNumber < 3) {
        // It uses the first ADC
        electricValue = piezo_electric_adc.readADC_Differential(channel_electric)*scaleFactor_PE;
    } else {
        // It uses the second ADC
        electricValue = piezo_electric_adc2.readADC_Differential(channel_electric)*scaleFactor_PE;
    }
}

// As long as the timer has not reached the duration, it will continue reading and writing data [PE]
void read_adc_PE()
{
    if (t.read() > duration) {
        t.stop();
        ready = false;
        s_PE.detach();
    } else if(ready == true) {
                // Get the current value in the ADC
        getSingleElectric();

        // Data is written through UART in ASCII
        // Datastream starts with sign (+/-), then 5 data digits, then carriage return and new line
        pc.printf("%+.5d\r\n", electricValue);
    }
}

// Reads the ADC from the load cell
void getLoadcellValue()
{
    loadcellValue = loadcell_adc.readADC_Differential(0)*scaleFactor_LC;
}

// As long as the timer has not reached the duration, it will continue reading and writing data [LC]
void read_adc_LC()
{
    if (t.read() > duration) {
        t.stop();
        ready = false;
        s_LC.detach();
    } else if(ready == true) {
        getLoadcellValue();

        // Data is written through UART in ASCII
        // Datastream starts with sign (+/-), then 5 data digits, then carriage return and new line
        pc.printf("%+.5d\r\n", loadcellValue);
    }
}

// Basic open and closing the valve
// Opens and closes based on the valve frequency and amount of times it should open/close
void valve_open()
{
    for(int i = 0; i<nValveOpen*2; i++) {
        valve = !valve;
        wait(1/(valveFrequency*2));
    }
}

// Basic open and closing the valve for load cell
// Should be a step function, the signal should be high till the end of the measurement
void loadcell_valve_open()
{
```

```
    valve = !valve;
    wait(duration-0.5);
    valve = !valve;
}

// The main process
int main()
{
    // Initializing settings
    SP.frequency(I2C_RATE);
    LC.frequency(I2C_RATE);
    NVIC_SetPriority(TIMER3_IRQn, 0); // Set ticker interrupt priorities as highest
    pc.baud(SERIAL_BAUD_RATE);
    valve = 1;

    while (1) {
        if(ready != true) {
                        // Waits for the MATLAB program to send the user configuration before reading out
            pc.scanf("%f %d %f %d %f %d %d", &sampleFrequency, &sensorNumber, &duration, &variableGain, &valveFrequency, &nValveOpen, &loadcellPulse);
            sensorNumber = sensorNumber - 1; // Sensor values in MCU are from 0-6, [0-5: sensor plate, 6: loadcell]

            if(sensorNumber < 6) {
                // Calls the function read_adc_PE (callback) periodicaly with interval provided as second argument (in micro seconds)
                s_PE.attach_us(&read_adc_PE, 1000000/sampleFrequency);

                // Set the gain factor of the PGA
                piezo_electric_adc.setGain(pga_table[variableGain]);
                piezo_electric_adc2.setGain(pga_table[variableGain]);
                scaleFactor_PE = scaleTable[variableGain];

                // Parameters are read, and MCU is ready to operate
                ready = true;
                t.reset();
                t.start();

                // MCU already starts reading the values, but the valve will open after a delay of 1 sec (arbitrary chosen)
                wait(1);

                                // Starts opening/closing the valve
                valve_open();
            } else if(sensorNumber == 6) {
                                // Calls the function read_adc_LC (callback) periodicaly with interval provided as second argument (in micro seconds)
                s_LC.attach_us(&read_adc_LC, 1000000/sampleFrequency);

                // Set the gain factor of the PGA
                loadcell_adc.setGain(pga_table[variableGain]);
                scaleFactor_LC = scaleTable[variableGain];

                // Parameters are read, and MCU is ready to operate
                ready = true;
                t.reset();
                t.start();

                // MCU already starts reading the values, but the valve will open after a delay of 1 sec (arbitrary chosen)
                wait(1);

                                // Based on the user settings, a pulse or a step is put on the load cell
                if(loadcellPulse == 1){
                    valve_open();
                } else {
                    loadcell_valve_open();
                }
            }
        }
    }
```

```
}
```