POLITECNICO DI TORINO

Master's Degree in Electronic Engineering

Master's Degree Thesis

# Characterization setup for biodegradable stretchable electrodes in the context of cardiac patches for organ transplant monitoring

**Supervisors**

Prof. Matteo COCUZZA

Prof. Clementine BOUTRY

**Candidate**

Edoardo DOMENELLA

July 2024

# Summary

This thesis focuses on the development of a characterization setup whose ultimate goal is to evaluate the performance of novel biodegradable stretchable electrodes intended for use in a more complex system, namely a cardiac patch. In general, the setup, implemented using LabVIEW and Arduino, provides a versatile platform for testing and characterizing flexible electronics. Performance evaluation of the setup was conducted to ensure its reliability and effectiveness. In addition to the setup development, flexible electrodes composed of PDMS and gold were fabricated during the thesis period. These electrodes were not intended for use in the cardiac patches; rather, they were created as a comparative benchmark against the innovative and biodegradable electrodes. Furthermore, they were designed to test the setup with a real-world use case and to begin exploring the characterization of such devices. The combination of the characterization setup and the fabricated electrodes represents a significant step towards the realization of reliable, effective monitoring systems and a new class of medical devices. Further research and refinement of both the setup and electrode fabrication processes will contribute to the advancement in this critical field.

# Table of Contents

# List of Tables

# List of Figures

# Acronyms

| | |
|---|---|
| **PDMS** | Polydimethylsiloxane |
| **MI** | Myocardial infarction |
| **LV** | Left ventricular |
| **FeNW** | Iron Nanowire |
| **POMAC** | poly(octamethylene maleate (anhydride) citrate) |
| **Au** | Gold |
| **DUT** | Device under test |
| **VI** | Virtual Instrument |

**GUI**

Graphical User Interface

**NFC**

Near Fiel Communication

**VISA**

Virtual Instrument Software Architecture

**FIFO**

First In First Out

# Chapter 1

# Introduction

Cardiovascular disease is one of the leading causes of mortality worldwide and can often lead to Myocardial infarction (MI) and other heart related illnesses which represents a formidable challenge in contemporary healthcare. Despite advancements in medical science, the ischemic insult inflicted by MI often culminates in irreversible damage to the myocardium, precipitating a cascade of events that compromise cardiac function and, ultimately, precipitate heart failure. The pathological hallmark of MI lies in the occlusion of coronary arteries, resulting in ischemia and subsequent necrosis of cardiomyocytes. This necrotic process triggers the formation of fibrotic scar tissue, predominantly within the left ventricular (LV) wall, disrupting the orderly propagation of electrical impulses and impeding synchronized ventricular contraction. Consequently, patients afflicted by MI face an elevated risk of life-threatening arrhythmias and progressive cardiac decompensation [1], [2]. In the quest for innovative therapeutic strategies, recent years have witnessed a burgeoning interest in the development of cardiac patches, a novel class of bioengineered constructs designed to enhance the structural and functional operation of the heart. These patches serve to different purposes: they provide mechanical support to the infarcted myocardium, concurrently they foster the restoration of native electrical conductivity and act as drug-delivery system for the damaged tissue [3]. Through the judicious integration of metals and polymers, these patches afford a temporary scaffold for cellular infiltration and tissue regeneration, ultimately promoting myocardial healing and functional recovery. Since wounded and fragile heart often can only tolerate light or non-invasive medical treatments, cardiac patches, typically composed of conductive materials, elastic polymeric substrates, and/or cardiac tissues resembling normal myocardium, offer an equivalent treatment approach to standard interventions such as heart transplantation. However, the development of minimally invasive cardiac patches poses a significant clinical challenge. Creating such patches requires simultaneous consideration of various materials attributes, including bioabsorption, non-toxicity, matching mechanical properties of heart tissues, efficient operation in wet and dynamic environments, electrical performances. The patch indeed is in contact with the heart tissue and must follow the heart beating during all the operational life. This obviously lead to the need of integrating conductive and non conductive materials in the patch that must coexist together ensuring mechanical and electronic performances. In this context soft and stretchable electronics acquire a predominant role and in the last years further improvement has been made in this direction. In order to acquire detailed signal about the heart activity from the patch high quality and stretchable electrodes are fundamental and represent one of the building blocks of the patch itself [4]. The goal of this project is to develop a characterization setup for biodegradable stretchable electrodes and sequentially use the setup to characterize and investigate the properties of novel biodegradable electrodes made up by iron nanowire (FeNW) on a POMAC, poly(octamethylene maleate (anhydride) citrate), substrate. The POMAC is a flexible and biodegradable polymer and the idea is to deposit the synthesised FeNW on the flexible substrate in order to build a stretchable electrode whose properties have to be investigated upon different strain conditions. To have a term of comparison with the FeNW electrodes also another type of electrodes made of gold (Au) and PDMS (Polydimethylsiloxane) has been developed and characterized.

Given their fragile and delicate nature, the characterization of such devices is a challenging task and hence a dedicated setup has been developed. The main chore of the setup is to stretch the electrode in a very precise manner in correspondence of a set of inputs given by software and to record the real time variation of the electrical parameters due to the applied elongation. The system components are here listed : PC that act as control station for the whole setup, Newmark eTRACK serie linear stage, motor driver TB6600, Arduino UNO board and Keysight Technologies E4980A precision LCR meter. To properly interface with the entire system and making everything automated the software from National Instrument LabVIEW (Laboratory Virtual Instrumentation Engineering Workbench) 2018 version has been adopted. LabVIEW is a system-design platform and development environment for creating custom applications that can monitor, control, and automate various processes and measurements. LabVIEW uses a graphical programming language called "G" (often referred to as "G code") that allows users to visually connect functional nodes to create programs known as virtual instruments (VIs). These VIs can incorporate hardware devices, analyze data, perform calculations, and present results in real-time graphical user interfaces (GUIs). LabVIEW is a very useful tool in those kind of applications, where multiple instruments need to be controlled and real time data acquisition is executed. A dedicated LabVIEW code has been developed and through the LabVIEW GUI both the linear stage and the LCR meter have been controlled to perform the stretching of the sample, acquiring real time impedance data. The DUT is clamped to the edges of the stage, one fixed and the other one movable, the stage is actually moved by a stepper motor integrated in the stage itself that let the user to precisely control the elongation applied to the electrode under test by defining the number of step that the stepper motor has to move. The stage is connected to the Arduino UNO board via the motor driver TB6600, thus a custom code for the Arduino has been generated to control the stepper motor. On the other end the LCR meter E4980A is connected on one side to the PC so that the instrument is read by the LabVIEW application and on the other side the output ports of the instrument are directly connected to the DUT (Device Under Test) using a pair of Kelvin Clips. However this is just an introductory description, a full detailed description of the setup building blocks and working principle will be given in the next chapters. This project is part of a major project whose goal is to manufacture a bioresorable cardiac patch that operates as an innovative medical device. The patch should accomplish to all the functionalities mentioned above, but, thanks to the biodegradable nature of the materials used ,it led to avoid further surgery that may be risky for the patient.

# Chapter 2

# Literature Review

In recent years, the challenge of coexistence between flexible electronic devices and skin tissue has been extensively investigated, to the extent that the class of "Lab-On-Skin" devices refers to all those electronic devices used in the medical field that operate in contact with the skin, providing indications on biopotential signals (such as ECG). These are nothing but stretchable and flexible electronic devices for health monitoring, whose physical properties seek to replicate those of the skin as closely as possible [5].

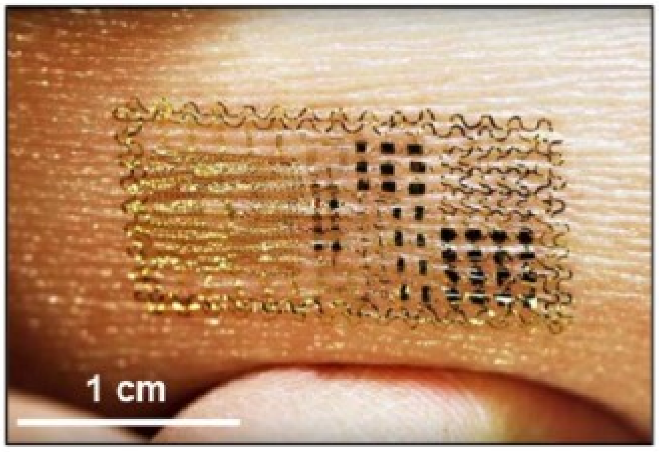


**Figure 2.1:** Lab-on-Skin example, NFC tattoo with bare die chip mounted on an arcylic adhesive film [5]

This short literature review chapter aims to give some insight about recent advancements about stretchable electronic technology in medical applications, with a particular focus on the integration of stretchable electrodes for real-time monitoring and therapeutic applications. Cardiac patches represent an innovative approach in the field of regenerative medicine, offering potential solutions for repairing damaged heart tissue and restoring cardiac functions. These patches, typically composed of biocompatible materials and often incorporating stem cells or growth factors, hold promise for treating conditions such as myocardial infarction and heart failure [6]. In Figure 2.2 an example of cardiac patch that integrates a PDMS substrate and a stretchable array of electrodes for sensing.

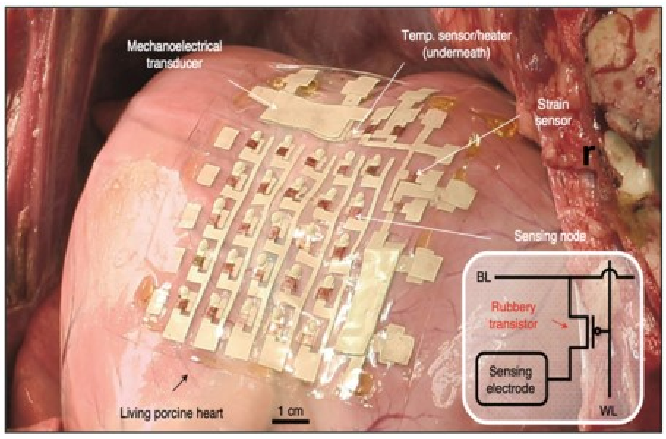


**Figure 2.2:** rubbery patch with 25 sensing nodes arranged in the format of a 5x5 thin-film transistor array on the epicardial surface of a living porcine heart.[6]

These electrodes play a crucial role in enabling real-time electrical monitoring of the heart's activity, providing valuable insights into cardiac function and facilitating the delivery of targeted therapies. The integration of stretchable electrodes into cardiac patches presents both opportunities and challenges, indeed while these electrodes offer unprecedented capabilities for monitoring cardiac activity and delivering therapeutic interventions, their successful implementation requires careful consideration of factors such as mechanical compatibility, biocompatibility, and long-term stability. In [7] a bioresorbable, highly conductive, and elastic cardiac patch is proposed. The patch is composed by poly(1,8-octamethylene-citrate-co-octanol) (POCO) substrate while the electrode is constitued by a metal mesh with a serpentine shape to enhance the intrinsic stretchability of the structure that must withstand the deformations of a beating heart. The decision, as observed in this instance, to structure the electrode with a rippled and winding pattern is a prevalent one. It's widely recognized that when materials are thin enough, they inherently become flexible due to bending strains that decrease proportionally with thickness. By arranging these structures into "wavy" configurations and adhering them to elastomeric substrates, we create systems capable not only of flexing but also of stretching and compressing. The most common way to achieve this result is to deposit a thin layer gold on PDMS substrate in order to achieve structures that can hold a large applied strain (Figure 2.3) [8].

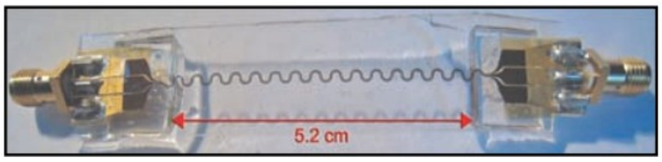


**Figure 2.3:** Serpentine metal coplanar waveguide embedded in a slab of PDMS and attached to standard SMA connectors for a stretchable, high-frequency cable.[8]

Regarding the electrode testing phase, in literature can be found numerous insights that, beyond minor variations, follow a similar approach. The device under test is mounted and secured onto a stage, one end fixed to the stationary side of the stage and the other end to the movable panel (Figure 2.4). By appropriately and precisely moving the stage, the desired elongation can be applied to the sample, and the data regarding impedance variation can be processed using a data acquisition board. LabVIEW is a commonly used software in setting up test benches as it allows for the easy creation of user-friendly graphical interfaces to control the setup [9],[10],[11], [12].

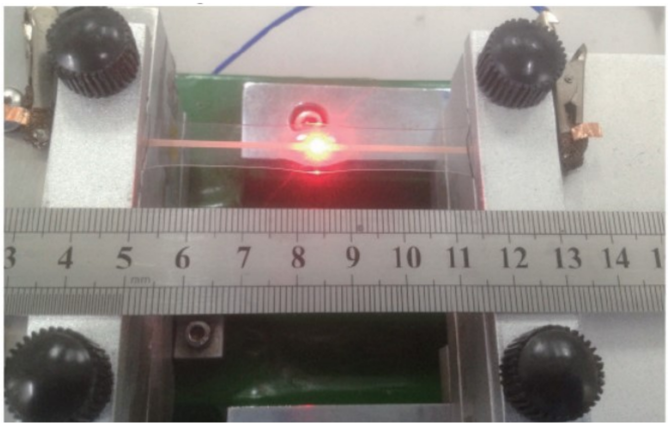


**Figure 2.4:** Example of printable stretchable circuit while being stretched.[9]

# Chapter 3

# Stretching Setup

In this chapter a detailed description of the developed stretching setup to characterize the stretchability of the flexible electronic samples is given. Firstly a general overview of the setup is presented and its working principle is provided in order to explain how the implementation of this setup allows for stretching and characterizing flexible electronics, particularly in this case, soft and stretchable electrodes. Each hardware and software component that constitutes the setup is described starting from its features and then moving on to the hardware connections in this particular case. Afterward the developed Arduino code to control the movement of the linear stage through the stepper motor is analyzed and in conclusion the final solution with the proposed LabVIEW code and GUI is described.

## 3.1 General Overview

At the heart of this project lies a fundamental question: how does the impedance of these components evolve in response to applied strain? The setup that has been developed and constructed was designed to meet the requirements necessary for the characterization and measurement of electrical parameters on flexible electronic components. What was needed was to stretch the DUT by a precise amount corresponding to a well-defined strain, while simultaneously measuring with equal precision the impedance value corresponding to the respective point in space, namely at the respective stretching point. The primary objective of this project is to investigate and quantify the intricate relationship between the applied elongation and the impedance characteristics of the DUT. By subjecting the DUT to controlled elongation along its x-axis, the aim is to elucidate the nuanced changes in its electrical properties. This endeavor is rooted in the fundamental understanding that the mechanical deformation of flexible electronic components inherently influences their electrical performance. Moreover, this project serves as a testament to our commitment to advancing the frontier of flexible electronics. By systematically characterizing the impedance response of the DUTs under different strain levels, the goal is to pave the way for the development of next-generation flexible electronic devices with enhanced performance and reliability.

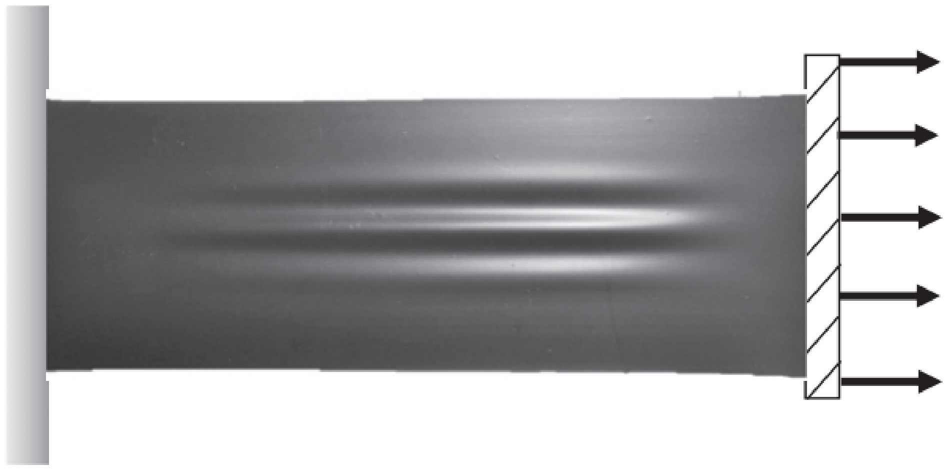


**Figure 3.1:** Optical image of a polyethylene sheet under stretch [13]

To match the stringent project requirements and achieve optimal precision both in terms of elongation and measurement, some decisions are made regarding our equipment selection. A linear stage integrated with a stepper motor has been employed for controlled elongation, and an LCR meter for accurate measurements. The choice of a linear stage with a stepper motor offers unparalleled control and repeatability in the elongation process. The stepper motor allows to precisely control the applied elongation by properly defining the input parameters according to our project specifications, ensuring consistency and reliability in our experimental procedures. With the ability to program precise movement sequences, we can confidently explore a wide range of strain conditions with utmost accuracy. Indeed, as stated by the name a stepper motor operates by dividing a full rotation of 360°into a fixed number of discrete steps. Each step of the motor corresponds to a precise angle of rotation, which can be controlled accurately. Each step is thus translated in an equivalent linear movement of the stage, with a sub-millimiter resolution. Similarly, the choice of an LCR meter for measurement purposes embodies the commitment. Renowned for its high precision and reliability, the LCR meter facilitated the capability to accurately characterize the electrical properties of our specimens. Its advanced features and intuitive interface lead to perform comprehensive impedance measurements with confidence and efficiency. Hence, the two main components of the setup are the linear stage and the LCR meter but they are not the only ones. To automatized the stretching sequence an Arduino UNO board has been chosen, thanks to the "AccelStepper.h" library provided by Arduino the board is programmed to control the motor and consequently move the stage based on certain input parameters that are : the number of step that the motor has to do correspondent to a given elongation, the acceleration and the speed of the motor, the number of times that the stretching sequence has to be performed and the resting time at each ending position. Through these input parameters, it is possible to modify and vary the stretching of the device, allowing for a multitude of different experiments to be conducted, from simulations where the elongation is sequentially increased from a minimum to a maximum, or simulations where the elongation is constant but stretching is repeated for a greater number of cycles. In Figure 3.2 a schematic representation of the setup with all the connections between hardware components is showed.

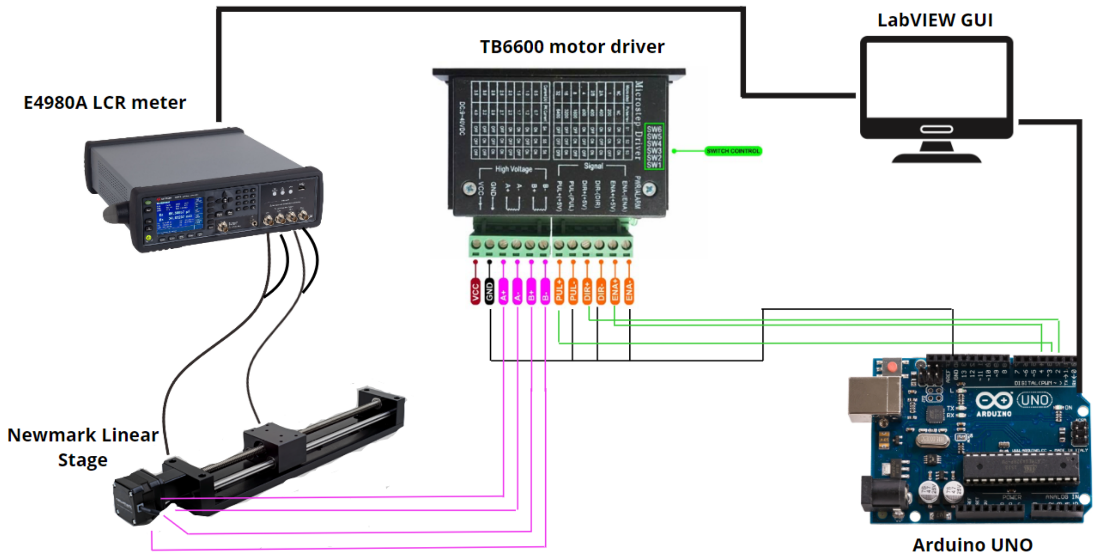


**Figure 3.2:** Schematic of the developed setup with its main building blocks

Then a motor driver TB6600 is adopted to interface the board with the stage wiring the pins of the Arduino to the correspondent ports of the stage, moreover the driver is capable to withstand the voltage and current needed to ensure optimum performance of the motor. In conclusion, to achieve a fully automated and PC-controlled system that is user-friendly and allows for the utilization of the setup by anyone requiring measurements on flexible electronics, a LabVIEW code and corresponding graphical user interface (GUI) have been developed. The developed GUI enables the definition of the aforementioned parameters as well as those related to the settings of the LCR meter. It subsequently controls all the instruments responsible for stretching the DUT, displays the collected results, and saves them in an Excel file for further analysis. The working principle of the setup is almost straightforward: the DUT is directly clamped on the stage and attached on one side to the fixed edge of the stage while on the other side is attached on the movable part of the stage. As already said the controls are defined through the GUI and sent via serial communication to the Arduino. The board execute the parsing of the command and based on these control the stepper motor that consequentially move the stage applying the wanted elongation on the device. At the same time, once that the LabVIEW code is ran, the LCR meter setting are sent through serial communication as well and the instrument is configured for the measurement. The LCR meter ports are directly connected to the DUT by a kelvin clips and the impedance values are measured and plotted on three different graphs of the GUI.

## 3.2 Components

### 3.2.1 Linear Stage

The Newmark eTrack Series linear stage, Figure 3.3, is a type of precision positioning system used in various scientific, industrial, and research applications. It offers high accuracy, repeatability, and stability in motion control, making it versatile for a wide range of applications, in particular for those tasks requiring precise linear movement. The stage is engineered to provide precise linear motion with sub-micron resolution, allowing for accurate positioning of objects or tools. It offers excellent stability during motion, minimizing vibrations and ensuring smooth movement, which is crucial for applications requiring precise positioning. It is compatible with various motion control systems, allowing for seamless integration into existing setups or automation systems.

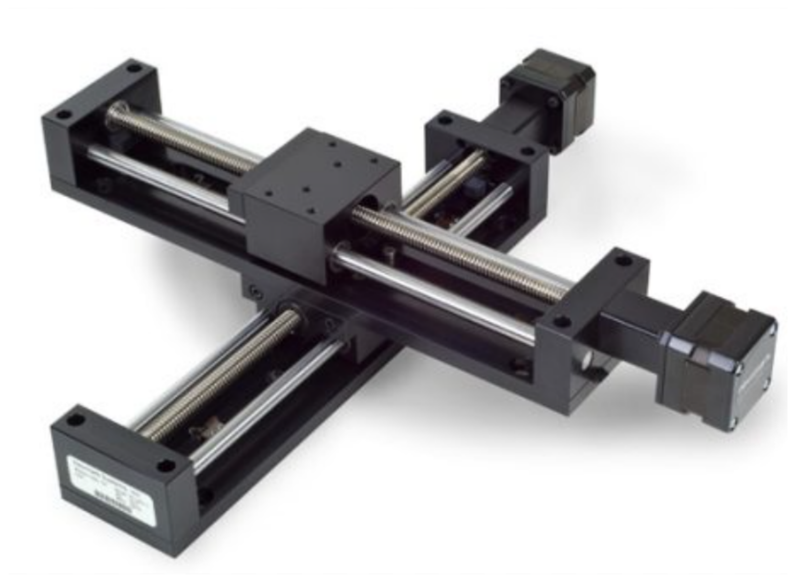


**Figure 3.3:** Newmark eTrack Series Linear stage

Hence, within the scope of this project, the linear stage aligns perfectly with the objective of precisely stretching the electrodes. Its main features include [14]:

1. Travel Range 300 mm
2. Resolution 0.04 um
3. 2 mm pitch lead screw
4. Maximum Speed 25 mm/sec

Furthermore, the movable part of the stage is controlled by a NEMA 17 bipolar stepper motor whose main features are [14]:

1. Step Size : 1.8 °/Step
2. Amps/Phase : 1.33
3. Holding Torque : 42 oz-in
4. -Resistance : 2.5 Ω/Phase

The NEMA 17 stepper motor is a two-phase stepper motor, this means that it has two sets of windings (coils) that are energized in sequence to generate rotational motion. These motors are arranged in a bipolar configuration, meaning that there are 4 phase (A, A',B, B'),each phase has two wires and to drive the motor the current is alternately applied to each coil to create a magnetic field that interacts with the motor's permanent magnets, causing the rotor to move in discrete steps. To control the motion of a NEMA 17 stepper motor, a dedicated stepper motor driver circuit is typically used. This driver circuit regulates the current flowing through the motor coils and determines the sequence in which the coils are energized to produce motion, in this project a motor driver TB6600 is used. The NEMA 17 stepper motor typically has a step angle of 1.8 degrees per step, meaning that it requires 200 steps (360 degrees / 1.8 degrees per step) to complete one full revolution.

Many stepper motor drivers, including those used with NEMA 17 motors, support microstepping. Microstepping is a technique used in stepper motors to increase resolution and precision of movement. Instead of advancing the stepper motor by a single full step at a time, as in standard operation, microstepping divides each step into a series of smaller fractions. This allows the motor to move more smoothly and precisely, also reducing noise and vibrations. To achieve microstepping, the driver circuit for the stepper motor controls the current in each phase of the motor windings with varying levels of intensity. By energizing the coils with different current levels at precise timing intervals, the motor can smoothly transition between microsteps, allowing for finer control over the motor's position and movement. In this project the microstepping technique is adopted increasing the number of steps per revolution to 800 in order to increase the resolution given to the elongation applied to the samples under test. Indeed, with 800 steps per revolution the step angle become

$$step_{angle} = \frac{360°}{800 steps per revolution} = 0.45°/step \tag{3.1}$$

which correspond, considering a pitch lead screw of 2mm, to a linear movement of the stage of

$$step_{mm} = \frac{2mm}{800step} = 0.0025mm \tag{3.2}$$

each step.

### 3.2.2 Motor Driver TB6600

The TB6600 motor driver is a type of stepper motor driver commonly used in various automation and motion control applications. It provides precise control over stepper motors, allowing for accurate and reliable motion in robotic systems, CNC machines, 3D printers, and other industrial and hobbyist projects. The TB6600 driver can handle relatively high currents, making it suitable for driving stepper motors with larger torque requirements, moreover it supports microstepping [15], a technique that allows for smoother motion and finer resolution by dividing each step of the motor into smaller increments. Users can adjust the motor current settings and the type of microstepping adopted through dip switches on the driver board, allowing for optimization of motor performance and power consumption. The device also includes protection features such as overcurrent protection and thermal shutdown to prevent damage to the motor or driver due to excessive current or temperature and it is compatible with a wide range of stepper motors, making it versatile for different applications and motor specifications.

The stepper motor driver TB6600 has 12 pins:

1. VCC and GND are the two power pins directly connected to the power supply, the voltage can range between 9 V and 24 V, but to have good performance and to move correctly the motor at least 20 V must be used.

2. A+ and A- pins are used to energize one of the two coils of the two-phase stepper motor, while B+ and B- are used for the other one. These 4 pins are wired to the motor trough a male connector installed on the stage and must be properly controlled to pilot the motor and move the stage.

3. PUL+ (Pulse+): This is the positive pulse pin. When a positive pulse is provided to this pin, the driver executes a step of the stepper motor.

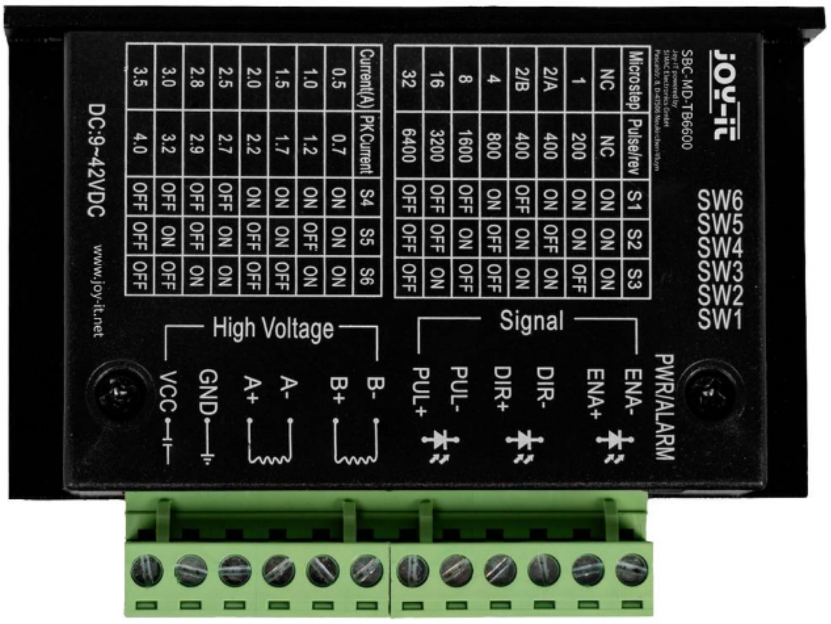


**Figure 3.4:** Motor driver TB6600

4. PUL- (Pulse-): This is the negative pulse pin. It is used in conjunction with the PUL+ pin to provide control pulses to the driver.

5. DIR+ (Direction+): This is the positive direction pin. It controls the rotation direction of the stepper motor. When a high logic signal (e.g., +5 V) is provided, the motor rotates in one direction, whereas when a low logic signal (e.g., 0 V) is provided, the motor rotates in the opposite direction.

6. DIR- (Direction-): This is the negative direction pin. It is used along with the DIR+ pin to control the direction of rotation of the stepper motor.

7. ENA+ (Enable+): This is the positive enable pin. When a high logic signal is applied to this pin, the driver is enabled, allowing it to respond to control signals. When a low logic signal is applied,

the driver is disabled, and the motor stops.

8. ENA- (Enable-): This is the negative enable pin. It is used in conjunction with the ENA+ pin to enable or disable the driver.

These pins play crucial roles in controlling the operation of the TB6600 stepper motor driver, allowing precise control over the motion and direction of the stepper motor.

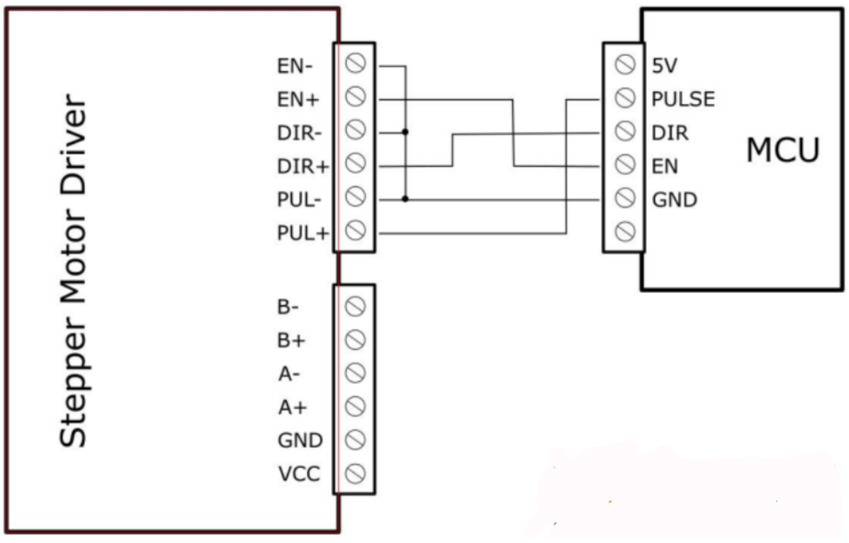


**Figure 3.5:** Motor driver TB6600 - acquisition board common anode connection schematic

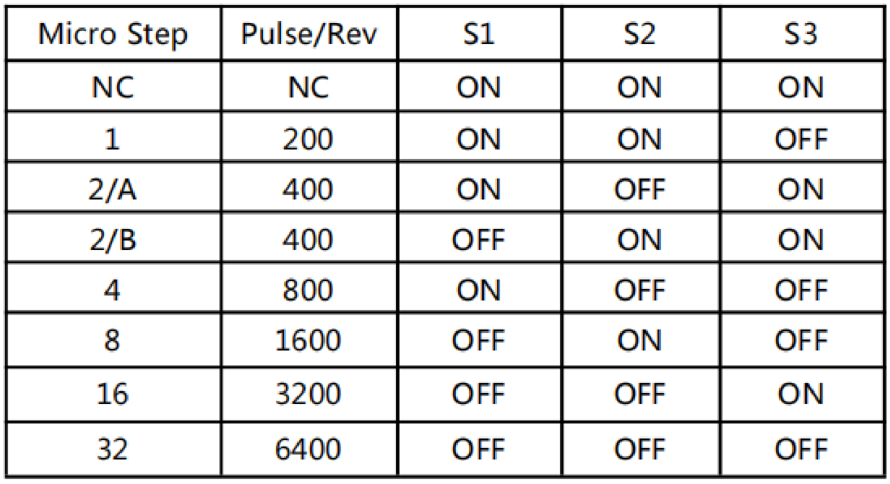
In Figure 3.5 the implemented configuration and the connections between the driver and the Arduino UNO board is showed. In Particular a Common Anode connection for the motor driver TB6600 is chosen, so all the negative signal (-) are grounded while the positive ones are connected to the Arduino pins; eventually also a Common Cathode could be used, in this situation the pin configuration is swapped. In the common anode configuration the pins are hence disposed :

1. ENA- , DIR- , PUL- -> grounded.

2. ENA+ -> Connected to an Arduino pin as OUTPUT.

3. DIR+ -> Connect to an Arduino pin as OUTPUT.
4. PUL+ -> Connect to an Arduino pin as OUTPUT.
5. B+ and B- -> Connect to Phase B Phase B' of the motor (MALE cable).
6. A+ and A- -> Connect to Phase B Phase B' of the motor (MALE cable).
7. VCC -> Connect to the high power supply.
8. GND -> Connect to ground.

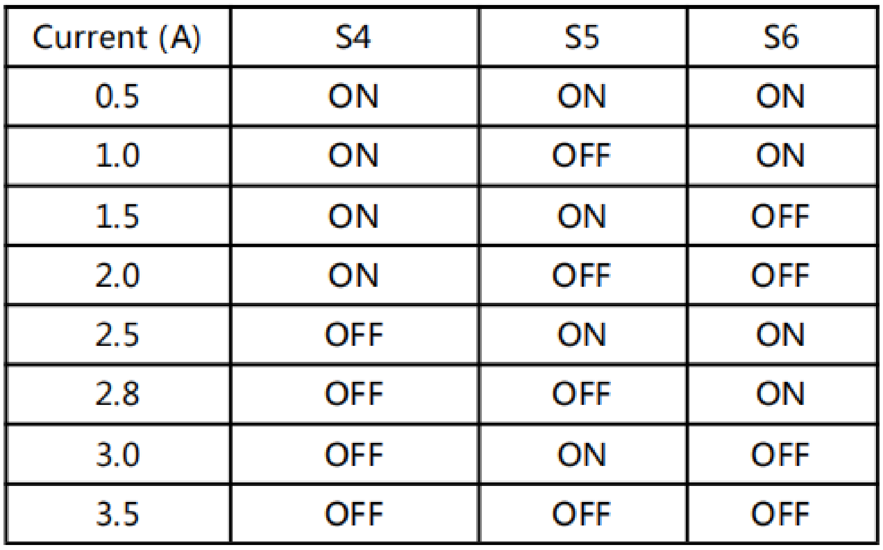
On the TB6600 are presents 6 dip switches (SW1, SW2, SW3, SW4, SW5, SW6) that can be used to implement different microstepping technique and to apply an higher current to the motor. The first three, SW1, SW2 and SW3 can be used to change the microstepping, while the last three, SW4, SW5, SW6 are used to modify the current setting. In Figure 3.6 the possible microstepping configurations are summarized.



| Micro Step | Pulse/Rev | S1 | S2 | S3 |
|---|---|---|---|---|
| NC | NC | ON | ON | ON |
| 1 | 200 | ON | ON | OFF |
| 2/A | 400 | ON | OFF | ON |
| 2/B | 400 | OFF | ON | ON |
| 4 | 800 | ON | OFF | OFF |
| 8 | 1600 | OFF | ON | OFF |
| 16 | 3200 | OFF | OFF | ON |
| 32 | 6400 | OFF | OFF | OFF |

**Figure 3.6:** Possible microstepping configurations allowed by TB6600 Driver

While in Figure 3.7 the allowed cuurent settings are listed.



| Current (A) | S4 | S5 | S6 |
|---|---|---|---|
| 0.5 | ON | ON | ON |
| 1.0 | ON | OFF | ON |
| 1.5 | ON | ON | OFF |
| 2.0 | ON | OFF | OFF |
| 2.5 | OFF | ON | ON |
| 2.8 | OFF | OFF | ON |
| 3.0 | OFF | ON | OFF |
| 3.5 | OFF | OFF | OFF |

**Figure 3.7:** Possible current configurations allowed by TB6600 Driver

In the chosen configuration the number of step is set to 800 while the current to 1.5 A This provides a switch configuration of: ON OFF OFF ON ON OFF.

### 3.2.3 Arduino UNO

The Arduino Uno is a popular open-source microcontroller board based on the ATmega328P microcontroller chip. It is widely used in electronics prototyping, hobbyist projects, and educational settings due to its ease of use, versatility, and affordability. The Uno is powered by the ATmega328P microcontroller chip, which provides processing power and input/output (I/O) capabilities for interfacing with sensors, actuators, and other electronic components. The board includes a set of digital input/output pins (14 in total), which can be used to read digital signals from sensors or control digital devices like LEDs or motors. Additionally, there are analog input pins (6 in total) that can read analog voltage values from sensors.

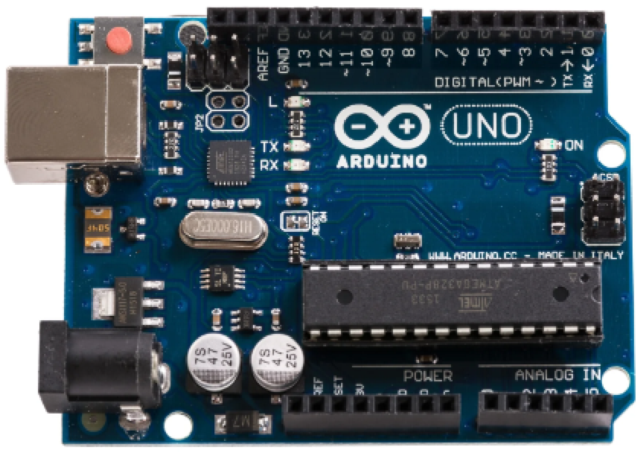


**Figure 3.8:** Arduino UNO board

The Uno can be easily connected to a computer via USB, allowing for programming and communication with the microcontroller using the Arduino Integrated Development Environment (IDE). In the context of this project, the Arduino Uno has proven to be incredibly valuable.

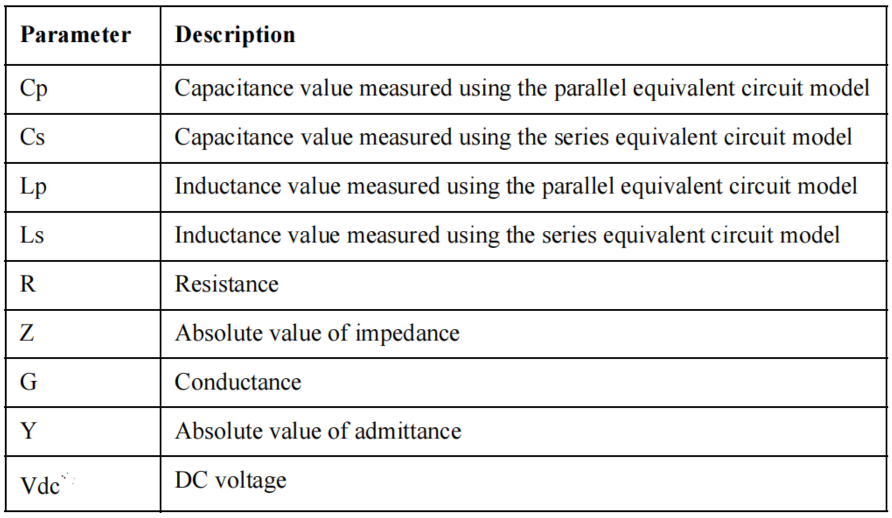
Initially, it facilitated motor testing by programming the board using the AccelStepper.h library. Subsequently, it enabled seamless integration with LabVIEW, allowing for comprehensive control of the entire system to drive the motor. For the hardware configuration of the connections, four pins of the Arduino were utilized. The GND pin was used to ground the ENA-, DIR-, and PUL- pins of the motor driver. Digital pin 2 was connected to DIR+, digital pin 3 was connected to PUL+, and finally, digital pin 4 was connected to ENA+.

### 3.2.4 LCR Meter

The Precision LCR Meter E4980A is a sophisticated electronic instrument used for measuring the electrical properties of passive electronic components, such as resistors, capacitors, and inductance.

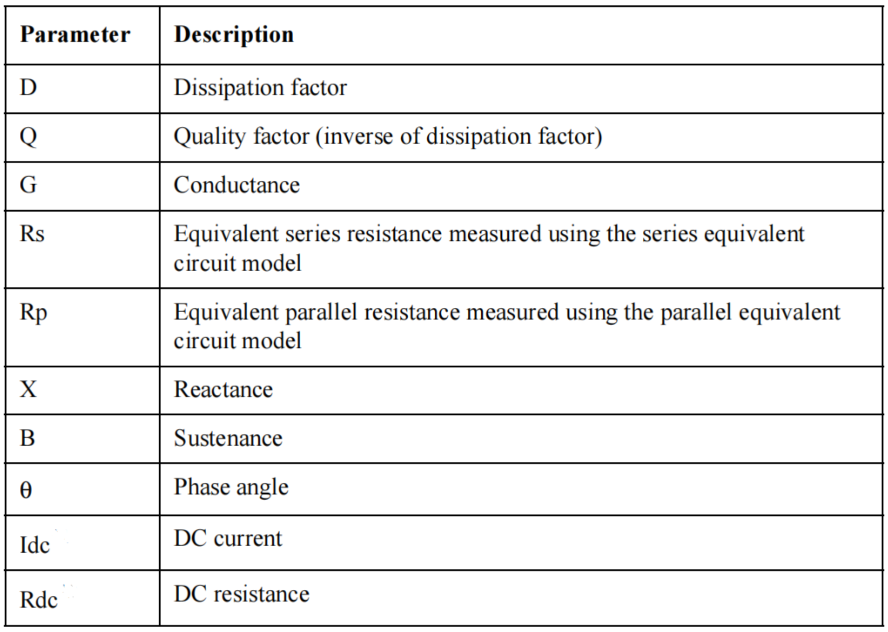
**Figure 3.9:** E4980A Precision LCR Meter

The E4980A offers high measurement accuracy and precision, making it suitable for demanding applications where precise characterization of components is essential. It covers a broad frequency range, from 20 hertz to 2 megahertz, allowing for comprehensive testing of components across various frequency domains. The instrument supports multiple measurement modes, including impedance (Z), admittance (Y), capacitance (C), inductance (L), resistance (R), and other parameters. This versatility enables comprehensive analysis of different types of components. It often features automatic measurement capabilities, allowing users to quickly and efficiently measure multiple components without the need for manual adjustments. The LCR meter E4980A adopt a differential measurement method to minimize the effects of parasitic components or wiring errors on electronic component measurements. This approach involves using two separate connections to the DUT, allowing for the measurement of the voltage or current difference between them. An electrical signal is applied to the DUT through the two connections, this signal can be an alternating current (AC) or direct current (DC), depending on the measurement requirements. The LCR meter then measures the voltage or current difference between the two connections of the DUT, since the difference is indicative of the electrical properties of the DUT itself, as it eliminates the effects of parasitic components or wiring errors that may affect measurements made with a single connection. Finally, the measured data is processed and analyzed to determine the electrical properties of the DUT, such as impedance, resistance, capacitance, or inductance, depending on the type of measurement performed. The measurements on the LCR meter are performed with the electrical parameters to be measured coupled in pairs, referred to as primary parameters and secondary parameters. In Figure 3.10 and Figure 3.11 are reported all the parameters that can be evaluated through the LCR meter whit a short related description. In Figure 3.12 instead all the possible measurement combinations are listed [16].



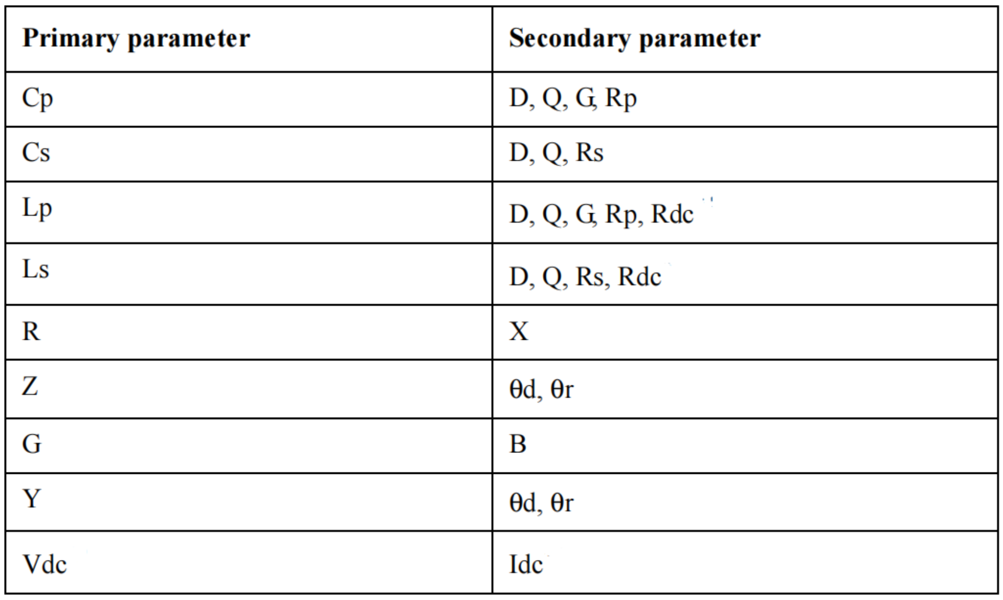
| Parameter | Description |
|---|---|
| Cp | Capacitance value measured using the parallel equivalent circuit model |
| Cs | Capacitance value measured using the series equivalent circuit model |
| Lp | Inductance value measured using the parallel equivalent circuit model |
| Ls | Inductance value measured using the series equivalent circuit model |
| R | Resistance |
| Z | Absolute value of impedance |
| G | Conductance |
| Y | Absolute value of admittance |
| Vdc | DC voltage |

**Figure 3.10:** Parameter description LCR meter E4980A (1)



| Parameter | Description |
|---|---|
| D | Dissipation factor |
| Q | Quality factor (inverse of dissipation factor) |
| G | Conductance |
| Rs | Equivalent series resistance measured using the series equivalent circuit model |
| Rp | Equivalent parallel resistance measured using the parallel equivalent circuit model |
| X | Reactance |
| B | Sustenance |
| θ | Phase angle |
| Idc | DC current |
| Rdc | DC resistance |

**Figure 3.11:** Parameter description LCR meter E4980A (2)



| **Primary parameter** | **Secondary parameter** |
|---|---|
| Cp | D, Q, G, Rp |
| Cs | D, Q, Rs |
| Lp | D, Q, G, Rp, Rdc |
| Ls | D, Q, Rs, Rdc |
| R | X |
| Z | θd, θr |
| G | B |
| Y | θd, θr |
| Vdc | Idc |

**Figure 3.12:** Measurement combinations of primary and secondary parameter with LCR meter E4980A

## 3.3 Algorithm

In this paragraph is described the algorithm that defines the motor speed profile and the accelerated motion of the motor itself to obtain the stage velocity profile. This was necessary to thoroughly study the performance and operation of the motor and to evaluate the time taken by the motor to achieve a certain displacement based on the set acceleration. Additionally, is reported and described the Arduino codes implemented following the aforementioned algorithm.

### 3.3.1 Stepper Motor Speed Profile

The development of an algorithm to calculate the stepper motor speed profile represents a pivotal aspect of this project, indeed understanding and effectively controlling the speed profile of the stepper motor is crucial for achieving precise and efficient motion control. Moreover through this algorithm I've extract the precise timing required for the motorized stage to traverse a specified distance. By inputting the N number of steps corresponding to a distance in millimeters and the desired acceleration, the algorithm calculates the time required for the stage to complete the designated travel. This calculated time is of paramount importance in both the design and operation of the developed setup. It ensures that the motorized components move accurately and efficiently, aligning with the project's objectives of precision and reliability. However, the main reason for calculating the time it takes for the stage to travel a certain distance is crucial for maintaining synchronization in the serial communication between LabVIEW and Arduino. Specifically, real-time data regarding the position of the stage is sent from Arduino to LabVIEW through the serial connection. This allows for the generation of a displacement vs. time graph, which can be compared with the respective impedance values. To acquire and display the data about the real time position of the stage, Arduino sends data about the stage position to LabVIEW at regular intervals, typically one value for every certain number of steps completed by the stage. By knowing the time it takes for the stage to complete this number of steps and setting the sampling frequency of the LabVIEW system equal to the frequency of data transmission over the serial connection from Arduino, we ensure accurate data transmission and sampling. The algorithm employed in this project is extracted from the "AccelStepper.h" library [17], which is provided by Arduino. It's worth noting that this library, as explicitly stated in its documentation, relies on this very algorithm. Through this approach is it possible to calculate both acceleration and deceleration curves, including an intermediate phase characterized by a constant velocity for the motor. Furthermore, it efficiently computes the time required for the motor to accelerate from a standstill to the desired maximum velocity, and likewise for the deceleration phase from maximum velocity to a stop. This process involves segmenting the total number of steps into three distinct phases. There are N1 steps dedicated to the linear acceleration phase, during which the motor gradually ramps up to the desired velocity, subsequently, N2 steps are allocated for maintaining a steady velocity and finally, the remaining N3 steps are utilized for the deceleration phase, gradually bringing the motor to a complete stop. This nuanced approach ensures precise control over the motor's motion profile, facilitating smooth and accurate movement within the system. The process unfolded in two stages: initially, the algorithm was studied and an Arduino code implementing it was generated. Subsequently, the same Arduino code underwent further refinement and adaptation to seamlessly integrate the algorithm into the LabVIEW code. This integration was necessary to incorporate the previously mentioned stage travel time as a parameter to be given in input to the LabVIEW system since it ensured synchronization in the serial communication between the LabVIEW interface and Arduino, facilitating real-time data transmission and effective coordination between the two platforms. In Figure 3.13 is described the working principle of the algorithm and the behaviour that it replicates.

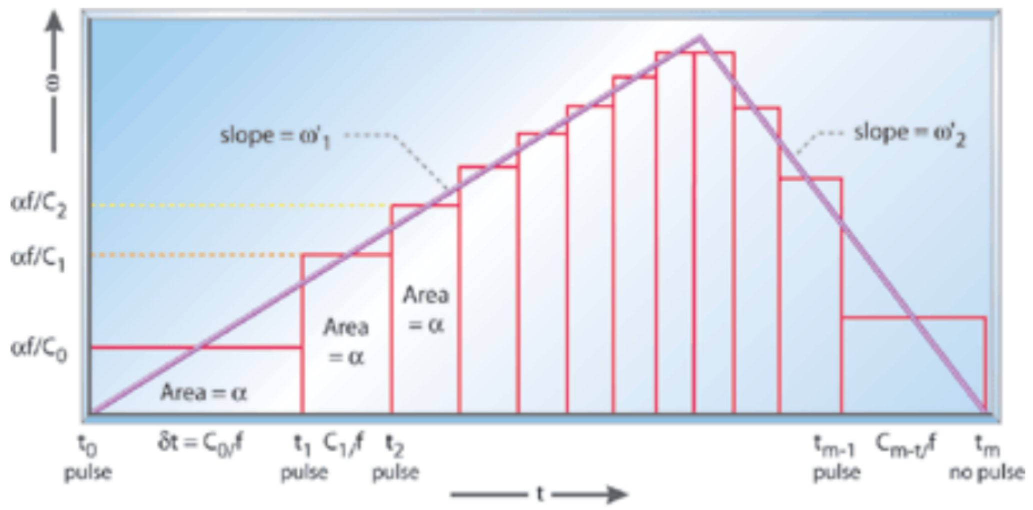


**Figure 3.13:** Stepper motor speed profile, the proposed algorithm aims to compute the series of the coefficient C that define the time interval between each pulse [18].

In the provided algorithm, the speed is increased by varying the coefficient C, which in turn affects the time t between each step pulse. From the graph in the picture, the x-axis represents time t, and the distance between two successive time instants $t_n - t_{n-1}$ corresponds to the distance between two pulses and thus two steps taken by the motor. On the y-axis, there is the angular speed $\omega$. The algorithm and the motion of the motor revolve around the series of divisor coefficients $c_n$; indeed, these are constantly updated starting from $c_0$ and divided by the pulse generator frequency f. The division between the coefficients $c_n$ and f defines the time interval $\delta$t between two successive pulses. Considering that the frequency is constant, progressively decreasing the value of the n-th c will result in a progressively shorter time interval between two successive pulses, corresponding to an increase in the motor's speed, which will rotate faster. The area under each pair of xy points on the graph is the motor's angular step $\alpha$ and remains constant throughout the motor's operation, while the slope of the line $w'$ is the

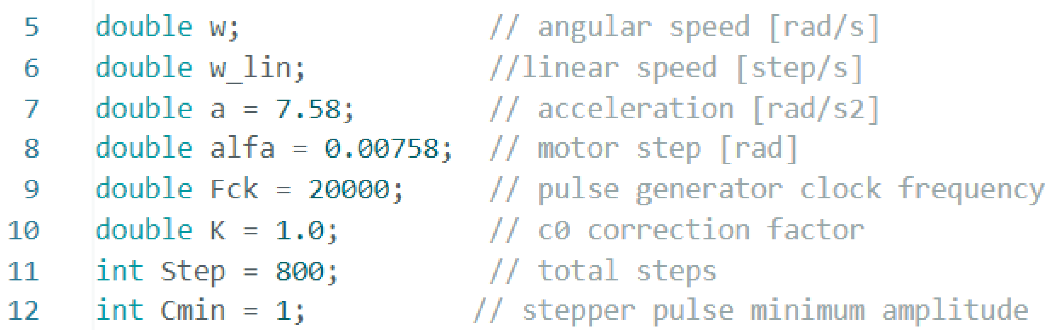
motor's angular acceleration.
The algorithm underlying the Arduino AccelStepper.h library is designed to control the stepper motor to achieve a desired speed profile, including acceleration, constant speed motion, and deceleration phases. During the acceleration phase, the motor gradually accelerates from its initial velocity to the desired speed. This calculation accounts for the number of steps required to smoothly reach the desired speed and once the desired speed is reached, the stepper motor moves at a constant speed for the required duration. During this phase, the number of steps per unit time remains constant, ensuring smooth and stable motion. When it is necessary to stop the motor or change its direction, the algorithm initiates deceleration to gradually bring the motor to a halt. Deceleration occurs at a constant rate until the motor stops.
In the next section the full algorithm is reported and described.

**Algorithm 1** Stepper motor speed profile : the goal is to calculate the linear speed ramp of a stepper motor real time. This approach approximates the motor behavior by accelerating constantly until reaching maximum speed, then maintaining a constant speed until approaching the destination, and decelerating constantly until coming to the selected stopping point.

```
1: ▷ An oscillator with frequency $F_{CK}$ is considered, this is the frequency of the pulses of the motor, hence the frequency of the steps
2: ▷ The series of coefficient $c$ is divided by the frequency $F_{CK}$, varying $c$ properly is it possible to change the frequency of the steps and hence the speed of the motor
3: ▷ $\alpha$ is the motor step angle (radian)
4: ▷ $F_{CK}$ is the timer frequency (Hz)
5: ▷ $a$ is the angular acceleration
6: ▷ $c$ is the timer count
7: ▷ $t(n)$ is a generic time instant
8: $\Delta_{t(n)} = \frac{c_n}{F_{CK}}$ ▷ time that divides two motor steps and it last $\Delta_{t(n)}$
9: $\Delta_{t(n-1)} = \frac{c_{n-1}}{F_{CK}}$ ▷ previous motor step
10: ▷ In Figure 3.13 each rectangle has an area equal to $\alpha$ so:
11: $w_n = \frac{\alpha}{\Delta_{t(n)}} = \frac{\alpha}{\frac{c_n}{F_{CK}}} = \frac{\alpha F_{CK}}{c_n}$
12: $c_n = \frac{\alpha F_{CK}}{w_n}$
13: ▷ It's possible to write $w_n$ as a function of $w_{n-1}$
14: $w_n = w_{n-1} + a \cdot \Delta_t$
15: $\Delta_{t(n)} = \Delta_{t(n-1)}$ ▷ In the time interval $\Delta_{t(n-1)}$ the motor goes from $w_{n-1}$ to $w_n$
16: ▷ Replacing $w_n$ in the $c_n$ formula we can now obtain
17: $c_n = \frac{\alpha F_{CK}}{w_n} = \frac{\alpha \cdot F_{CK}}{w_{n-1} + a \cdot \Delta_t(n-1)}$
18: ▷ Replacing
19: $w_{n-1} = \frac{\alpha \cdot F_{CK}}{c_{n-1}}$
20: $\Delta_{t(n-1)} = \frac{c_{n-1}}{F_{CK}}$
21: ▷ We get
```

```
22: $c_n = \frac{\alpha \cdot F_{CK}}{\frac{\alpha \cdot F_{CK}}{c_{n-1}} + a \cdot \frac{c_{n-1}}{F_{CK}}}$
23: $c_n = \frac{c_{n-1}}{1+\frac{a}{\alpha \cdot F_{CK}} \cdot {c_{n-1}}^2} = \frac{c_{n-1}}{1+K_a} \cdot {c_{n-1}}^2$        ▷ This formula permits to calculate the $c_n$ dynamically while the motor proceed knowing the precedent $c_{n-1}$
24: ▷ with
25: $K_a = \frac{a}{\alpha \cdot F_{CK}}$
26: ▷ Now we can compute $c_0$
27: $\Delta_{t(0)} \frac{c_0}{F_{CK}}$
28: $w_0 = \frac{\alpha \cdot F_{CK}}{c_0}$
29: $a \cdot \Delta_{t(0)} = a \cdot \frac{c_0}{F_{CK}} = 2 \cdot w_0 = 2 \cdot \frac{\alpha \cdot F_{CK}}{c_0}$
30: $c_0 = F_{CK} \cdot \sqrt{\frac{2 \cdot \alpha}{a}}$
31: ▷ In conclusion
32: $c_0 = F_{CK} \cdot \sqrt{\frac{2 \cdot \alpha}{a}}$
33: $c_n = \frac{c_{n-1}}{1+K_a} \cdot {c_{n-1}}^2$
34: $K_a = \frac{a}{\alpha \cdot F_{CK}}$
```

At this point, the algorithm has been adapted in C language and written in the Arduino IDE to be tested, evaluating the achieved time and speed variations as the input parameters change, particularly with varying input acceleration.
Following are reported the main parts of the implemented code considering an acceleration $a_{lin} = 1000$ steps/s.



```
double w;                   // angular speed [rad/s]
double w_lin;               //linear speed [step/s]
double a = 7.58;            // acceleration [rad/s2]
double alfa = 0.00758;      // motor step [rad]
double Fck = 20000;         // pulse generator clock frequency
double K = 1.0;             // c0 correction factor
int Step = 800;             // total steps
int Cmin = 1;              // stepper pulse minimum amplitude
```

**Figure 3.14:** Input variable definition (1)

1. w: Angular speed of the motor, measured in radians per second (rad/s). $w_{lin}$: Linear speed of the motor, measured in steps per second (steps/s), it is an output of the code.

2. alfa: Motor step size, measured in radians per step (rad), considering 800 steps per revolution

$$alfa = \frac{360}{800} \cdot \frac{\pi}{180} \tag{3.3}$$

3. a: Acceleration of the motor, measured in radians per second squared (rad/$s^2$).

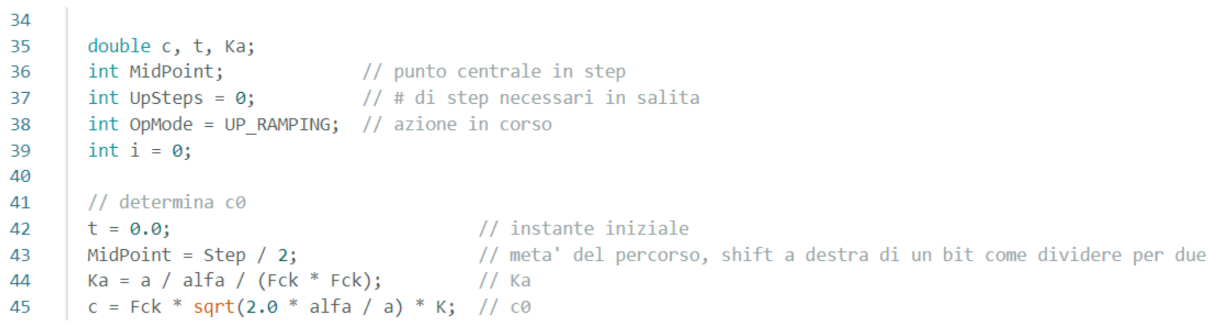
$$a = a_{lin} \cdot alfa \tag{3.4}$$

4. Fck: Pulse generator clock frequency, measured in Hertz (Hz).

5. K: Correction factor for $c_0$, dimensionless.

6. Step: Total number of steps, dimensionless, in this example is 800 so one revolution of the motor.

7. Cmin: Minimum amplitude of stepper pulses, dimensionless, serves as the minimum amplitude of the stepper motor pulses. When the motor speed reaches a certain point, the intervals between pulses become very small, indicating that the maximum speed has been reached. At this point, the Cmin variable is used to set a lower limit on the pulse amplitude, preventing further speed increases. This ensures that the motor maintains the desired speed without further escalation.



```

double c, t, Ka;
int MidPoint;             // punto centrale in step
int UpSteps = 0;          // # di step necessari in salita
int OpMode = UP_RAMPING;  // azione in corso
int i = 0;

// determina c0
t = 0.0;                             // instante iniziale
MidPoint = Step / 2;                 // meta' del percorso, shift a destra di un bit come dividere per due
Ka = a / alfa / (Fck * Fck);         // Ka
c = Fck * sqrt(2.0 * alfa / a) * K;  // c0
```

**Figure 3.15:** Input variable definition (2) and $c_0$ starting coefficient calculation

1. c: Variable used to store the time interval between stepper motor pulses, which determines the speed of the motor. Its value is dynamically updated during the execution of the algorithm.

2. t: Variable representing the current time in the algorithm, initially set to 0.0.

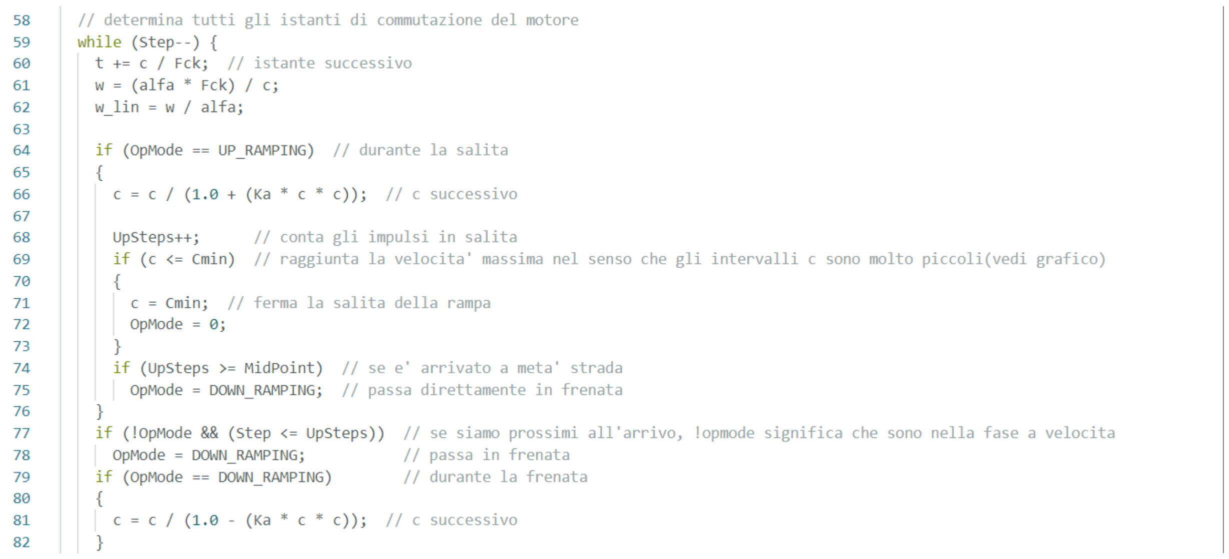
3. Ka: Variable representing a coefficient used in the calculation of c. It is computed based on the acceleration (a), motor step size (alfa), and pulse generator clock frequency (Fck).

4. MidPoint: Variable representing the midpoint of the total number of steps (Step). It is calculated by dividing Step by 2.

5. UpSteps: Variable used to count the number of steps taken during the upward ramping phase of the motor.

6. OpMode: Variable representing the current operation mode of the motor. It is initially set to $UP_{RAMPING}$. i: Variable used as an index counter for the loop iterations

In Figure 3.16 the actual implementation of the algorithm in C language to be uploaded and test through the Arduino board



```c
// determina tutti gli istanti di commutazione del motore
while (Step--) {
  t += c / Fck;  // istante successivo
  w = (alfa * Fck) / c;
  w_lin = w / alfa;

  if (OpMode == UP_RAMPING)  // durante la salita
  {
    c = c / (1.0 + (Ka * c * c));  // c successivo

    UpSteps++;      // conta gli impulsi in salita
    if (c <= Cmin)  // raggiunta la velocita' massima nel senso che gli intervalli c sono molto piccoli(vedi grafico)
    {
      c = Cmin;  // ferma la salita della rampa
      OpMode = 0;
    }
    if (UpSteps >= MidPoint)  // se e' arrivato a meta' strada
      OpMode = DOWN_RAMPING;  // passa direttamente in frenata
  }
  if (!OpMode && (Step <= UpSteps))  // se siamo prossimi all'arrivo, !opmode significa che sono nella fase a velocita
    OpMode = DOWN_RAMPING;           // passa in frenata
  if (OpMode == DOWN_RAMPING)        // durante la frenata
  {
    c = c / (1.0 - (Ka * c * c));  // c successivo
  }
```

**Figure 3.16:** Algorithm implementation

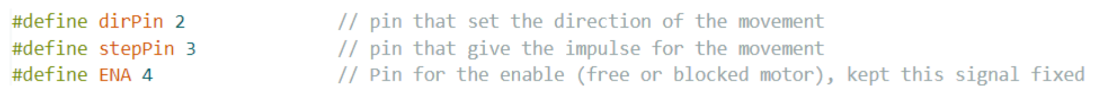
The while loop iterates through each step of the motor motion, decrementing the Step variable until it reaches zero. Within each iteration, the code calculates the next instance of switching for the motor.
Calculating Time and Speed: The variable t is updated to represent the next time instance based on the current time t and the time interval c between motor pulses. Additionally, the angular speed w and linear

speed $w_{lin}$ of the motor are calculated based on the motor step size alfa and the time interval c.
During the upward ramping phase (OpMode == $UP_{RAMPING}$) the time interval c is recalculated to gradually decrease as the motor accelerates. This calculation is based on the current value of c and the coefficient Ka. The variable UpSteps is incremented to count the number of steps taken during the upward ramping phase. If the time interval c becomes less than or equal to the minimum pulse amplitude Cmin, indicating that the maximum speed has been reached, the upward ramping phase is stopped by setting c to Cmin and switching the operation mode to $DOWN_{RAMPING}$. If the number of steps taken (UpSteps) exceeds the midpoint of the total steps (MidPoint), the operation mode is switched directly to the downward ramping phase ($DOWN_{RAMPING}$). If the operation mode is not upward ramping (!OpMode) and the current step is less than or equal to the number of steps taken during the upward ramping phase (Step <= UpSteps), the operation mode is switched to downward ramping ($DOWN_{RAMPING}$).
During the downward ramping phase (OpMode == $DOWN_{RAMPING}$), the time interval c is recalculated to gradually decrease as the motor decelerates. This calculation is similar to that of the upward ramping phase but with a negative acceleration coefficient. This code essentially implements a control algorithm for the stepper motor, adjusting the time intervals between motor pulses to achieve the desired speed profile, including acceleration, constant speed, and deceleration phases.

### Pulse generator clock frequency evaluation

Since the clock frequency of the pulse generator is one of the required input parameters but was not known a priori while working with an Arduino, an additional Arduino code for motor control was generated solely for the purpose of evaluating the value of $F_{CK}$ and determining the appropriate value to use. This code does not use the "AccelStepper.h" library but is a "low level approach" in which the motor moves by

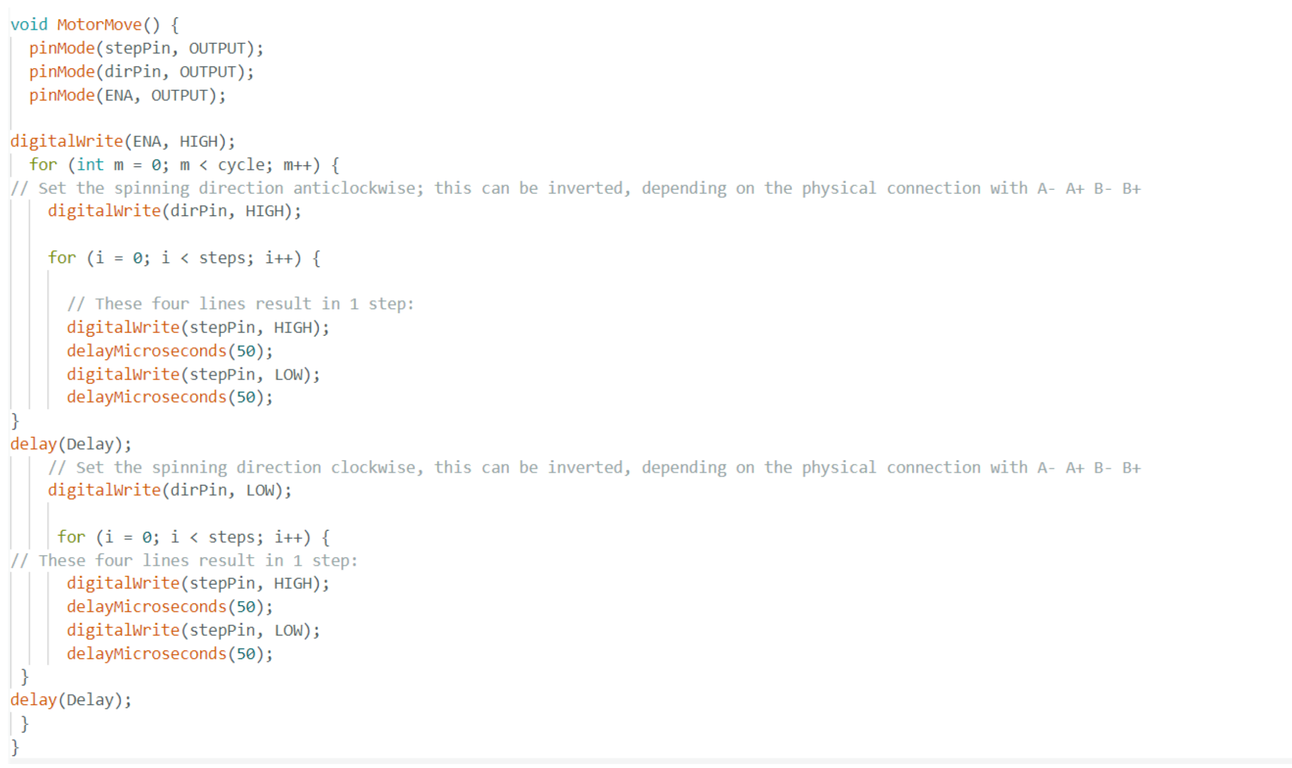
directly toggling the step pin for a certain number of times equal to the required number of step, hence this code provides basic step-by-step control without acceleration or deceleration. By directly acting through the "digitalWRITE" function on the three pins, 2,3 and 4, of the Arduino that are connected to the motor driver TB6600, respectively to the DIR+, PUL+ and ENA+, is it possible to control the motor and evaluate which is the lower limit of the pulse frequency at which the motor moves.
Firstly, the pins are defined according to the connections between Arduino and the motor driver TB6600.



```
#define dirPin 2              // pin that set the direction of the movement
#define stepPin 3             // pin that give the impulse for the movement
#define ENA 4                 // Pin for the enable (free or blocked motor), kept this signal fixed
```

**Figure 3.17:** TB6600 - Arduino PINs declaration

looking at Figure 3.18, the ENA pin is set to HIGH to enable the motor driver, the outer loop controls the number of cycles the motor will complete. After what the inner loop controls the number of steps the motor will take in the specified direction, the first inner loop moves the motor clockwise by the digitalWrite(dirPin, HIGH) function, while the second inner loop moves the motor anti-clockwise by the digitalWrite(dirPin, LOW) function. The four lines digitalWrite(stepPin, HIGH), delayMicroseconds(50), digitalWrite(stepPin, LOW), delayMicroseconds(50) sequence the steps, toggling the step pin to make the motor move.



```
void MotorMove() {
  pinMode(stepPin, OUTPUT);
  pinMode(dirPin, OUTPUT);
  pinMode(ENA, OUTPUT);

digitalWrite(ENA, HIGH);
  for (int m = 0; m < cycle; m++) {
// Set the spinning direction anticlockwise; this can be inverted, depending on the physical connection with A- A+ B- B+
    digitalWrite(dirPin, HIGH);

    for (i = 0; i < steps; i++) {

      // These four lines result in 1 step:
      digitalWrite(stepPin, HIGH);
      delayMicroseconds(50);
      digitalWrite(stepPin, LOW);
      delayMicroseconds(50);
}
delay(Delay);
    // Set the spinning direction clockwise, this can be inverted, depending on the physical connection with A- A+ B- B+
    digitalWrite(dirPin, LOW);

     for (i = 0; i < steps; i++) {
// These four lines result in 1 step:
      digitalWrite(stepPin, HIGH);
      delayMicroseconds(50);
      digitalWrite(stepPin, LOW);
      delayMicroseconds(50);
 }
delay(Delay);
 }
}
```

**Figure 3.18:** Implemented C code to move the motor by directly toggling the stepPin 3

By changing the delayMicroseconds value and hence the rate at which the pin is toggled the clock frequency of the pulse generator has been evaluated.
In particular with 50 $\mu$s the motor move fine while going below the frequency at which the steps are given is to high and the motor does not move. Hence 50 $\mu$s is taken as reference and a frequency of

$$F_{CK} = \frac{1}{50 \cdot 10^-6} = 20kHz \tag{3.5}$$

is used

## 3.4 Labview Code implementation

LabVIEW, short for Laboratory Virtual Instrument Engineering Workbench, is a graphical programming environment developed by National Instruments and uses a graphical programming language, called "G". It's widely used in engineering and scientific research for creating control systems and data acquisition applications. LabVIEW enables users to create programs through an intuitive graphical interface, where operations are executed by connecting graphical blocks called "virtual instruments" (VI). These VIs can be configured to control hardware devices, acquire data from sensors and analyze signals. LabVIEW is constitued by two foundational components within the graphical programming environment employed for application development, the front panel and the block diagram. The block diagram serves as the central canvas where the program's logic and functionality are constructed using graphical icons and interconnections. It encapsulates the data flow and underlying logic of the program. Throughout development, various nodes, functions, structures, and controls/indicators are placed onto the block diagram and interconnected via wires to craft the desired behavior. This diagrammatic representation facilitates the visualization and comprehension of the program's operational flow. On the other hand, the front panel functions as the user interface, providing an intuitive means for users to interact with the program. It hosts a graphical representation of user interface elements such as buttons, sliders, graphs, and numeric displays. Controls are utilized for user input, parameter adjustment, while indicators display outputs and results during program execution. The design of the front panel is pivotal in ensuring user-friendliness and accessibility of the application. Notably, the front panel and block diagram are closely linked, with changes made to controls/indicators on the front panel directly impacting the behavior of nodes on the block diagram, and vice versa. Due to its flexibility and wide range of functionalities, LabVIEW is employed across various sectors including industrial automation, biomedical engineering, academic research, and test and measurement system development. In this research, LabVIEW played a pivotal role as a versatile and powerful tool for system control and data acquisition and it provided the framework for integrating and controlling both an Arduino microcontroller and a linear stage in the experimental setup. Simultaneously, LabVIEW facilitated communication with an LCR meter, enabling precise measurement and analysis. By leveraging LabVIEW's capabilities, the integration between different components and instruments has been possible , automating and controlling the experimental stretching setup to test the electrodes stretchability. In constructing the setup, the aim was to utilize LabVIEW to create a user-friendly interface for establishing the connection and configuring parameters of the LCR meter, while also setting parameters for stretching the DUT. Communication between the PC and the LCR meter, as well as between the PC and Arduino, occurs independently via serial communication. Once all stretching parameters are configured and the serial connection with Arduino is established, these parameters are sent via serial to the board, acquired, and properly assigned to variables within the Arduino code. Subsequently, Arduino is responsible for moving the stage and performing the stretching operation. In addition to parameter transmission, LabVIEW is concurrently employed for data acquisition and plotting from both the LCR meter regarding impedance and Arduino regarding stage position. The operational flow implemented by the LabVIEW code to execute the tasks previously described can be hence divided in six different macro step that are reported and briefly explained in Figure 3.19. The first thing that has to be done is to set the serial connection with the Arduino board and LCR meter E4980A. After what the parameter for the stretching sequence must be defined, those parameters are seven : a command in char format, acceleration in steps/$s^2$, speed in step/s, cycle e.g. the number of iteration of the stretching sequence, delay ,e.g the time that the stage has to remain blocked at the final and at the initial position

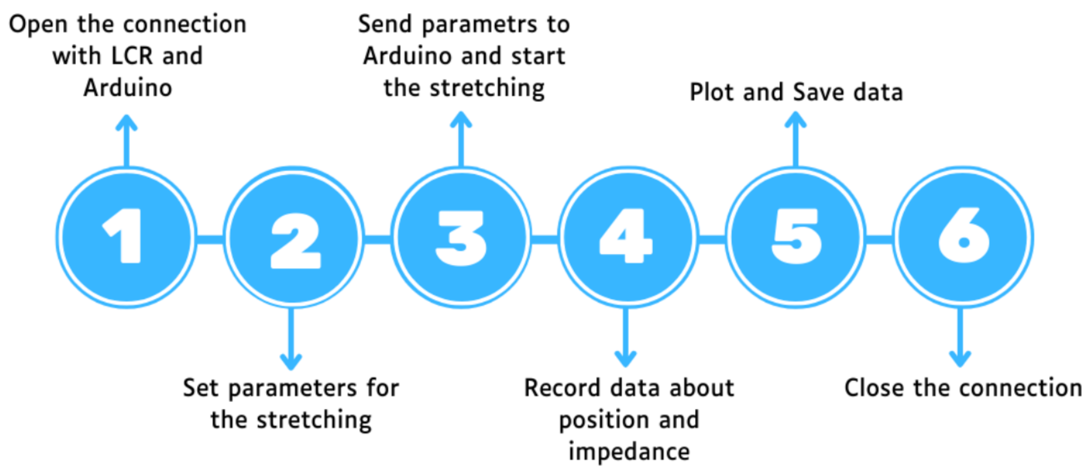

Open the connection with LCR and Arduino
Send parametrs to Arduino and start the stretching
Plot and Save data
1
2
3
4
5
6
Set parameters for the stretching
Record data about position and impedance
Close the connection

**Figure 3.19:** Operational workflow to implement the stretching sequence

and the resolution , e.g. the number of data points acquired by the LCR meter and hence about the position of the stage while travelling. The third step is about concatenating these seven parameters in an unique string, send the string to the Arduino and start the stretching of the sample. As soon as the third step is completed the fourth step can start, the data are recorded both from the LCR meter and from the Arduino, plotted separately and saved in an excel file. In conclusion the connection with the instruments is closed and the LabVIEW program stopped. In the next pages a more comprehensive description is given and both the LabVIEW and Arduino code are showed and explained.

### 3.4.1 STEP 1 : Connection aperture

As already mentioned the very first thing to do is to open the serial communication between the PC and the instrumentation, respectively the LCR meter A4980e and the Arduino Board.

To properly interface with the LCR meter the LabVIEW driver blocks for the LCR meter E4980A have been used, these driver blocks

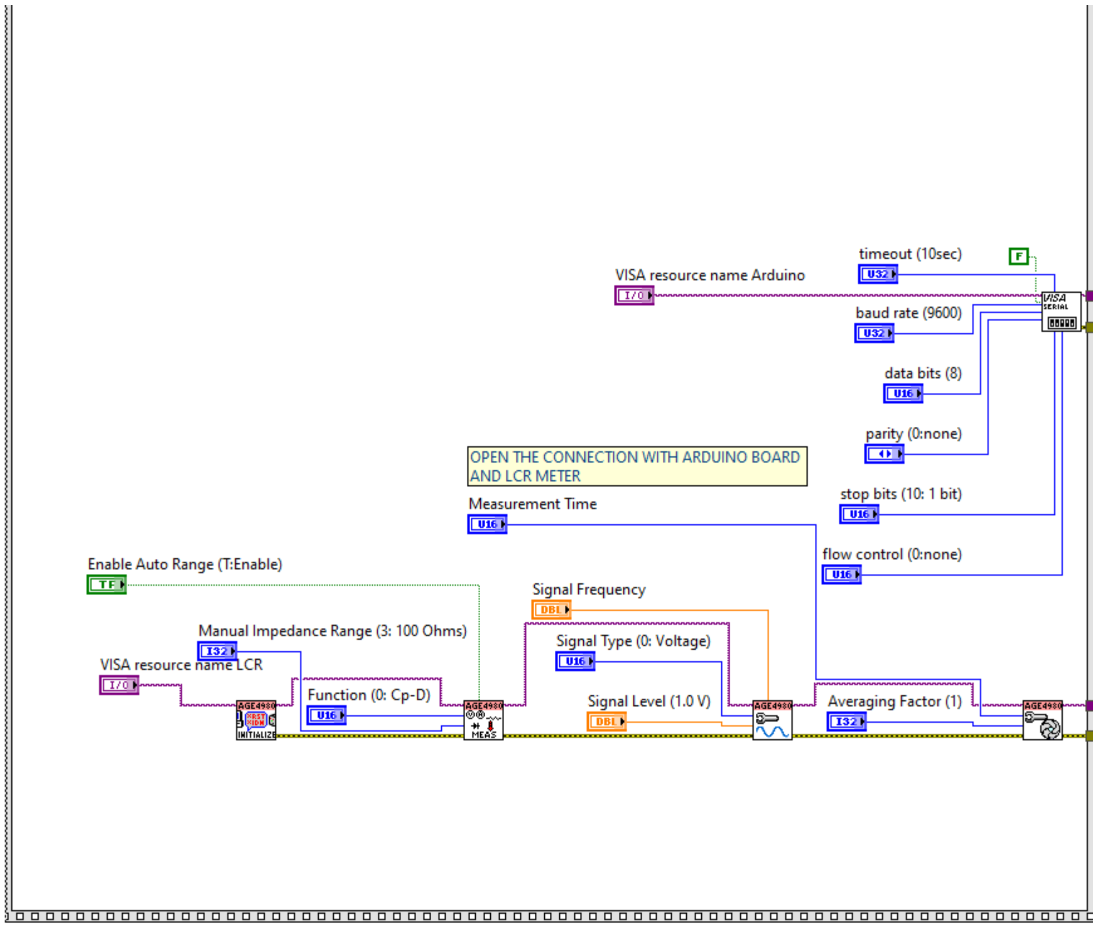


**Figure 3.20:** LabVIEW code to open the connection with the Arduino and the LCR meter

provide users with the flexibility to tailor the measurement settings, signal characteristics, and integration parameters of the LCR meter to meet the specific requirements of the measurement application. These drivers are based on VISA (Virtual Instrument Software Architecture), which is a software library and set of APIs (Application Programming Interfaces) provided by National Instruments for instrument control and communication. It allows LabVIEW programs to communicate with various instruments, such as oscilloscopes, multimeters, function generators, and others, over different communication interfaces like

GPIB, USB, Ethernet, and serial ports. The first one, used to establish the communication is the "initialize.vi", which gets as input only the VISA resource name. The "VISA resource name" (Virtual Instrument Software Architecture) is a unique identifier used to communicate with hardware instruments, essentially, it is an address that specifies the location and type of instrument you want to communicate with. The next three blocks, from left to right, are the one actually used to set the parameters for the measurement.

1. Configure Measurement: This block is used to set up the specific measurement parameters such as the impedance range, which can be also be imposed as AUTO with the proper control, and the function, e.g. the pair of parameters (primary and secondary) that the LCR has to measure in order perform different measurement (impedance, capacitance, inductance, or resistance)

2. Configure Measurement Signal: This block enables users to specify additional parameters related to the measurement signal applied itself, such as the signal frequency, amplitude, and type. It allows users to customize the characteristics of the signal that the LCR meter will apply to the DUT during the measurement process.

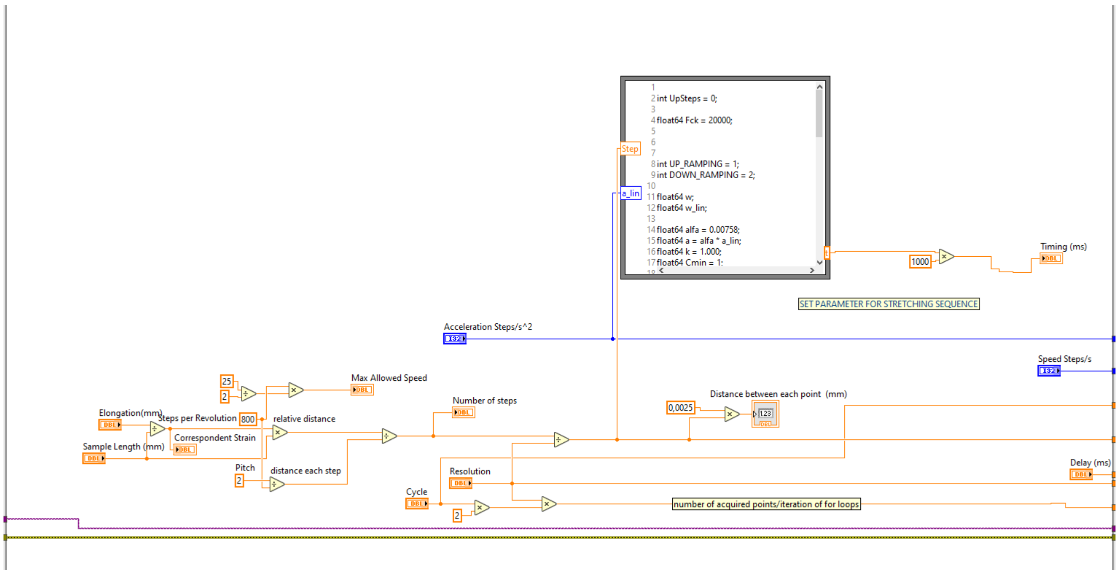
3. Configure Aperture: The configure aperture block is used to adjust the integration time or aperture time of the measurement. This parameter determines the duration over which the LCR meter will integrate the measured signal to obtain accurate results. By configuring the aperture, users can optimize the measurement resolution and accuracy based on the characteristics of the DUT and the desired measurement precision.

Moving to the Arduino the connection was still established using VISA blocks, notwithstanding the absence of drivers; instead, the original VISA blocks were utilized. Indeed through the block VISA Configure serial port is it possible to define the following parameters:

1. Visa resource name for the Arduino
2. timeout = 10 s, refers to the duration for which the system waits for a response before considering the communication attempt unsuccessful.
3. baud rate = 9600, the rate at which data is transferred over the serial connection, measured in bits per second (bps). It defines the speed at which data is transmitted and received between LabVIEW and Arduino.
4. data bits = 8, the number of bits used for each character in the serial communication.
5. parity = none, is a method used to check the integrity of transmitted data
6. stop bits = 1, indicate the end of a data frame in serial communication.
7. flow control = none, refers to the mechanism used to regulate the flow of data between the transmitting and receiving devices

### 3.4.2 STEP 2 : parameters setting

The second step in the stretching sequence flow is about the parameters setting to actually implement the stretching , so to make the motor move following a precise "behaviour". What is done here is the definition of all the six parameters necessary to fully describe the movement of the stage. These parameters are : acceleration in steps/$s^2$, speed in step/s, cycle e.g. the number of iteration of the stretching sequence, delay ,e.g the resting time that the stage has to remain blocked at the final and at the initial position and the resolution , e.g. the number of data points acquired from the LCR meter and from the position of the stage while travelling. The Figure 3.21 shows how the different parameters are computed within the LabVIEW environment.



```

int UpSteps = 0;

float64 Fck = 20000;



int UP_RAMPING = 1;
int DOWN_RAMPING = 2;

float64 w;
float64 w_lin;

float64 alfa = 0.00758;
float64 a = alfa * a_lin;
float64 k = 1.000;
float64 Cmin = 1;
```

**Figure 3.21:** Parameters definition in LabVIEW

The user must define the sample length [mm] and the desired elongation to apply to the sample in mm, from these two controls the code returns the correspondent strain $\epsilon_x$.

$$\epsilon_x = \frac{\Delta x}{l} \tag{3.6}$$

where $\epsilon_x$ is the applied strain, Δx is the desired elongation and $l$ is the length of the sample. Those parameter are expressed in mm but what the motor needs as input to move is the number of step correspondent to the given elongation. To satisfy this requirement firstly the distance correspondent to each step is computed by dividing the lead screw pitch of 2 mm (value from data sheet , the "lead screw pitch" in a linear stage refers to the longitudinal distance traveled by the linear stage for each complete rotation of the lead screw) by the number of step per revolution, in that case 800.

$$Distance_{step} = \frac{2\,\mathrm{mm}}{800steps_{revolution}} = \frac{0.0025mm}{step} \tag{3.7}$$

by dividing the elongation $\Delta_x$ by the Distance each step the total number of step can be obtained.

$$TOT_{steps} = \frac{\Delta x}{Distance_{step}} \tag{3.8}$$

The next two parameters that has to be defined through the LabVIEW GUI are the number of Cycle and the Resolution. The Resolution quantity is crucial because is used to divide the total number of step computed before in sub intervals of steps that the motor has to do

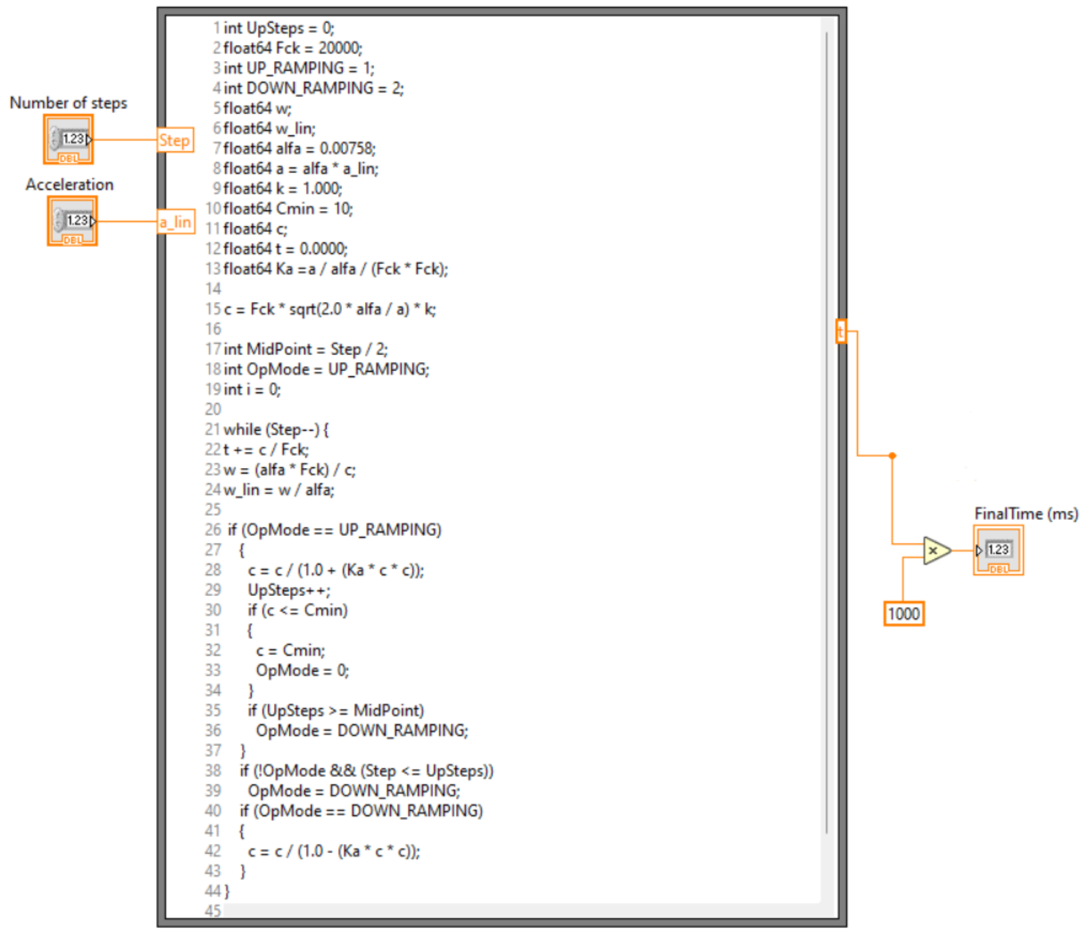
$$step = \frac{TOT_{steps}}{Resolution} \tag{3.9}$$

So for example if the motor has to move by 1000 steps and the resolution parameter is 10 the motor will do 100 steps 10 times until it reach the final position of 1000 steps. The resulting value of step is the one actually sent to the Arduino with the Resolution one and with the Cycle value, the working principle of these variables inside the Arduino code will be explained later. The last three inputs that the user has to defined to start the stretching are the Delay (ms), the speed (step/s) and the acceleration (steps/$s^2$), no data manipulation occur on those variables.

### Formula node

In step 2 also the travelling time taken by the stage to cover the required number of steps is computed thanks to the algorithm already describer in the subsection 3.3.1. Here the algorithm has been adapted to the LabVIEW paradigm through the Formula Node structure. In LabVIEW, the Formula Node structure allows users to create custom mathematical expressions using a syntax similar to traditional programming languages, it is a graphical programming element that allows to write mathematical or logical equations directly within a LabVIEW block diagram. The Formula Node structure accept some inputs and after the execution of the code inside of it return the wanted outputs.

In that case the two inputs are the Acceleration of the motor and the number of steps to be executed, the code will return as outputs the a value correspondent to the time (called FinalTime) taken by the motor to complete the number of step. This time is multiplied by 1000 to bring it in ms and then will be use in STEP 4 and STEP 5 to give synchronization to the LabVIEW-Arduino communication by defining the time between each iteration of the for loop used to acquire and plots the incoming data.



```
int UpSteps = 0;
float64 Fck = 20000;
int UP_RAMPING = 1;
int DOWN_RAMPING = 2;
float64 w;
float64 w_lin;
float64 alfa = 0.00758;
float64 a = alfa * a_lin;
float64 k = 1.000;
float64 Cmin = 10;
float64 c;
float64 t = 0.0000;
float64 Ka =a / alfa / (Fck * Fck);

c = Fck * sqrt(2.0 * alfa / a) * k;

int MidPoint = Step / 2;
int OpMode = UP_RAMPING;
int i = 0;

while (Step--) {
t += c / Fck;
w = (alfa * Fck) / c;
w_lin = w / alfa;

 if (OpMode == UP_RAMPING)
  {
    c = c / (1.0 + (Ka * c * c));
    UpSteps++;
    if (c <= Cmin)
    {
      c = Cmin;
      OpMode = 0;
    }
    if (UpSteps >= MidPoint)
      OpMode = DOWN_RAMPING;
  }
  if (!OpMode && (Step <= UpSteps))
    OpMode = DOWN_RAMPING;
  if (OpMode == DOWN_RAMPING)
  {
    c = c / (1.0 - (Ka * c * c));
  }
}
```

**Figure 3.22:** Implemented formula node to adapt the stepper motor speed profile algorithm in LabVIEW

### 3.4.3 STEP 3 : parameters concatenation and sequence start

In this section the six parameters (step, resolution, delay, acceleration, speed, cycle) are provided as input to "number to decimal string" blocks. The LabVIEW block "Number to Decimal String" is used to convert numerical data, such as integers or floating-point numbers, into a string format that represents the number in decimal notation. This block is particularly useful when it's needed to display numerical data on a user interface, save it to a file, or transmit it over a communication interface that requires string data. In Figure 3.23 the LabVIEW code used to concatenate the parameters and create the string, as well as the blocks used to send the string to the Arduino to start the stretching are showed. Once the data has been converted into string format (pink wires), they are inputted into the "concatenate string" block. Through this block, each parameter is concatenated with the others into a single string. The string always begins with a letter (command), followed by the values of the six aforementioned parameters, each separated by a comma. The formed string now moves towards the next block, namely "Visa Write", which writes the data from the write buffer to the device specified by the Visa resource name. Since when the string enters the input of Visa Write, the data is automatically sent to the Arduino, causing the motor to move, the block has been incorporated into a more complex structure, a case structure, which allows the string to enter the write block only when the boolean control (green) 'start stretching' is pressed from the GUI. In fact, when the 'start stretching' button is pressed, it assumes a logical value of 1, causing the case to enter the true case and thus the string into the Visa Write block. This approach provides the user with greater control over the system's operation. Furthermore, the entire case structure is embedded within an additional structure, a while loop. This allows the user to press the stretching button at any time. If the button is not pressed, the boolean

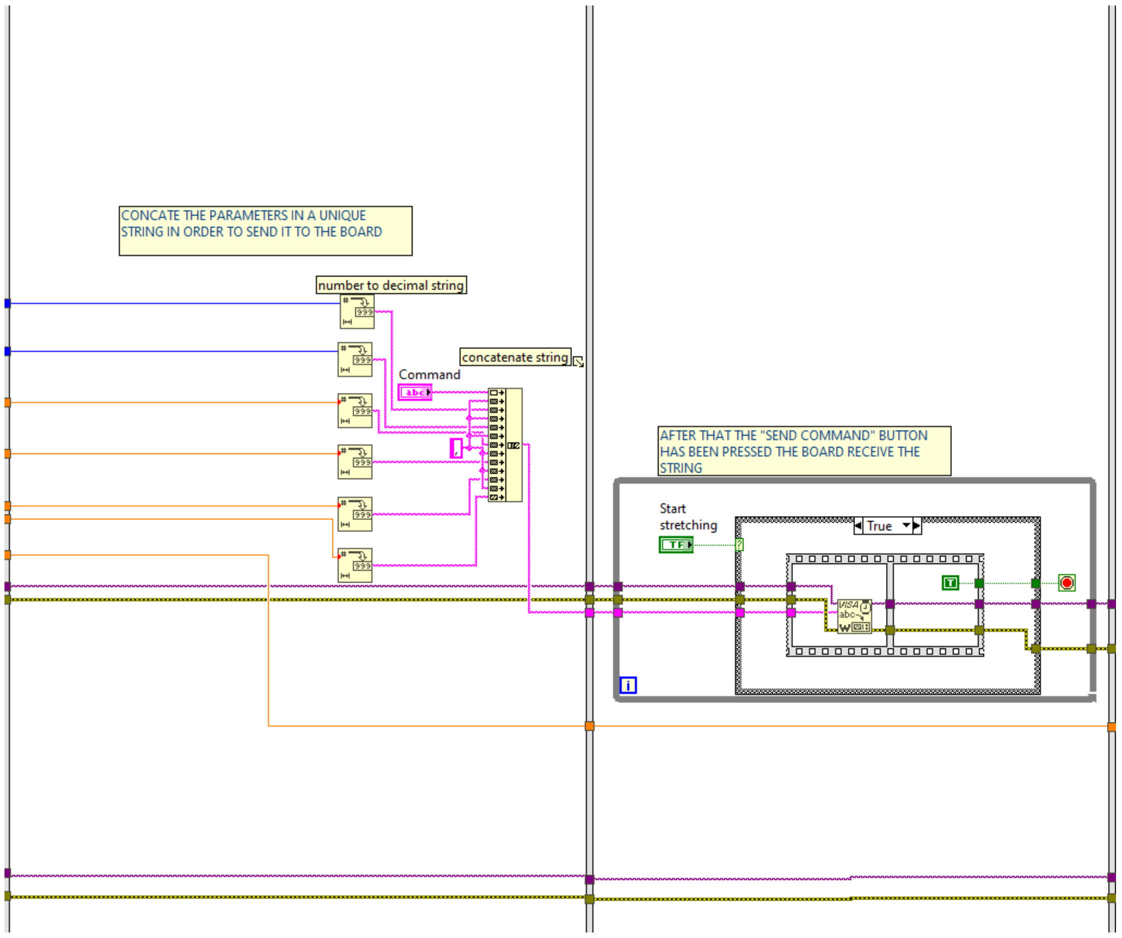
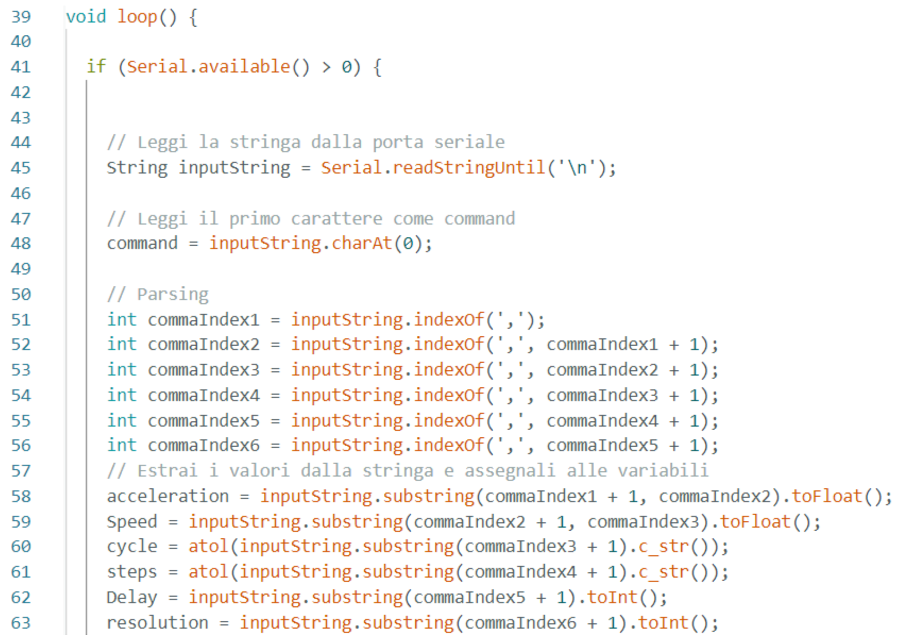

CONCATE THE PARAMETERS IN A UNIQUE STRING IN ORDER TO SEND IT TO THE BOARD
number to decimal string
concatenate string
Command
AFTER THAT THE "SEND COMMAND" BUTTON HAS BEEN PRESSED THE BOARD RECEIVE THE STRING
Start stretching
True

**Figure 3.23:** Parameters string creation, sequentially the string is sent to Arduino through the "Visa write" block.

control value will be 0, causing the case to enter the 'false' case and the while loop to continue iterating indefinitely (unless the LabVIEW program is stopped), effectively placing the system in a state of waiting for an 'external' action from the user via the graphical interface.

### 3.4.4 Arduino code Implementation

In this section the code that orchestrates the precise movement sequence of the stepper motor, including forward and backward movements, as well as pauses between cycles, is analyzed. This code leads to achieve the desired stretching behavior through careful control of the loop iterations and motor commands, it ensures the smooth and controlled execution of the stretching sequence.
The section of code in Figure 3.24 is responsible for the parsing of the incoming string, sent via the serial port, and assigning the extracted values to specific variables. Parsing is the process of analyzing a string to identify and extract structured information within it. In this context, parsing the serial string involves analyzing the string to identify the positions of commas and extracting the corresponding numerical values to be used in the program.

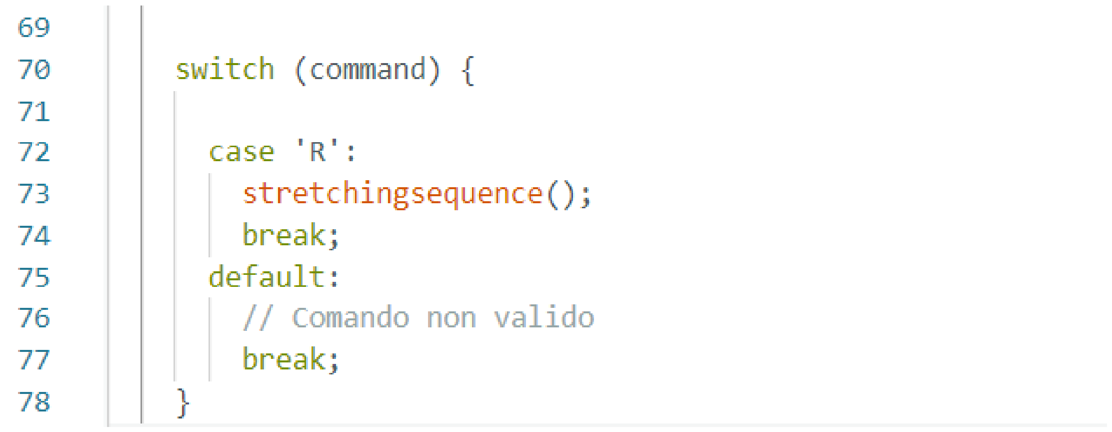


```
void loop() {

  if (Serial.available() > 0) {


    // Leggi la stringa dalla porta seriale
    String inputString = Serial.readStringUntil('\n');

    // Leggi il primo carattere come command
    command = inputString.charAt(0);

    // Parsing
    int commaIndex1 = inputString.indexOf(',');
    int commaIndex2 = inputString.indexOf(',', commaIndex1 + 1);
    int commaIndex3 = inputString.indexOf(',', commaIndex2 + 1);
    int commaIndex4 = inputString.indexOf(',', commaIndex3 + 1);
    int commaIndex5 = inputString.indexOf(',', commaIndex4 + 1);
    int commaIndex6 = inputString.indexOf(',', commaIndex5 + 1);
    // Estrai i valori dalla stringa e assegnali alle variabili
    acceleration = inputString.substring(commaIndex1 + 1, commaIndex2).toFloat();
    Speed = inputString.substring(commaIndex2 + 1, commaIndex3).toFloat();
    cycle = atol(inputString.substring(commaIndex3 + 1).c_str());
    steps = atol(inputString.substring(commaIndex4 + 1).c_str());
    Delay = inputString.substring(commaIndex5 + 1).toInt();
    resolution = inputString.substring(commaIndex6 + 1).toInt();
```

**Figure 3.24:** Read from serial and parsing of the string

In detail, the code performs the following:
The "indexOf" method is used to find the positions of commas within

the inputString. This method returns the index of the first occurrence of the specified character within the string. Variables commaIndex1, commaIndex2, commaIndex3, commaIndex4, commaIndex5, and commaIndex6 represent the positions of commas within the string. Using the values obtained from the comma positions, the code uses the "substring" method to extract substrings from the inputString. These substrings contain the values of acceleration, speed, cycle, steps, delay, and resolution parameters. The extracted values are then converted to the appropriate data types using "toFloat" for floating-point values and "toInt" for integer values, and assigned to the acceleration, Speed, cycle, steps, Delay, and resolution variables.
Once that the string has been parsed the code utilizes a switch statement to determine the action to take based on the command received. In this case, if the command character is 'R', it proceeds to execute the "stretchingsequence" function that contains the necessary logic for performing a stretching sequence. If the command character does not match 'R', indicating an unrecognized or invalid command, the code does not execute any specific action and continues with the loop.



```

    switch (command) {

      case 'R':
        stretchingsequence();
        break;
      default:
        // Comando non valido
        break;
    }
```

**Figure 3.25:** Driver Board connection

Moving forward the code enters in the "strecthingsequence" function. Here an unsigned long variable named "starttime" is initialized to track

the starting time of the sequence. Next, the code calculates the "finalposition" by multiplying the number of steps by the resolution. This variable signifies the ultimate position that the stepper motor needs to reach during the stretching sequence.
After what, the code enables the outputs of the stepper motor using the "stepper.enableOutputs" function call. This action ensures that the motor is ready to receive commands and execute movements. Subsequently, the current position of the stepper motor is set to 0 via the "stepper.setCurrentPosition" function. This step effectively resets the motor's position to its initial state, preparing it for the upcoming sequence.
Following this, the code configures the speed of the stepper motor to the specified value using the "stepper.setSpeed(Speed)" function. This adjustment ensures that the motor moves at the desired velocity during the stretching process. Additionally, the acceleration of the stepper motor is set to the specified value using the "stepper.setAcceleration(acceleration)" function. By configuring the acceleration, the code ensures smooth and controlled movements of the motor as it transitions between different speeds.

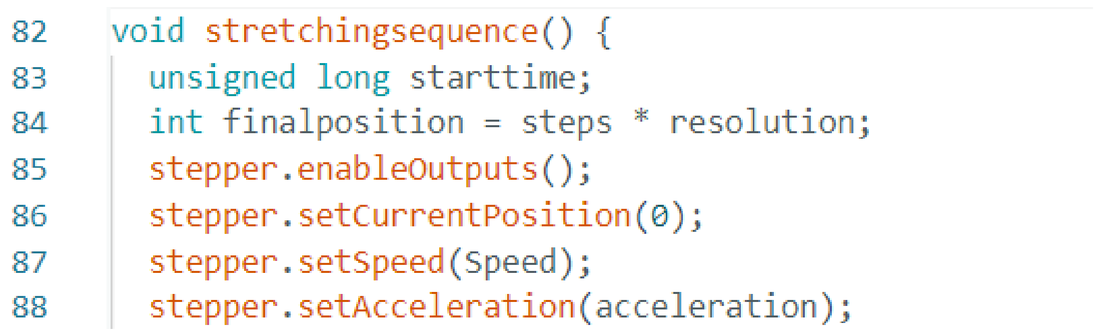
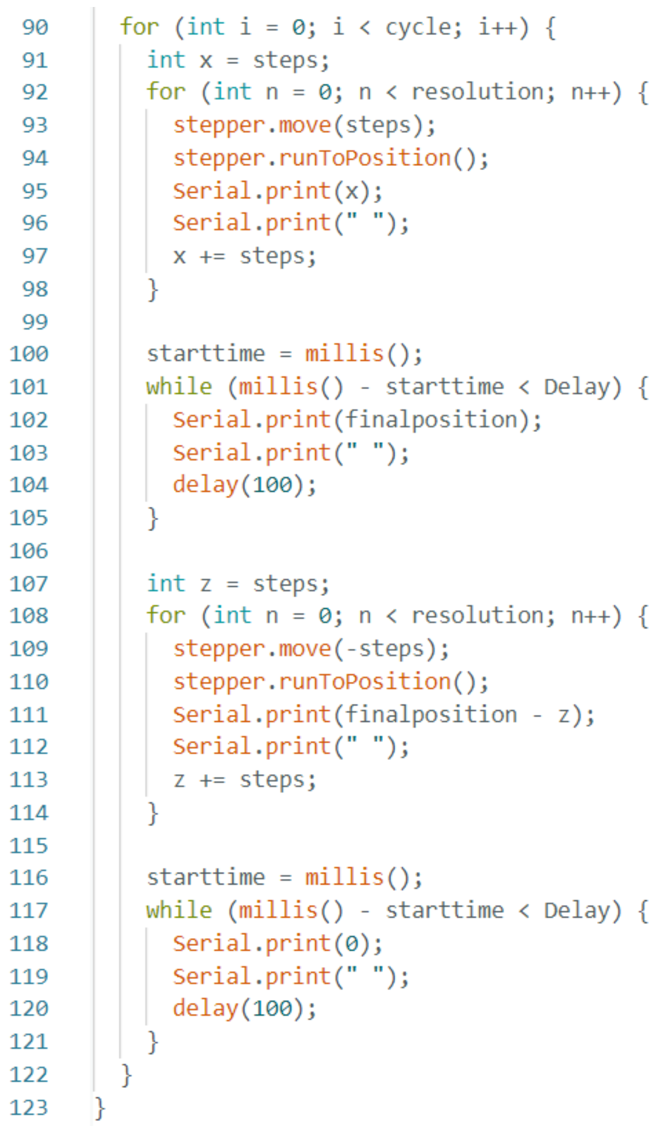

```
void stretchingsequence() {
  unsigned long starttime;
  int finalposition = steps * resolution;
  stepper.enableOutputs();
  stepper.setCurrentPosition(0);
  stepper.setSpeed(Speed);
  stepper.setAcceleration(acceleration);
```

**Figure 3.26:** Stepper motor settings

In conclusion the core of the function to control the motor is here reported in Figure 3.27. This segment of code contains a nested loop structure repeated for the specified number of cycles designed to properly control the motor and hence move the stage to finally stretch the DUT. Firstly, an outer loop iterates cycle times, representing the number of cycles for the stretching sequence. Within each cycle, an inner loop executes resolution times, representing the number of steps to be taken during each cycle. During each iteration of the inner loop, the stepper motor is commanded to move forward by a distance of steps using the "stepper.move(steps)" function. Subsequently, the "stepper.runToPosition" function ensures that the motor reaches the desired position before proceeding. The current position of the motor, represented by the variable x, is printed to the serial monitor using "Serial.print(x)", followed by a space character. The value of x is then incremented by steps to prepare for the next movement. After completing the forward movement sequence, a delay is introduced using the "millis" function to wait for the specified Delay duration. During this delay, the final position of the stepper motor (calculated as "finalposition") is repeatedly printed to the serial monitor, along with a space character. Following the delay, another inner loop is executed to perform the reverse movement sequence. Similar to the forward movement, the stepper motor is commanded to move backward by a distance of steps using "stepper.move(-steps)", followed by "stepper.runToPosition" to ensure precise positioning. The current position of the motor, represented by the value "finalposition - z", is printed to the serial monitor, along with a space character. The value of z is then incremented by steps to prepare for the next backward movement. Finally, another delay is introduced to wait for the specified Delay duration before proceeding to the next cycle. During this delay, the value 0 is repeatedly printed to the serial monitor, indicating the resting position of the stepper motor.



```
  for (int i = 0; i < cycle; i++) {
    int x = steps;
    for (int n = 0; n < resolution; n++) {
      stepper.move(steps);
      stepper.runToPosition();
      Serial.print(x);
      Serial.print(" ");
      x += steps;
    }

    starttime = millis();
    while (millis() - starttime < Delay) {
      Serial.print(finalposition);
      Serial.print(" ");
      delay(100);
    }

    int z = steps;
    for (int n = 0; n < resolution; n++) {
      stepper.move(-steps);
      stepper.runToPosition();
      Serial.print(finalposition - z);
      Serial.print(" ");
      z += steps;
    }

    starttime = millis();
    while (millis() - starttime < Delay) {
      Serial.print(0);
      Serial.print(" ");
      delay(100);
    }
  }
}
```

**Figure 3.27:** "stretchingsequence" function that controls the motor to implement the stretching.

### 3.4.5 STEP 4 : Data recording

Returning to the LabVIEW code, the next step is to accurately acquire real-time data from both the Arduino and the LCR meter. To achieve this, the task has been broken down using a structure comprised of multiple for loops and a queue system. In fact, the main challenge in these types of real-time data acquisition systems lies in synchronizing the various instruments and incoming data so that they are consistent with each other, as well as ensuring their correct sampling [19]. In other words, for every data point received from the Arduino regarding the position, there must correspond the respective impedance data acquired at the same instant. In figure are present two different rectangles that in LabVIEW are two for loop, the upper one is used to collect the serial data coming from the Arduino board about the position, while the bottom is used to record data about the impedance from the LCR meter. The two for loops start at the same time and are iterated for the same number of time since the control N in each loop has in input the same wire coming from STEP 2, so they also finish together. The number of iterations and hence the number of data acquisition is computed as follow

$$N_{iterations} = 2 \cdot Cycle \cdot Resolution \tag{3.10}$$

Furthermore, by creating a local variable, the timing value (in milliseconds) calculated through the formula node section 3.4.2 is passed as input to the wait function present in each of the for loops. This ensures that each iteration of the for loop occurs every Timing milliseconds. As a result, the data reception between LabVIEW and Arduino is synchronized, enabling correct sampling. Additionally, impedance data is collected at the same instant, ensuring correspondence between the impedance and position data. The data regarding position and impedance are extracted in the LabVIEW interface respectively through the VISA read blocks and the Read block from the Agilent E4980 library. Precisely, one data is sampled by those two block at each iteration of the the respective for loop. The former returns the position data via the output buffer (pink wire), while the latter simultaneously returns two outputs, the primary parameter and the secondary parameter, at each iteration of the loop.

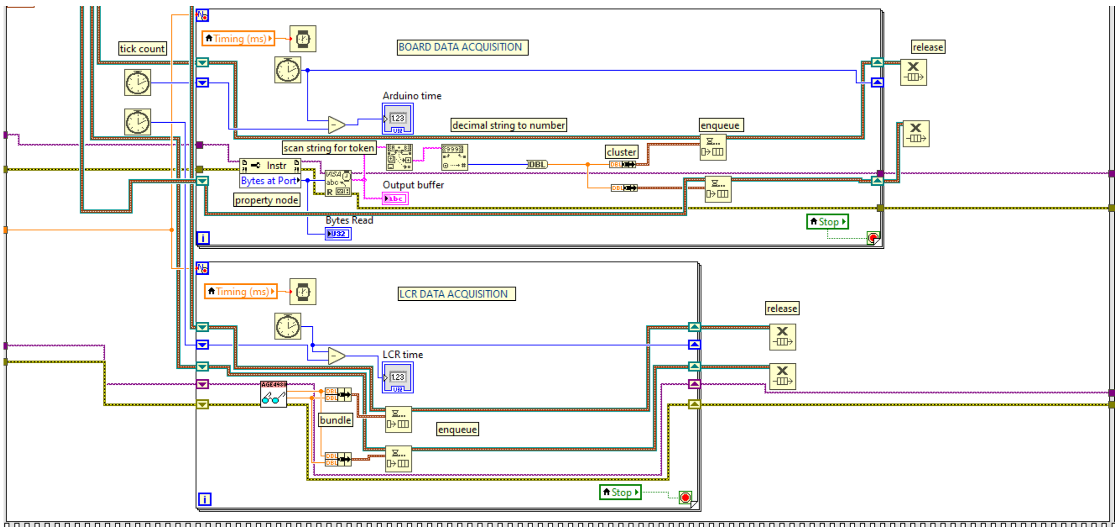


**Figure 3.28:** Acquisition loops, the bottom loop is used to read data form Arduino while the upper one to read data coming from the LCR meter

The decision to use multiple for loops instead of a single loop that encompasses all operations to be performed is tied to reducing computational burden. In fact, by employing multiple loops, it's possible to decrease the number of operations occurring within each loop, thereby reducing the time required to complete the instructions and operations within the loop. It's essential, indeed, for the data to be sampled correctly, that the operations carried out within each loop conclude in a shorter time than the one defined by the "timing (ms)" variable. Essentially, it's necessary for the operations performed in a for loop to conclude before a new data on the serial port arrives. Simultaneously dividing tasks and "breaking" the code into multiple for loops entails increased complexity in managing the flow of incoming data, as these, once sampled by their respective LabVIEW blocks, need to be "transported" to other for loops (described in the next section, Step 5), where they will be plotted and saved to Excel. To handle this problem and manage the data flow a system of queue has been used to make the different loops communicate and exchange the input data, indeed, "Queue" blocks in LabVIEW are fundamental components used for managing data flow within LabVIEW programs. They operate based on the principle of a First In, First Out (FIFO) queue, and they are particularly valuable in scenarios where data needs to be exchanged asynchronously or synchronized between different sections of the program One key advantage of "Queue" blocks is their ability to support asynchronous communication. This means that data can be passed between different sections of the program without requiring them to operate in lockstep. As a result, one part of the program can continue its operation without waiting for another part to finish processing the data. Additionally, "Queue" blocks provide a mechanism for synchronization, ensuring that data is transferred in a coordinated manner between different sections of the program, helping to prevent issues such as data loss or inconsistency and ensures that the program operates smoothly. Other benefit is that they help to decouple different components of a LabVIEW program leading to a greater flexibility and ease of development; furthermore their dynamic data handling capabilities let them automatically resize based on the number of elements they contain, eliminating the need for manual memory management and improving overall program efficiency. To the scope of this project the output data from the VISA Read and Agilent E4980 Read block are inputted in two different "Enqueue" block, where two different queue for each type of data are created. In this way there will be two distinct and identical streams of data output from each of the two loops, which will then be processed and utilized in the subsequent step.

### 3.4.6 STEP 5 : Plot and save the data

In Figure 3.29, two additional loops are depicted: the bottom one is utilized to perform the task of processing and plotting the data, while in the top one, the data is simply saved to an Excel file. The data reaches these two loops through the queue system, where the "Dequeue Element" block makes the data flow available to the instruction blocks and functions within the loops.

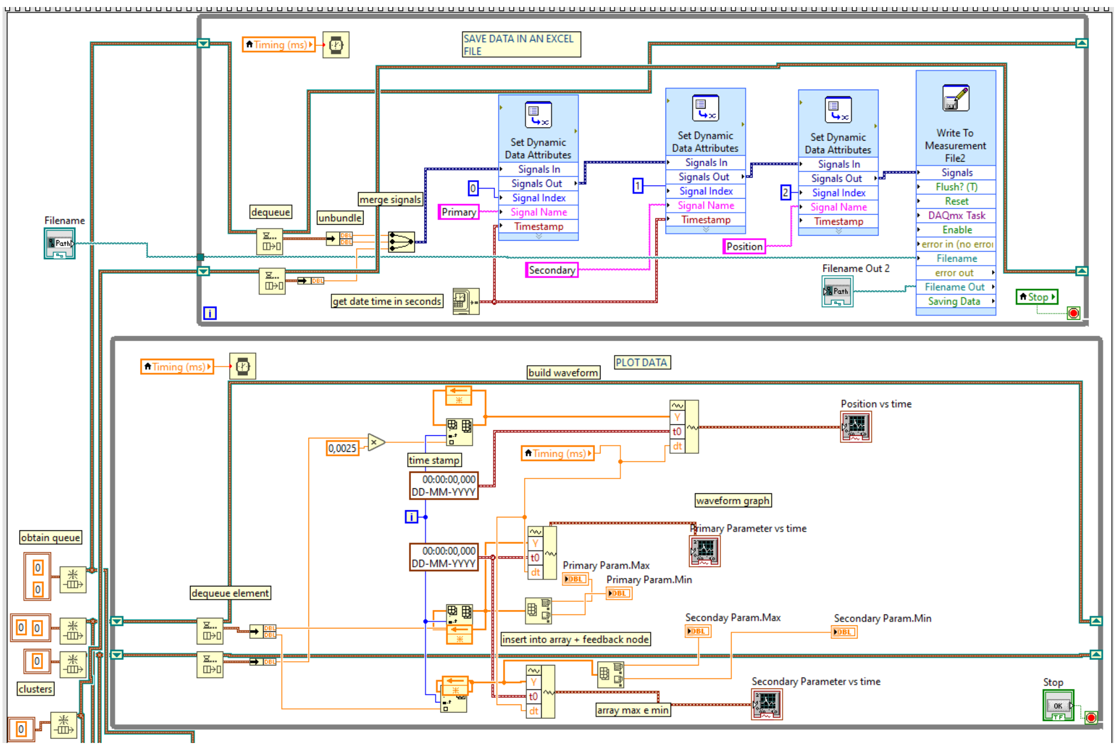


**Figure 3.29:** Elaboration loops, the bottom one is used to display the incoming data within the LabVIEW GUI, the upper one is used to save the same data in an excel file.

In the upper loop, the data output from the "Dequeue Element" blocks is fed into the "Write to Measurement File" block, used to generate an Excel spreadsheet where each row includes the date and time, primary parameter, secondary parameter, and position. In the lower loop, the data is input to three different "Waveform Graph" blocks, namely "Position vs Time," "Primary Parameter vs Time," and "Secondary Parameter vs Time," used for real-time visualization of the incoming data. To maintain synchronization with the description in Step 4, once again, the number of iterations N of the loops is the same as calculated in Equation 3.10, while each iteration of the loops occurs every "Timing (ms)" milliseconds (section 3.4.2).

## 3.4.7 STEP 6 : Close the connection

In conclusion, once the 4 aforementioned for loops have been completed, and thus all the data has been acquired and processed, the connection with the Arduino board and the LCR meter can be closed, and consequently, the execution of the LabVIEW program terminated. This is achieved using the two "VISA Close" blocks to close communication with the Arduino and the "Agilent E4980 Close" block to close communication with the LCR meter.

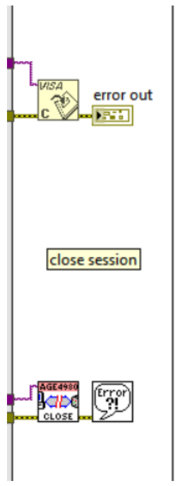
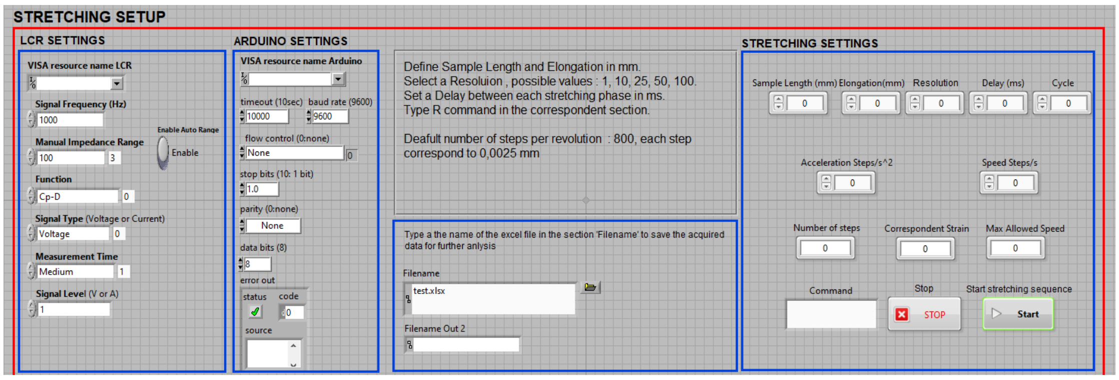


**Figure 3.30:** Connection close

Additionally, these two blocks allow any error messages that may have occurred during execution to be displayed via the graphical interface using the two "error out" indicators.

### 3.4.8 GUI : graphical user interface

In the following the resulting graphical user interface developed from the LabVIEW code is showed, this GUI has been used to stretch the flexible electrodes and to evaluate how their electrical parameter change in relation to the applied elongation. For simplicity, the GUI has been divided into two images, Figure 3.31 and Figure 3.32, however, the two images are to be imagined as one, seamlessly continuing from the other.



STRETCHING SETUP

LCR SETTINGS

VISA resource name LCR:
Signal Frequency (Hz): 1000
Manual Impedance Range: 100 | 3
Enable Auto Range: Enable
Function: Cp-D | 0
Signal Type (Voltage or Current): Voltage | 0
Measurement Time: Medium | 1
Signal Level (V or A): 1

ARDUINO SETTINGS

VISA resource name Arduino:
timeout (10sec): 10000
baud rate (9600): 9600
flow control (0:none): None | 0
stop bits (10: 1 bit): 1.0
parity (0:none): None
data bits (8): 8
error out: status ✓ | code 0 | source

Define Sample Length and Elongation in mm.
Select a Resoluion , possible values : 1, 10, 25, 50, 100.
Set a Delay between each stretching phase in ms.
Type R command in the correspondent section.

Deafult number of steps per revolution : 800, each step correspond to 0,0025 mm

Type a the name of the excel file in the section 'Filename' to save the acquired data for further anlysis

Filename: test.xlsx
Filename Out 2:

STRETCHING SETTINGS

| Sample Length (mm) | Elongation(mm) | Resolution | Delay (ms) | Cycle |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |

Acceleration Steps/s^2: 0
Speed Steps/s: 0

Number of steps: 0
Correspondent Strain: 0
Max Allowed Speed: 0

Command:

**Figure 3.31:** Graphical user interface (1)

In Figure 3.31 is showed the first part of the GUI, through this is possible to set all the parameters described in the previous paragraphs to control all the instrumentation involved. From left to right, the LCR settings and the Arduino UNO settings, then in the central part there is a short section with some basic instruction on how to use the GUI and also the possibility to define a name for the excel worksheet where the data will be saved. Finally on the right side of the the interface, stretching settings, the parameters about the movement of the stage must be specified. The software will return the "number of steps", the "correspondent strain" and the "max allowed speed". The stage will start to move only after the "start" button has been pressed and the execution of the LabVIEW program can be stopped through the "Stop" button.

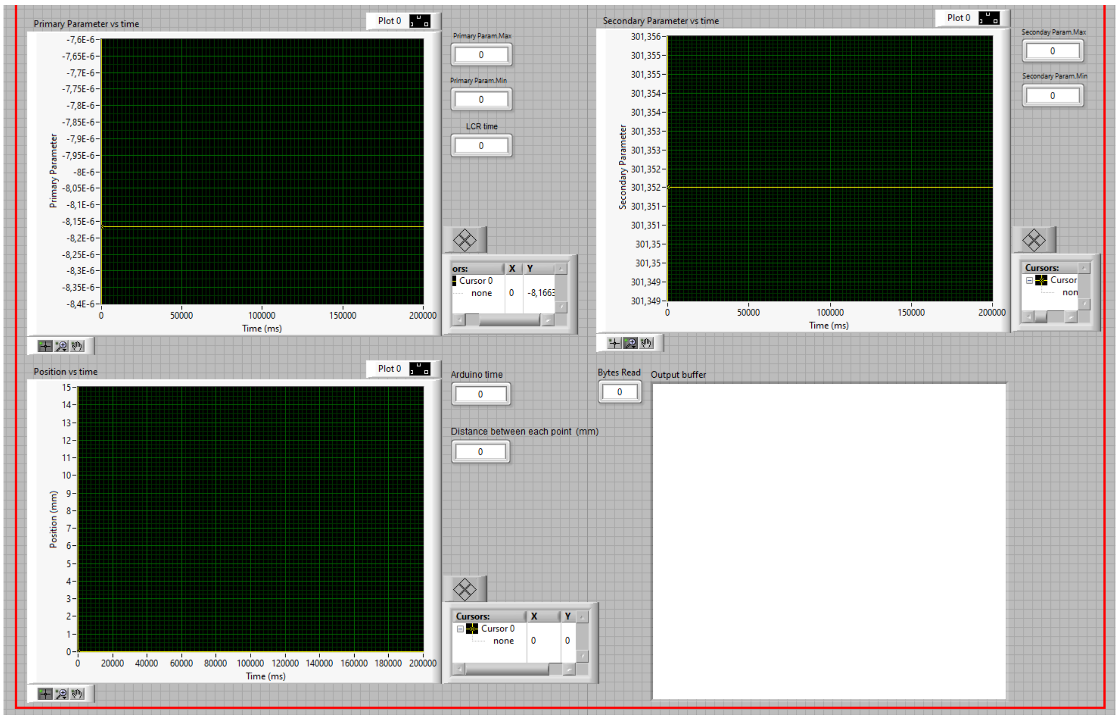


**Figure 3.32:** Graphical user interface (2)

The second part of the GUI, that actually is underneath the first one, is dedicated to the visualization of the data. In Figure 3.32 three different graphs are present, namely Primary parameter vs Time, Secondary parameter vs Time, Position vs Time , plus an "output buffer" section where what's come from the serial port of the Arduino is showed. Near to each graph some additional information are given, the maximum and minimum values of the primary and secondary parameter can be visualized, while the distance between two points is reported near to the "Position vs Time" plot. In conclusion, The two indicators, "LCR time" and "Arduino Time," serve a dual purpose. They indicate the distance on the x-axis between two consecutive points but were also instrumental during the testing phase for verifying the actual synchronization between the LCR meter and Arduino. Indeed, these two times represent the execution time of their respective for loops described in Step 4, where data was read from the Arduino and

LCR meter. As long as the time reported by these two indicators is equal, the sampling between the two instruments occurs simultaneously, and the data correspond one-to-one. However, if the two times are no longer equal, synchronization is lost, along with the correlation between impedance data and position data. These two indicators and the time showed by them is hence used as a sort of "debug" tool to understand which is the limit on the system performance, this will be explained better later but what happen in general is that if the resolution is too high or the space that the stage needs to cover is too small, the position data sent by the Arduino will be transmitted too quickly, resulting in a very small time interval between two consecutive points. On the other hand, the LCR meter will not be able to send impedance data as quickly. As a result, the for loop where data is sampled from the LCR will be slower than that of the Arduino, causing a loss of synchronization between them.

# Chapter 4

# Frequency Sweep

This chapter focuses on the implementation of a frequency sweep, a fundamental procedure in characterizing the behavior of the DUT. The frequency sweep is a technique used to systematically vary the frequency of a signal applied to the DUT while observing its electrical response. In this context, the LabVIEW code developed for the frequency sweep serves as an essential component of the experimental setup, providing valuable insights into the electrical properties of the DUT. The primary purpose of the frequency sweep is to understand how the electrical characteristics of the DUT vary with frequency. By sweeping through a range of frequencies, it is possible to gain insights into the DUT's impedance, capacitance, and other electrical parameters across different frequency regimes. This information is crucial for optimizing the performance of electronic devices and circuits, as it helps identify frequency-dependent effects and design considerations. It's important to note that the LabVIEW code for the frequency sweep represents a preliminary step in respect to the stretching setup, preceding the stretching experiments. While the stretching experiments involve both the LCR meter and the linear stage, the frequency sweep focuses solely on the LCR meter and the insights gained from the frequency sweep inform the selection of the appropriate frequency for signal application during the stretching experiments. Since the electrical properties of the electrode may vary with frequency, understanding these variations is essential for ensuring accurate and reliable experimental results. Unlike the setup used for stretching experiments, which involves more complex procedures and interactions between multiple components, the frequency sweep in LabVIEW is characterized by its simplicity and directness. It focuses solely on controlling the frequency variations applied to the DUT through the LCR meter. This streamlined approach facilitates the systematic exploration of the DUT's electrical response across different frequencies, providing valuable insights into its frequency-dependent behavior. The frequency sweep plays a critical role in the characterization of the DUT's electrical behavior and this chapter will delve into the details of the LabVIEW code developed to control the LCR meter E4980A for the frequency sweep.

## 4.1 Frequency sweep block diagram

In this paragraph the developed LabVIEW code is reported and described. To control the LCR meter and execute the sweep the same driver used for the previous LabVIEW code are adopted and the same sequence of block to initially configure the instrument is followed : Initialize , Configure measurement, Configure measurement signal and Configure aperture. The approach to performing the frequency sweep in LabVIEW is relatively straightforward. Essentially, it involves defining an initial frequency, a final frequency, and a frequency step size. With these three parameters, the number of iterations needed to reach the final frequency from the initial one, advancing by the specified "step" can be derived as follow:

$$Acquired_{points} = \frac{F_{final} - F_{initial}}{F_{Step}} \tag{4.1}$$

To execute the frequency sweep, two nested for loops are employed. The outer loop is responsible for performing multiple consecutive sweeps,

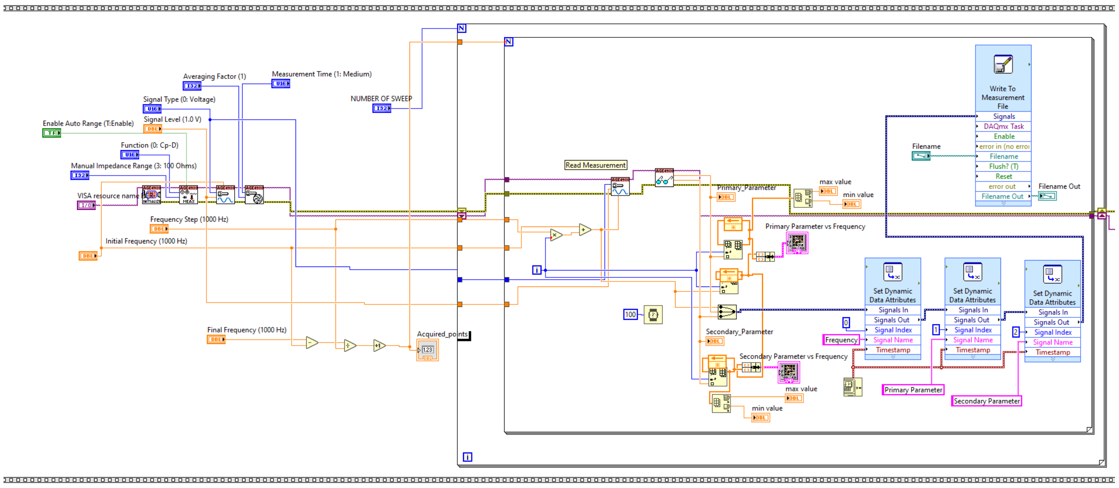
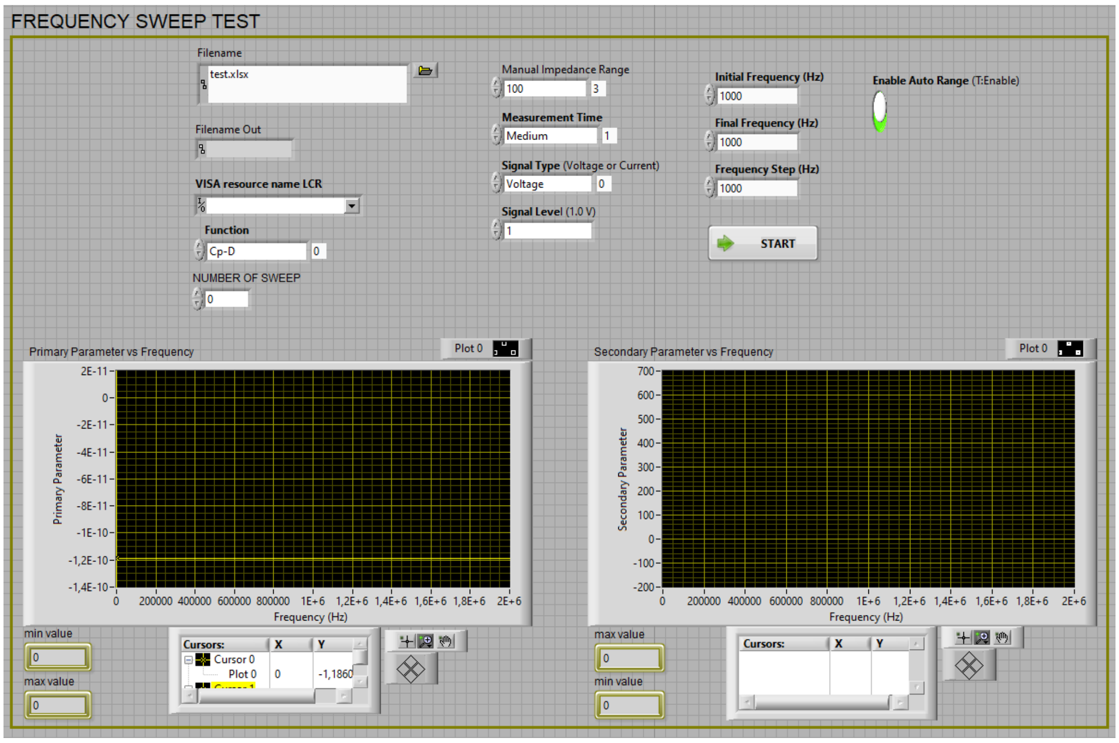


**Figure 4.1:** Frequency sweep block diagram

thereby measuring multiple values at the same frequency intervals. The number of iterations is controlled by the value of "number of sweep." Within this loop, the inner loop conducts the actual sweep. The number of iterations is set equal to the number of "acquired points," calculated in Equation 4.1. Within this loop, two blocks are reused to manage the operation of the LCR meter. Firstly, the "configure measurement signal" block is utilized. This block receives the frequency value at which the electrical parameter of interest is measured during each iteration of the sweep. This value is incremented by an amount equal to the "Frequency Step" at each iteration of the loop. Subsequently, the "Read" block is encountered. Through this block, both the "primary parameter" and the "secondary parameter" are extracted at each iteration. Finally, the read values are plotted in two separate graphs and saved once again in an Excel file, along with their respective frequency values, using the "write to measurement file" block.

## 4.2 Frequency sweep front panel

The resulting graphical user interface is showed in Figure 4.2.


FREQUENCY SWEEP TEST

Filename: test.xlsx

Filename Out

VISA resource name LCR

Function: Cp-D 0

NUMBER OF SWEEP: 0

Manual Impedance Range: 100 3

Measurement Time: Medium 1

Signal Type (Voltage or Current): Voltage 0

Signal Level (1.0 V): 1

Initial Frequency (Hz): 1000

Final Frequency (Hz): 1000

Frequency Step (Hz): 1000

Enable Auto Range (T:Enable)

**Figure 4.2:** Frequency sweep graphical user interface

Once again there is the possibility to chose the name for the excel file and as for to the stretching setup, the same parameters need to be specified for the proper functioning of the LCR meter. These include the Visa resource name, function, impedance range, measurement time, signal type, and signal level. In addition to these parameters, four more fields must be filled for the frequency sweep application: the number of sweeps to be executed, the initial frequency, the final frequency, and the incremental frequency step in Hertz. Referring to the data sheet of the Agilent E4980A LCR meter, the frequency range it can operate within is 20 Hz to 2 MHz. Once these parameters are set, pressing the "start" button will initiate the frequency sweep on the device connected to its terminals. The results will be graphed in the two plots provided: "Primary parameter vs Frequency" and "Secondary parameter vs Frequency," allowing for a visual representation of the device's response across different frequencies.

# Chapter 5

# Testing

## 5.1 Setup evaluation

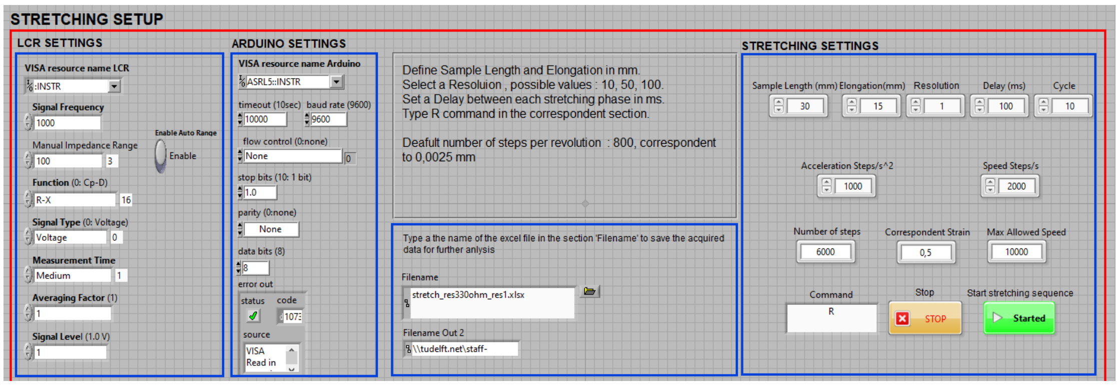
This chapter, titled "Testing," aims to evaluate the actual performance of the developed setup. The primary objective is to validate the setup's functionality through comprehensive tests and to achieve this, several key experiments were conducted, focusing on assessing various aspects of the setup's operation. To validate the setup, the following approach was adopted: a standard 330 Ω resistor was connected to the LCR meter, ensuring that the measured value was known and could serve as a reference for calibration, additionally, the linear stage was set in motion without any attached device/electrode, allowing for the evaluation of its movement and the reception and processing of data through LabVIEW. In this chapter are presented the results of the most significant tests conducted, aiming to elucidate the functionality and effectiveness of the developed setup while also identifying any potential limitations. These results provide valuable insights into the setup's performance, guiding further improvements and optimizations. Through those experiments and analysis, the goal is to demonstrate the reliability and robustness of the developed setup, paving the way for its effective utilization in practical applications. First of all the resolution limit has been tested, indeed increasing the resolution parameters means that the

space between two consecutive data points is smaller and hence the correspondent number of step, a reduced number of steps will result in the motor taking less time to execute them. If the time reduces it means that the Arduino will send serial data faster and hence the respective LabVIEW code must catch up to have a correct sampling. Has already explained the time required by the motor to complete a certain number of steps is computed a priori in the LabVIEW code, through the formula node structure (section 3.4.2), before that the stretching sequence start and given in input to the for loops so that the timing of the LabVIEW code and the one of the Arduino are the same, however, the LCR meter poses an additional limit on the speed at which the system can operate and its maximum speed must be taken into account. Actually the timing does not depend only on the number of step but also the acceleration and speed of the motor play a central role, in any case the scope of these tests is to verify how the "Resolution" parameter impact on the system performances, hence the tests are conducted by fixing all the parameters and increasing the Resolution parameters among the different simulations.
In Table 5.1 are reported the settings used in the next tests.

| | |
|---|---|
| Signal Frequency [Hz] | 1000 |
| Impedance range | Auto |
| Function | R-X |
| Signal type | Voltage |
| Measurement time | Medium |
| Signal level [V] | 1 |
| Sample Length [mm] | 30 |
| Elongation [mm] | 15 |
| Delay [ms] | 100 |
| Cycle | 10 |
| Acceleration[Step/$s^2$] | 1000 |
| Speed[Step/s] | 2000 |

**Table 5.1:** Setup evaluation, stretching sequence settings

At first a Resolution = 1 is imposed, this means that only the value at the resting position (x = 0 mm) and at the final position are recorded at each cycle. With a resolution = 1, one position/impedance point every 15 mm is recorded in that case.



STRETCHING SETUP

LCR SETTINGS

VISA resource name LCR: :INSTR
Signal Frequency: 1000
Manual Impedance Range: 100 | 3
Enable Auto Range: Enable
Function (0: Cp-D): R-X | 16
Signal Type (0: Voltage): Voltage | 0
Measurement Time: Medium | 1
Averaging Factor (1): 1
Signal Level (1.0 V): 1

ARDUINO SETTINGS

VISA resource name Arduino: ASRL5::INSTR
timeout (10sec): 10000
baud rate (9600): 9600
flow control (0:none): None | 0
stop bits (10: 1 bit): 1.0
parity (0:none): None
data bits (8): 8
error out: status ✓ | code 1073
source: VISA Read in

Define Sample Length and Elongation in mm.
Select a Resoluion , possible values : 10, 50, 100.
Set a Delay between each stretching phase in ms.
Type R command in the correspondent section.

Deafult number of steps per revolution : 800, correspondent to 0,0025 mm

Type a the name of the excel file in the section 'Filename' to save the acquired data for further anlysis

Filename: stretch_res330ohm_res1.xlsx
Filename Out 2: \\tudelft.net\staff-

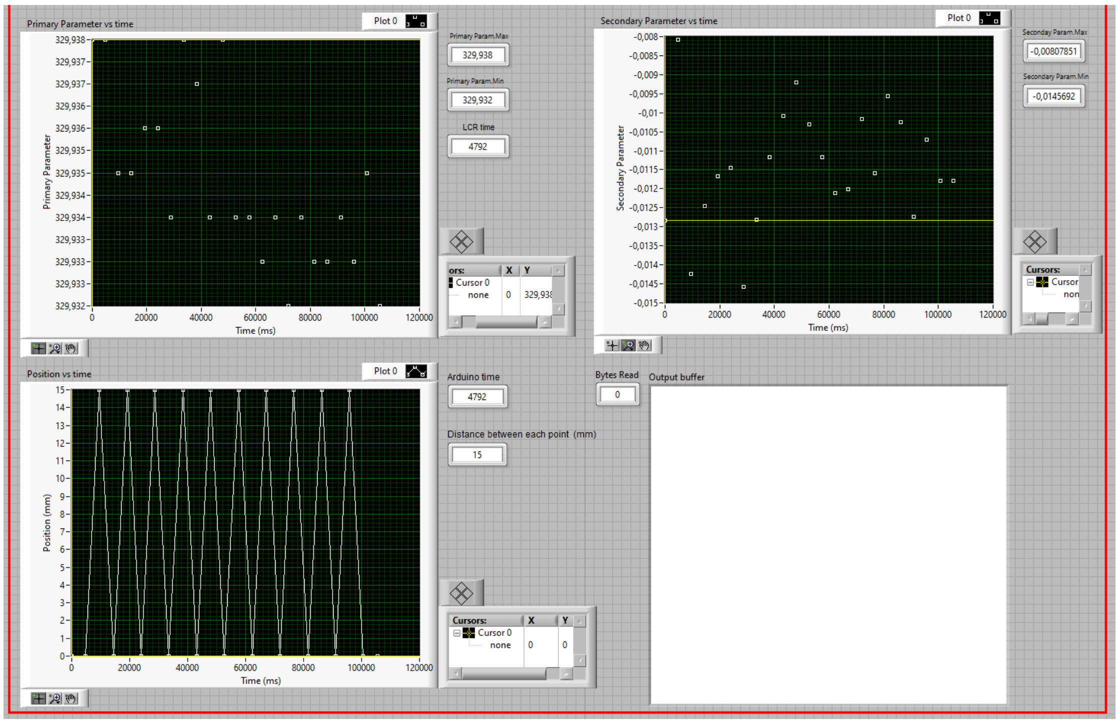
STRETCHING SETTINGS

| Sample Length (mm) | Elongation(mm) | Resolution | Delay (ms) | Cycle |
|---|---|---|---|---|
| 30 | 15 | 1 | 100 | 10 |

Acceleration Steps/s^2: 1000
Speed Steps/s: 2000
Number of steps: 6000
Correspondent Strain: 0,5
Max Allowed Speed: 10000
Command: R

**Figure 5.1:** Stretching sequence settings, resolution = 1

In Figure 5.2 the recorded data are displayed, obviously since a standard 330 Ω resistor is connected to the LCR meter the resulting measurement is an almost fixed value of 329 Ω that oscillates at the third decimal digit, assessing however the goodness of the measurement. For what concern the "Position vs Time" plots the result is consistent with the expectation since ten different peaks are visible, meaning that the stage has moved from 0 mm to 15 mm for ten times and only the initial and the final position at each cycle have been recorded.



Primary Param.Max: 329,938
Primary Param.Min: 329,932
LCR time: 4792
Seconday Param.Max: -0,00807851
Secondary Param.Min: -0,0145692
Arduino time: 4792
Distance between each point (mm): 15
Bytes Read: 0
Output buffer

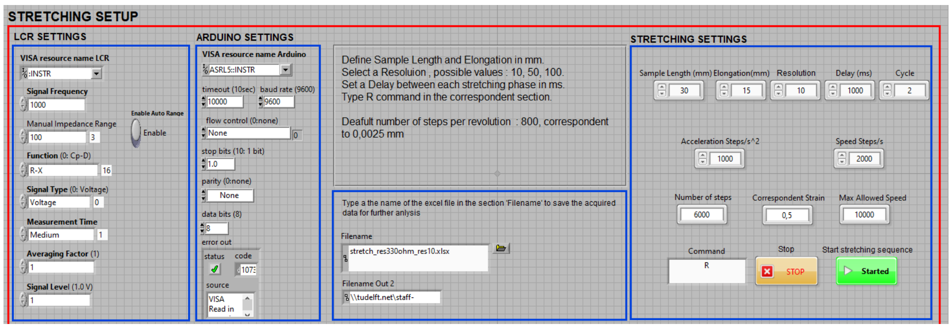
| ors: | X | Y |
|---|---|---|
| Cursor 0 | | |
| none | 0 | 329,938 |

| Cursors: | X | Y |
|---|---|---|
| Cursor 0 | | |
| none | 0 | 0 |

| Cursors: |
|---|
| Cursor |
| non |

**Figure 5.2:** Stretching sequence plots, resolution = 1

Next the resolution has been increased to 10 , keeping all the other parameters fixed, in this case one point every 1.5 mm is recorded.



STRETCHING SETUP

LCR SETTINGS

VISA resource name LCR: :INSTR

Signal Frequency: 1000

Manual Impedance Range: 100 | 3

Enable Auto Range: Enable

Function (0: Cp-D): R-X | 16

Signal Type (0: Voltage): Voltage | 0

Measurement Time: Medium | 1

Averaging Factor (1): 1

Signal Level (1.0 V): 1

ARDUINO SETTINGS

VISA resource name Arduino: ASRL5::INSTR

timeout (10sec): 10000

baud rate (9600): 9600

flow control (0:none): None | 0

stop bits (10: 1 bit): 1.0

parity (0:none): None

data bits (8): 8

error out: status ✓ code 1073; source: VISA Read in

Define Sample Length and Elongation in mm.
Select a Resoluion , possible values : 10, 50, 100.
Set a Delay between each stretching phase in ms.
Type R command in the correspondent section.

Deafult number of steps per revolution : 800, correspondent to 0,0025 mm

Type a the name of the excel file in the section 'Filename' to save the acquired data for further anlysis

Filename: stretch_res330ohm_res10.xlsx

Filename Out 2: \\tudelft.net\staff-

STRETCHING SETTINGS

Sample Length (mm): 30

Elongation(mm): 15

Resolution: 10

Delay (ms): 1000

Cycle: 2

Acceleration Steps/s^2: 1000

Speed Steps/s: 2000

Number of steps: 6000

Correspondent Strain: 0,5

Max Allowed Speed: 10000

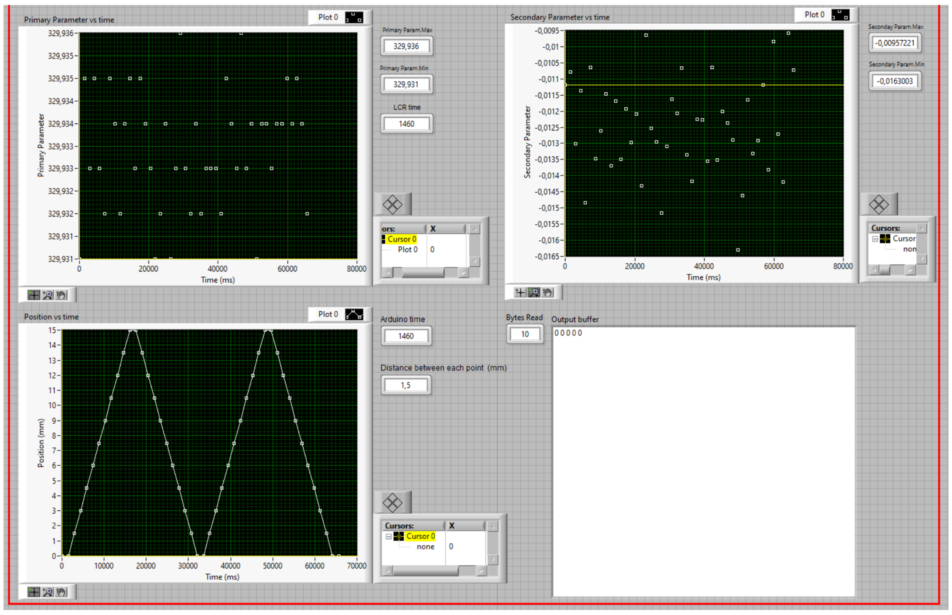
Command: R

Stop: STOP

Start stretching sequence: Started

**Figure 5.3:** Stretching sequence settings, resolution = 10

Again the results are correct and, as can been seen from the "Position vs Time" graph in Figure 5.4, ten points each half cycle are sampled and displayed.



Primary Param.Max: 329,936

Primary Param.Min: 329,931

LCR time: 1460

Secondary Param.Max: -0,00957221

Secondary Param.Min: -0,0163003

Arduino time: 1460

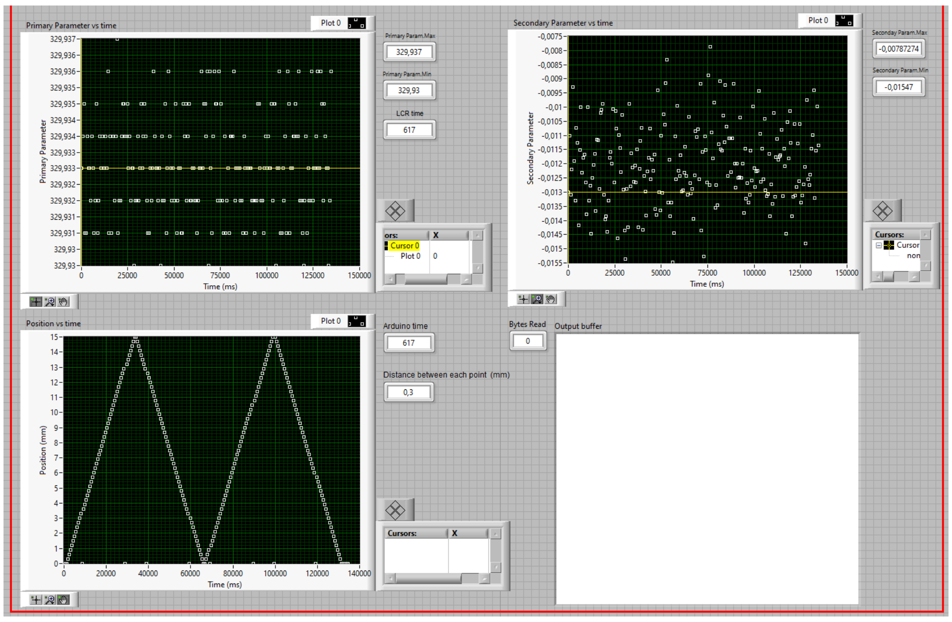
Distance between each point (mm): 1,5

Bytes Read: 10

Output buffer: 0 0 0 0 0

**Figure 5.4:** Stretching sequence plots, resolution = 10

Further increasing the resolution to 50, correspondent to 0.3 mm of distance between each point, what is possible to notice from the position vs time graph in Figure 5.5 is that some unwanted zeroes, 4 in each cycle, are present. Those points results from a misalignment in the serial communication between LabVIEW and Arduino. They start to occur when pushing the performance of the system, in particular when the resolution start to be high in respect the total distance covered by the stage. However they does not affect the quality of the measurement in the sense that through post analysis of the data saved in the excel file can be easily eliminated

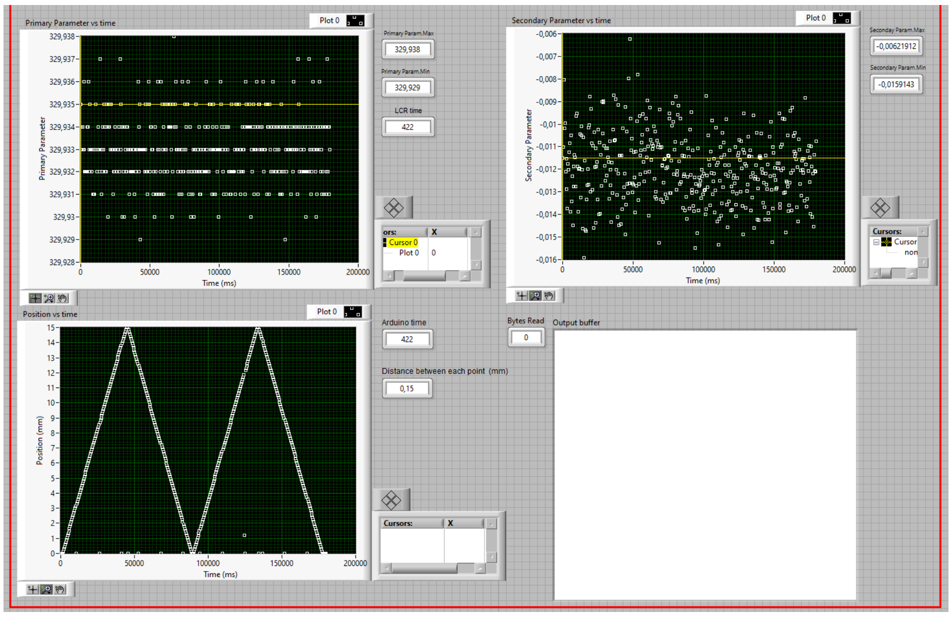


Primary Param.Max 329,937
Primary Param.Min 329,93
LCR time 617
Seconday Param.Max -0,00787274
Secondary Param.Min -0,01547
Arduino time 617
Distance between each point (mm) 0,3
Bytes Read 0
Output buffer

**Figure 5.5:** Stretching sequence plots, resolution = 50

The problem is even more clear doubling the resolution to 100, correspondent to one point every 0.15 mm. In this case, besides the zeros, sampling errors cause the number to be sampled halfway through transmission, resulting in an incorrect value on the LabVIEW graph.

As can be seen in Position vs time plot in Figure 5.6, in the second cycle there is a "random" not aligned point whose value on the y-axis is 1.2 mm. Analyzing the collected data in the correspondent file is clear that the correct value should be 12 mm but due to an incorrect sampling the result is 1.2 mm.



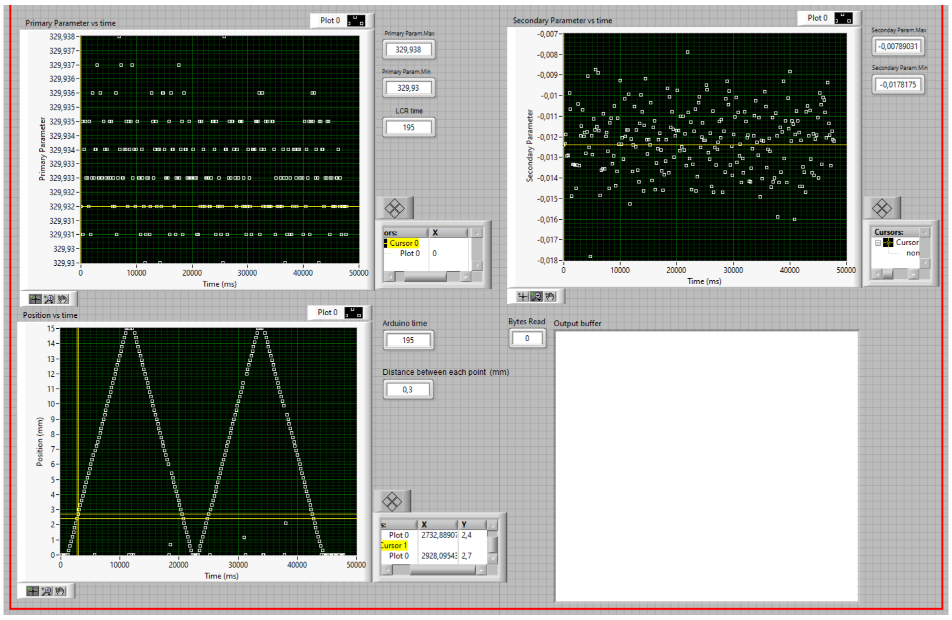
Primary Param.Max: 329,938
Primary Param.Min: 329,929
LCR time: 422
Seconday Param.Max: -0,00621912
Secondary Param.Min: -0,0159143
Arduino time: 422
Distance between each point (mm): 0,15
Bytes Read: 0

**Figure 5.6:** Stretching sequence plots, resolution = 100

As previously mentioned, a crucial requirement for the system to function properly is that the data sampling in LabVIEW by both the LCR meter and the Arduino occurs simultaneously, ensuring that the position and impedance data correspond to each other. The bottleneck lies in the for loop containing the "read" block of the LCR meter, as the execution time of an iteration of the loop cannot fall below a certain threshold. On the other hand, the execution time of the for loop containing the visa read block related to the Arduino can vary significantly.

Previously, the only parameter varied was the resolution, but as mentioned earlier, other factors contribute to the execution speed, primarily acceleration, as well as the total distance the stage must travel. Increasing the acceleration or decreasing the overall distance reduces both the time taken by the motor to cover the desired number of steps and the time taken by the respective for loops to be executed (as they are correlated by the "timing (ms)" variable within the LabVIEW code). Below, some illustrative examples will be presented to clarify this concept further.

In this first example the same parameters given in the tab were used with a resolution of 50 and an acceleration of 10000 steps/$s^2$, the distance between each point is equal to 0.3 mm with these settings.



Primary Param.Max 329,938
Primary Param.Min 329,93
LCR time 195
Seconday Param.Max -0,00789031
Secondary Param.Min -0,0178175
Arduino time 195
Bytes Read 0
Distance between each point (mm) 0,3

| | X | Y |
|---|---|---|
| Plot 0 | 2732,88907 | 2,4 |
| Cursor 1 | | |
| Plot 0 | 2928,09543 | 2,7 |

**Figure 5.7:** Stretching sequence plots, resolution = 50, acceleration = 10000 step/$s^2$

The Arduino time indicator and the LCR time indicator show the same time, 195 ms, meaning that the LCR meter still catch up with the Arduino and there is a correspondence one to one between the two types of data. Nevertheless some more errors appears in the Position vs Time plot (Figure 5.7).

In Figure 5.8 below the same test is repeated but increasing the resolution to 100. In that case a small difference begins to arise between the two timings since the Arduino time is equal to 133 ms while the LCR time is 138 ms , with a displacement of 5 ms and a gap between points of 0.15 mm.

This is due to the dual increase in acceleration and resolution, where the former becomes ten times larger than in the initial tests, while the

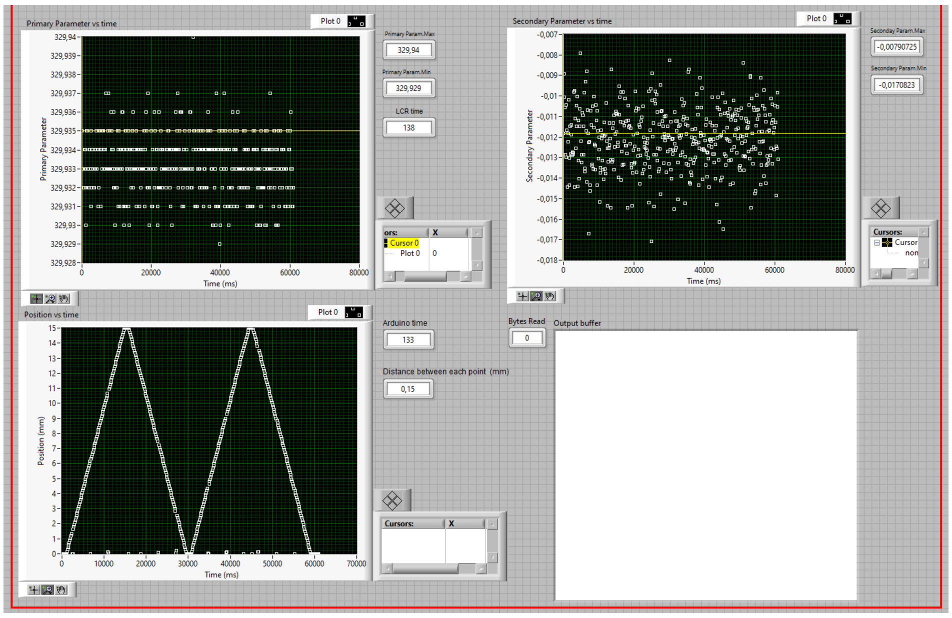


**Figure 5.8:** Stretching sequence plots, resolution = 100, acceleration = 10000 step/$s^2$

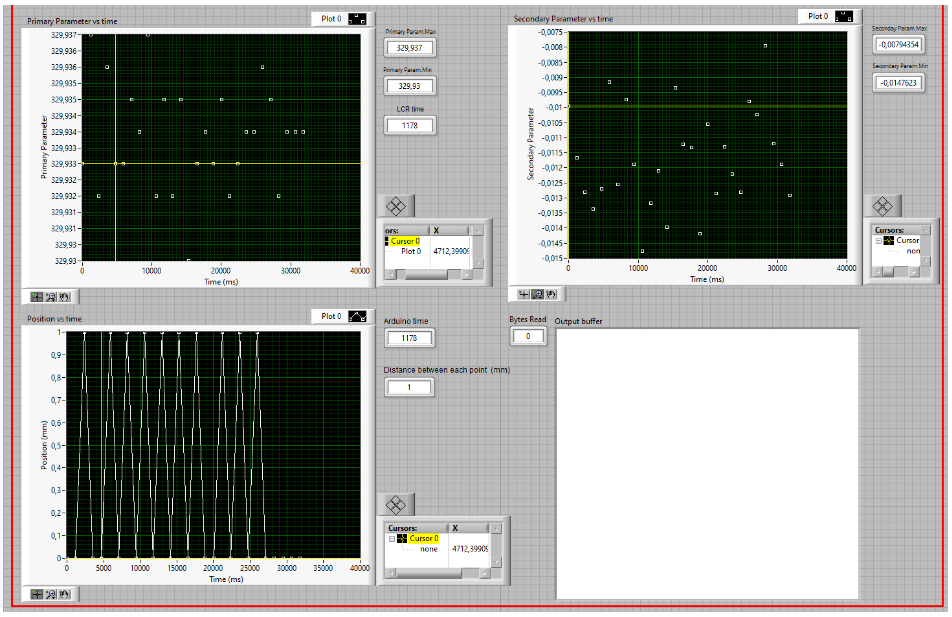
resolution doubles compared to the previous case. In this scenario, the correspondence between the points is lost.

In conclusion, returning to the initial setup and the settings outlined in the table, further tests were conducted with a significantly smaller distance traveled, hence the "elongation" parameter, placing ourselves in a limiting case once again to understand how the distance traveled by the stage affects the speed and thus the functioning of the system.

In the next three test reported the elongation settled is equal to 1 mm, keeping all the other parameter unchanged in respect to the table, the resolution has been made vary again. In Figure 5.9 is showed the test with a resolution equal to 1, even though the travelled distance is small the motor take 1178 ms to complete it. This can be addressed to the fact that 1 mm, since one step is equal to 0.0025 mm, correspond to

$$steps = \frac{1mm}{0.0025mm} = 400 \tag{5.1}$$

which is however a reasonable number, also taking into account that the acceleration and the speed are kept to 1000 steps/$s^2$ and 2000 steps/s, which gives a smooth speed profile to the motor but must be considered as not excessively large values, especially considering that, after some experimental tests, the motor also moves with an acceleration of 40000 steps/$s^2$. In any case the correspondence between



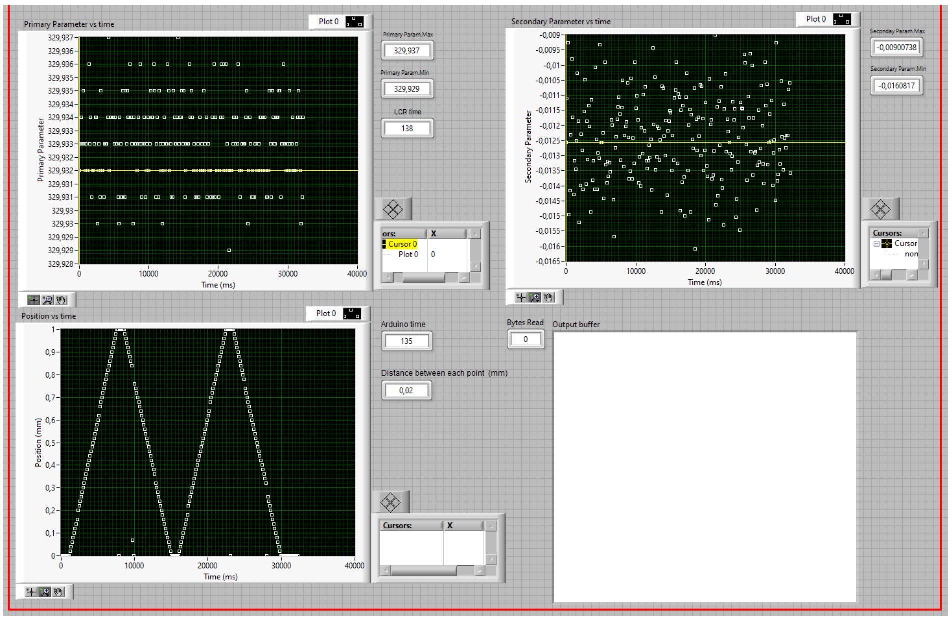
Primary Param.Max 329,937
Primary Param.Min 329,93
LCR time 1178
Seconday Param.Max -0,00794354
Seconday Param.Min -0,0147623
Arduino time 1178
Distance between each point (mm) 1
Bytes Read 0

**Figure 5.9:** Stretching sequence plots, resolution = 1, elongation = 1 mm

LCR time and Arduino time is still standing meaning that the system is working properly.

By significantly increasing the resolution to 50 and keeping the elongation = 1 mm the distance between each point is 0.02 mm. At this point the LCR, whose timing is 138 ms, did not catch up anymore with the Arduino, whose timing is 135 ms, hence the correspondence between recorded data is lost (Figure 5.10).

Further increasing the resolution to 100 the distance between points becomes even smaller and equal to 0.01 mm and the difference between the Arduino and the LCR meter timing more evident. Indeed while

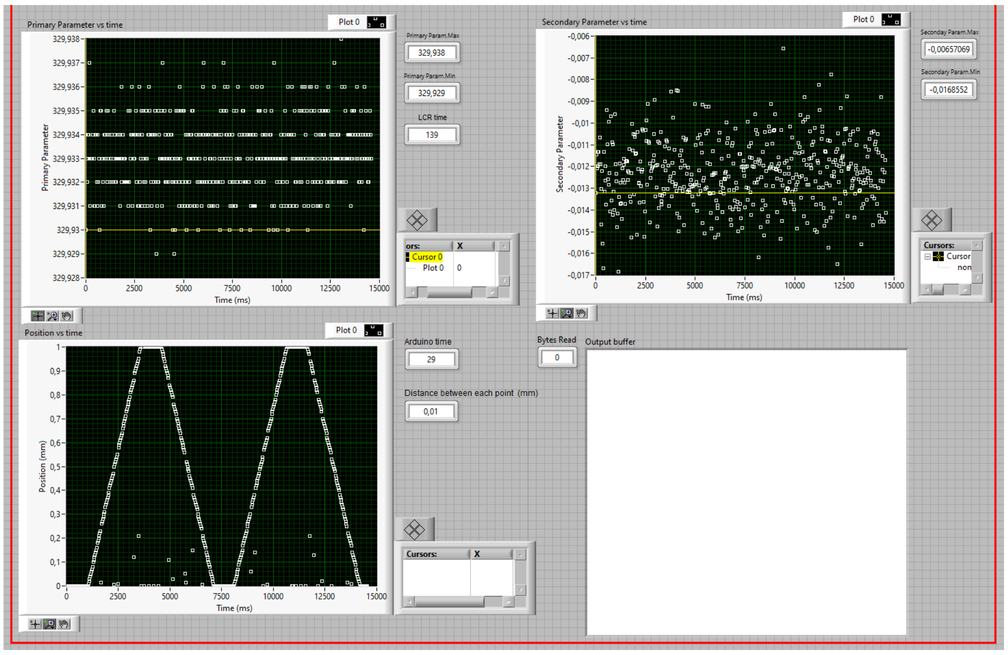


Primary Param.Max: 329,937
Primary Param.Min: 329,929
LCR time: 138
Seconday Param.Max: -0,00900738
Secondary Param.Min: -0,0160817
Arduino time: 135
Distance between each point (mm): 0,02
Bytes Read: 0

**Figure 5.10:** Stretching sequence plots, resolution = 50, elongation = 1mm

the LCR time remains constant, 139 ms actually so 1 ms more than the previous test, the Arduino time drops to 29 ms (Figure 5.11).

In conclusion, after several test in which different combinations of parameters has been tested, what has been noticed is that the lower limit of the LCR for loop and hence the LCR time oscillates between 137 and 139 ms while the Arduino time can swipe over a wider range of values. To ensure the system operates correctly, it's advisable that when dealing with extreme cases, such as when the distance traveled is particularly small, some relaxation of the specifications on other parameters is pursued. This may involve sacrificing a bit of performance (perhaps acquiring fewer data points, making the measurement less precise), but it ensures the validity and accuracy of the measurement.



Primary Param.Max: 329,938

Primary Param.Min: 329,929

LCR time: 139

Seconday Param.Max: -0,00657069

Secondary Param.Min: -0,0168552

Arduino time: 29

Distance between each point (mm): 0,01

Bytes Read: 0

**Figure 5.11:** Stretching sequence plots, resolution = 100, elongation = 1mm

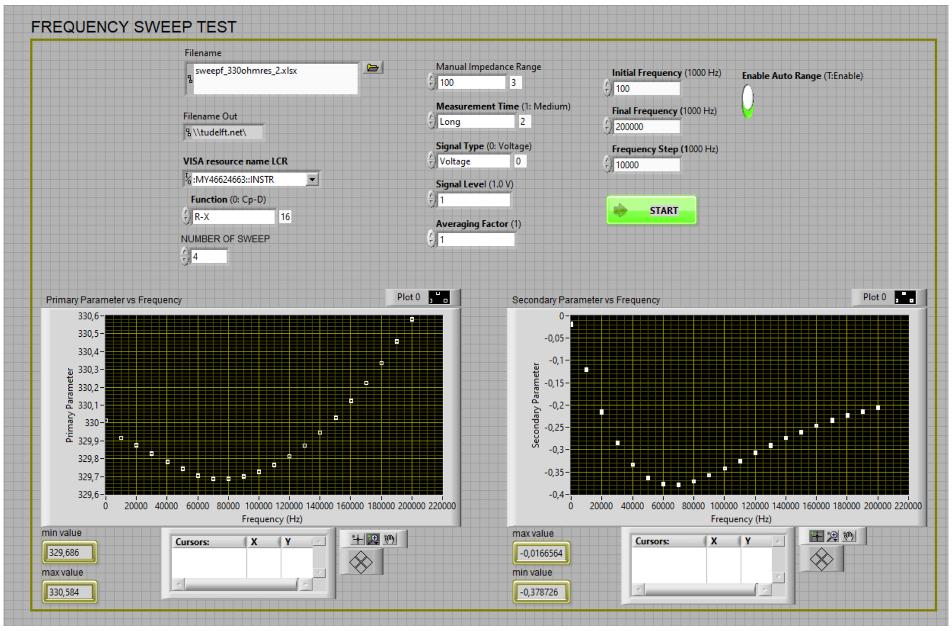
## 5.2 Frequency sweep test

Lastly also the frequency sweep LabVIEW code has been tested, a couple of example are here reported.
In this simulation the same 330 Ω resistor has been connected to the LCR meter E4980A, the simulation setting are showed in Table 5.2.

| | |
|---|---|
| Signal Frequency [Hz] | 1000 |
| Impedance range | Auto |
| Function | R-X |
| Signal type | Voltage |
| Measurement time | Long |
| Signal level [V] | 1 |
| Number of sweep | 4 |
| Initial Frequency [Hz] | 100 |
| Final Frequency [Hz] | 200000 |
| Frequency Step [Hz] | 10000 |

**Table 5.2:** Frequency sweep, test 1 settings

As expected the measured values are basically constant over a wide range of frequency and oscillates from 329.686 Ω to 330.585 Ω (Figure 5.12).
In Figure 5.13 is showed a zoom on the primary parameter vs frequency plot to highlight the number of sweep implemented , correspondent to 4. Indeed 4 resistance values are gained at the same frequency value on the x axis.
In conclusion one last example of a sweep with a smaller frequency step of 1000 Hz over the same frequency range is reported, test parameter in Table 5.3.
The sweep as been made just for one time, the points measured are closed due to the smaller step so the resolution is higher and the measured resistance value oscillates between 329.666 and 330.678 Ω (Figure 5.14).



FREQUENCY SWEEP TEST

Filename: sweepf_330ohmres_2.xlsx

Filename Out: \\tudelft.net\

**VISA resource name LCR**: :MY46624663::INSTR

**Function** (0: Cp-D): R-X 16

NUMBER OF SWEEP: 4

Manual Impedance Range: 100 3

**Measurement Time** (1: Medium): Long 2

**Signal Type** (0: Voltage): Voltage 0

**Signal Level** (1.0 V): 1

**Averaging Factor** (1): 1

**Initial Frequency** (1000 Hz): 100

**Final Frequency** (1000 Hz): 200000

**Frequency Step** (1000 Hz): 10000

**Enable Auto Range** (T:Enable)

START

min value: 329,686

max value: 330,584

max value: -0,0166564

min value: -0,378726

**Figure 5.12:** Frequency sweep test of a standard 330 Ω resistor, number of sweep = 4, frequency range = [100 Hz - 2 MHz], frequency step = 10000 Hz.

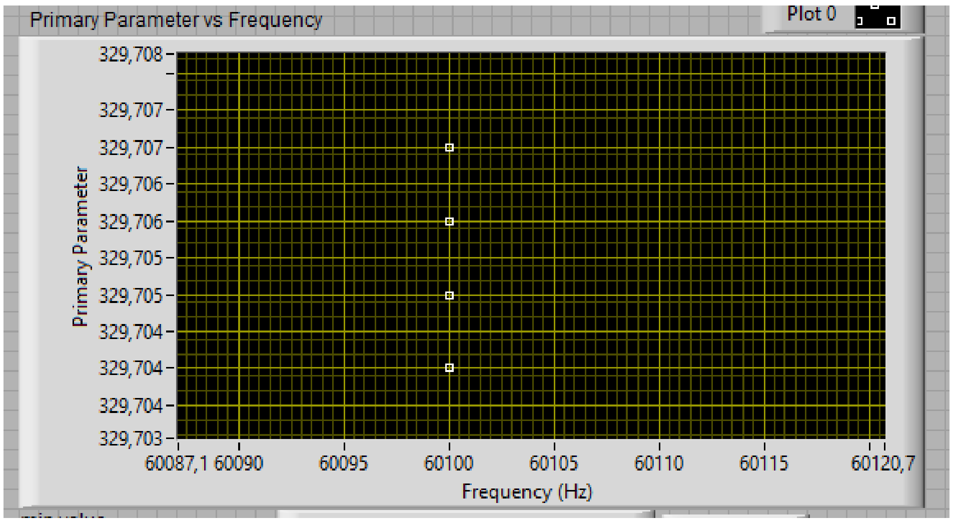
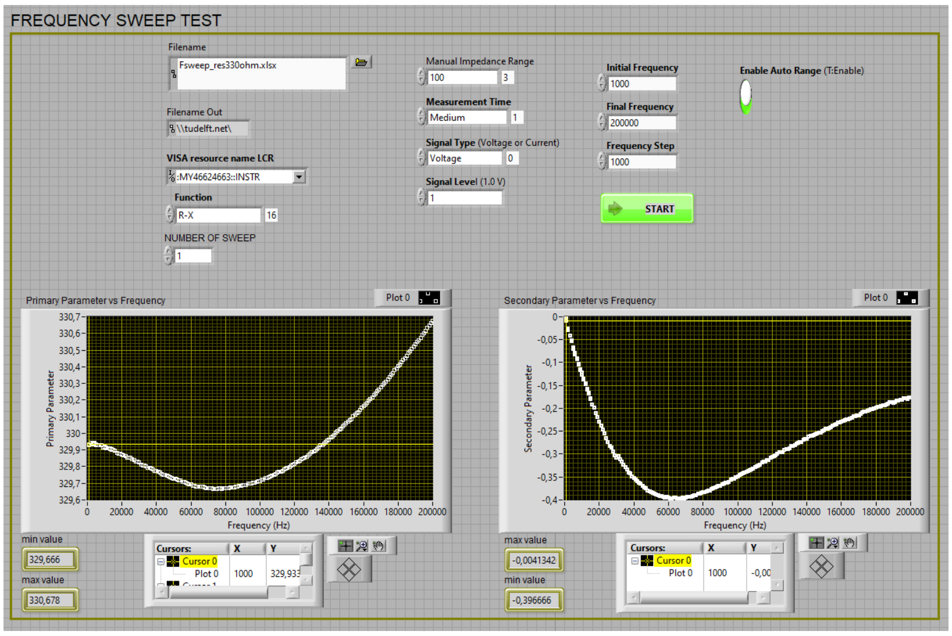


**Figure 5.13:** Frequency sweep zoom

| | |
|---|---|
| Signal Frequency [Hz] | 1000 |
| Impedance range | Auto |
| Function | R-X |
| Signal type | Voltage |
| Measurement time | Medium |
| Signal level [V] | 1 |
| Number of sweep | 4 |
| Initial Frequency [Hz] | 100 |
| Final Frequency [Hz] | 200000 |
| Frequency Step [Hz] | 1000 |

**Table 5.3:** Frequency sweep, test 2 settings


FREQUENCY SWEEP TEST

Filename: Fsweep_res330ohm.xlsx

Filename Out: \\tudelft.net\

VISA resource name LCR: :MY46624663::INSTR

Function: R-X 16

NUMBER OF SWEEP: 1

Manual Impedance Range: 100 3

Measurement Time: Medium 1

Signal Type (Voltage or Current): Voltage 0

Signal Level (1.0 V): 1

Initial Frequency: 1000

Final Frequency: 200000

Frequency Step: 1000

Enable Auto Range (T:Enable)

min value: 329,666

max value: 330,678

| Cursors: | X | Y |
|---|---|---|
| Cursor 0 | | |
| Plot 0 | 1000 | 329,933 |

max value: -0,0041342

min value: -0,396666

| Cursors: | X | Y |
|---|---|---|
| Cursor 0 | | |
| Plot 0 | 1000 | -0,00 |

**Figure 5.14:** Frequency sweep test of a standard 330 Ω resistor, number of sweep = 4, frequency range = [100 Hz - 2 MHz], frequency step = 1000 Hz.

# Chapter 6

# PDMS-Au electrodes

The idea of fabricating flexible electrodes to test their actual stretchability, taking the first steps towards the characterization of an extensible component, arises from a dual need. Firstly, the desire to use the produced setup in a real-use case, by testing an actually flexible and stretchable component, unlike what was done in Chapter 5, where a standard resistor was used to evaluate the setup's performance. This is undoubtedly useful to understand the real problems involved in this type of measurements, starting from the mechanical resistance of the electrode, which must be mounted on the stage, through the difficulty of actually detecting the electrical parameters of interest of a device whose equivalent electrical circuit is not known a priori. In this perspective, the decision was to attempt to fabricate and subsequently characterize electrodes composed of a PDMS substrate on the surface of which a layer of gold was deposited to act as a conductor. These types of electrodes do not fall into the category of biodegradable electrodes, the ultimate application for which the setup is intended to be used, but they are nevertheless a challenging test to investigate and better understand the dynamics and issues mentioned above. The choice of using these materials, PDMS and gold, was dictated by practical reasons, namely the ability to use more standard methodologies in fabrication, such as those required in PDMS fabrication or gold evaporation [20]

[21], compared to other types of electrodes such as those composed of iron nanowires, whose fabrication process is undoubtedly more difficult. Additionally, a wider range of reference materials in the literature with which to compare the obtained results certainly led to the choice of this type of electrode. The second reason for the realization of these devices is closely related to what has just been said. In fact, as anticipated in the introductory chapter, the aim of the project was to use the setup to provide an initial characterization of biodegradable electrodes composed of Fe-NW deposited on a POMAC substrate.Hence, a comparison between the latter and the PDMS-Au electrodes, a more widely-used technology, can prove useful in the purpose of this thesis. In this chapter the procedure followed to fabricate the electrodes will be described and the methodology used to prepare them for the testing setup explained, lastly the results achieved will be showed.

## 6.1 Manufacture process

In Figure 6.1, the main fabrication steps involved in the production of the electrodes are shown.

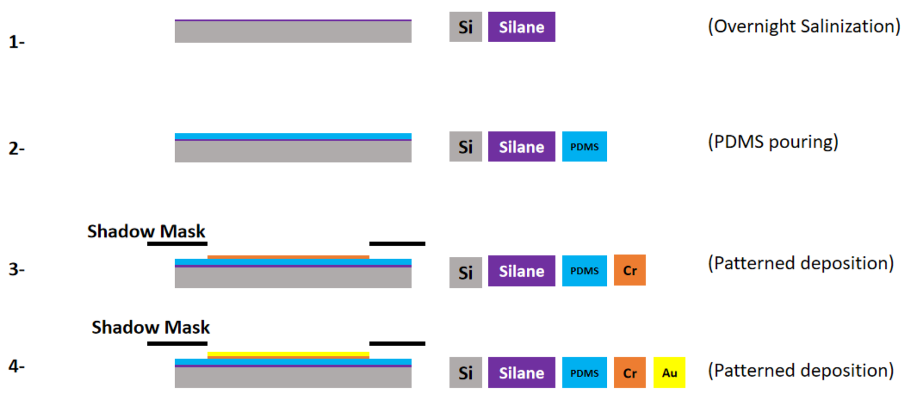


**Figure 6.1:** PDMS-Au electrodes fabrication process

These can be summarized into 4 major steps: firstly a 10 cm wafer has been placed in a petri dish acting as an holder and a preliminary overnight silanization step has been executed. The idea was just to pour the PDMS on top of the wafer inside the petri dish so this step was necessary to easily remove the PDMS from the wafer after the casting. Subsequently, the PDMS itself is fabricated and poured onto the wafer inside a petri dish to obtain a thin substrate. Then, using a specific mask, chromium (Cr) is deposited onto the PDMS, and finally, the gold layer is deposited via sputtering.

### 6.1.1 PDMS fabrication

PDMS (Polydimethylsiloxane) is a silicone elastomeric polymer widely used in various industrial and scientific applications, including biomedical ones. It is known for its properties of transparency, biocompatibility, temperature resistance, it can be molded into complex shapes to be used in a wide range of applications and it is often used for manufacturing flexible shells or membranes due to its ability to conform to various surfaces and its high tensile strength [22], [23]. Overall, in the scope of this project, the most interesting feature of that the PDMS exhibits is its hyperelastic characteristics, meaning it can endure significant deformations without breaking, that makes the PDMS perfectly suited to act as a substrate for stretchable electrodes. The PDMS properties are listed in Table 6.1 for comprehensive coverage.

| | |
|---|---|
| Index of refraction | 1.4 |
| Thermal conductivity (W / $m \cdot K$ | 0.2 - 0.27 |
| Dielectric strength (kV / mm) | 19 |
| Dielectric constant | 2.3-2.8 |
| Electrical conductivity ($ohm \cdot m$) | 4 x $10^2$3 |
| Young's modulus [kPa] | 360-870 |
| Poisson ratio | 0,5 |
| Tensile strength (MPa) | 2.24-6.7 |
| Viscosity ($Pa \cdot s$) | 3.5 |
| Melting point (C) | -49.9 to -40 |

**Table 6.1:** PDMS properties

To prepare the PDMS the curing agent and the base have been mixed with a standard ratio 10:1, after the mixing the PDMS is poured into the petri dish were the silanized wafer is located. Now the wafer is completely covered by the liquid PDMS and the next step require to put the wafer in the vacuum desiccator. The Wafer remains in the vacuum desiccator to be degassed for 2 hours in order to completely remove all the bubbles and given strength to the compounds. The last step is about curing : the wafer with the

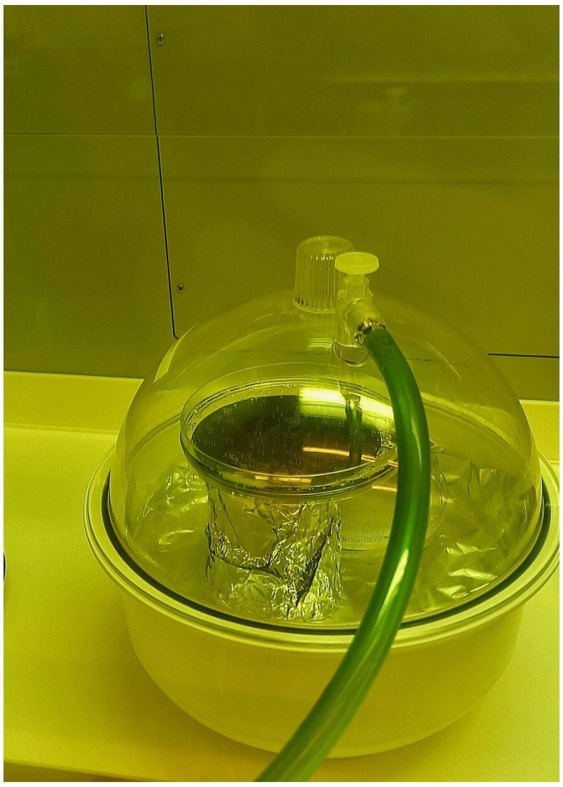


**Figure 6.2:** Wafer kept in vacuum desiccator for 2 hours

PDMS layer on top is placed in the oven for other 2 hours at 80°. The curing is essential to bring the PDMS from the liquid state to the solid elastomeric state in which the PDMS exhibits the desired elesticity and mechanical strength.

### 6.1.2 Mask creation and metal sputtering

Based on various articles and previous works found in the literature, aiming to have electrodes with the main characteristic of being stretchable, it was decided to give the deposited gold a serpentine shape. This was done to increase the chances of the thin gold layer not breaking during stretching, thus enhancing the resistance during strain. For example in [24] "the tensile and conformal properties of three different interconnection structures (mesh, arc, serpentine)" are investigated, stating that the serpentine shapes have the best stretchability, while in [25] a stretchable strain sensor with an Au metal electrode with a serpentine shape is proposed. To give to the electrode a serpentine shape a mask has been designed using Autocad. To optimize time, it was decided to create a single mask with the dimensions of the wafer. On this mask, two different shapes were designed: one corresponding to a resistor and one corresponding to a capacitor. Additionally, for each type, two versions were made, one slightly smaller than the other, to check for potential differences during characterization.

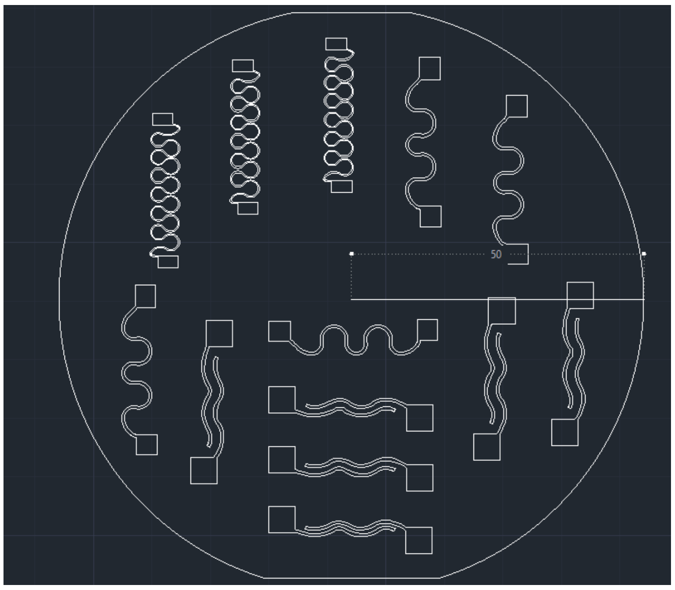


**Figure 6.3:** Mask design in Autocad

In Figure 6.3 a screenshot from the Autocad file is taken, basically for different structure can be recognized :

1. Resistor big : width of the trace 0.5 mm, total length 30 mm , distance between pads 22 mm.
2. Resistor small : width of the trace 0.2 mm, total length 30 mm , distance between pads 22 mm.
3. Capacitor big : width of the trace 0.5 mm, total length 28 mm , distance between pads 20 mm, distance between plates 1.5 mm
4. Capacitor big : width of the trace 0.5 mm, total length 28 mm , distance between pads 19 mm, distance between plates 0.5 mm

Next, the mask was fabricated using laser cutting on a Kapton sheet. Kapton is commonly used as a masking material, it is a polyimide film known for its high temperature resistance, chemical inertness, and excellent electrical insulation properties.
Also the Kapton film was chosen for its adhesive properties, ensuring good adhesion to the PDMS surface and preventing the penetration of other materials or the sputtered gold underneath.

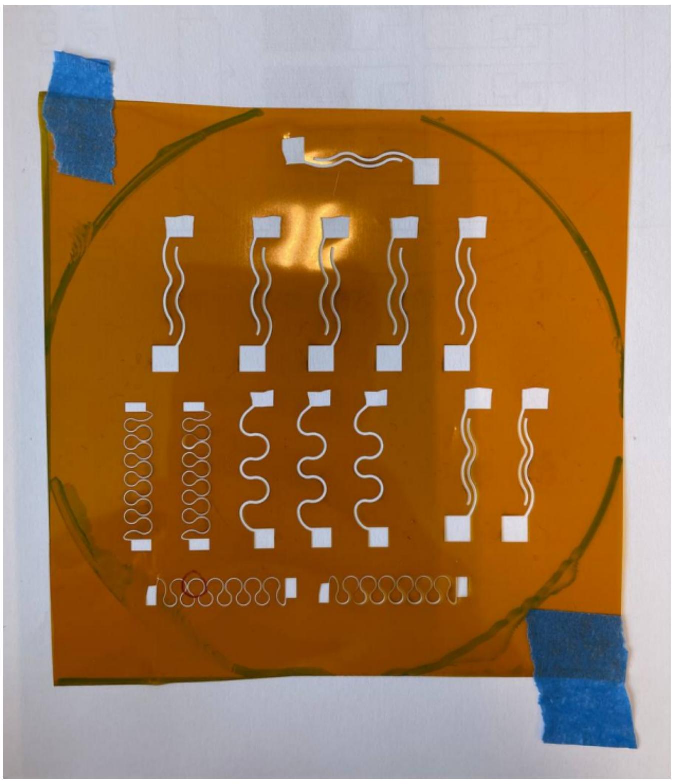


**Figure 6.4:** Developed kapton mask after laser cutting

In Figure 6.4 the Kapton mask is showed and after the gold sputtering the final result can be seen in Figure 6.5.

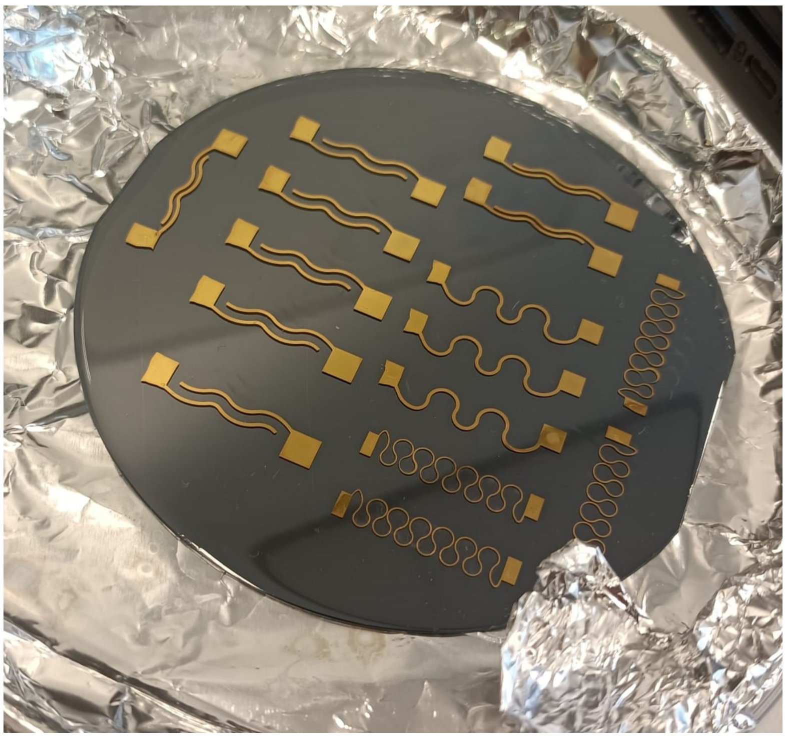


**Figure 6.5:** Developed PDMS-Au electrodes

### 6.1.3 Electrodes preparation to the setup

Once the electrodes were obtained, they needed to be prepared for mounting on the setup in order to measure their electrical properties at rest and under varying elongation. While this may seem like a secondary issue, it is actually quite complex and of crucial importance to ensure good and consistent results as well as reliable measurements. Especially in contexts like this, where the device under test is very fragile (the gold serpentine is only 100 nm thick) and delicate, the variations in the parameters to be measured can be very small and the testing setup itself applies further external forces that can compromise

the measurement and the integrity of the sample, various precautions need to be taken, and the device must be somehow 'adapted' to the measurement context. Precisely, when the electrode is clamped on the stage, one side to the fixed part and the other one to the movable part, the two metal bars used the fix it exert some transversal forces on the electrode. This may cause the electrode to bend even before stretching begins or lead to damage to the underlying parts beneath the two bars. To face do this problem and give strength to the electrodes the same type of approach adopted in [26] has been followed. In this paper the FeNW and carbon nanotubes compound is encapsulated in a double PDMS layer and two copper foil at the ends are used as contacts for the measurement. In this work the same PDMS encapsulation has been replicated but differently two iron wires have been used instead of the copper foil as contacts for the LCR meter.
The procedure is here summarized :

1. The PDMS layer is peeled of from the wafer and each electrode is cut and separated to be placed in another petri dish

2. The iron wires are brought into contact with the electrode pads

3. To enhance the conductivity between pads and wire some droplets of conductive silver paste have been poured on the point of contact

4. Once the silver paste has dried, to improve the mechanical strength of the contact point between the two metal, which is the most fragile point of the structure , some droplets of silicone have been poured to cover the pads surface and the portion of wire that overlaps.

5. When also the silicone paste has dried the samples are cured for 1.5 hours in the oven at 80°C

6. Now the encapsulation PDMS layer is poured on the petri dish that contains the electrodes in order to cover everything with a

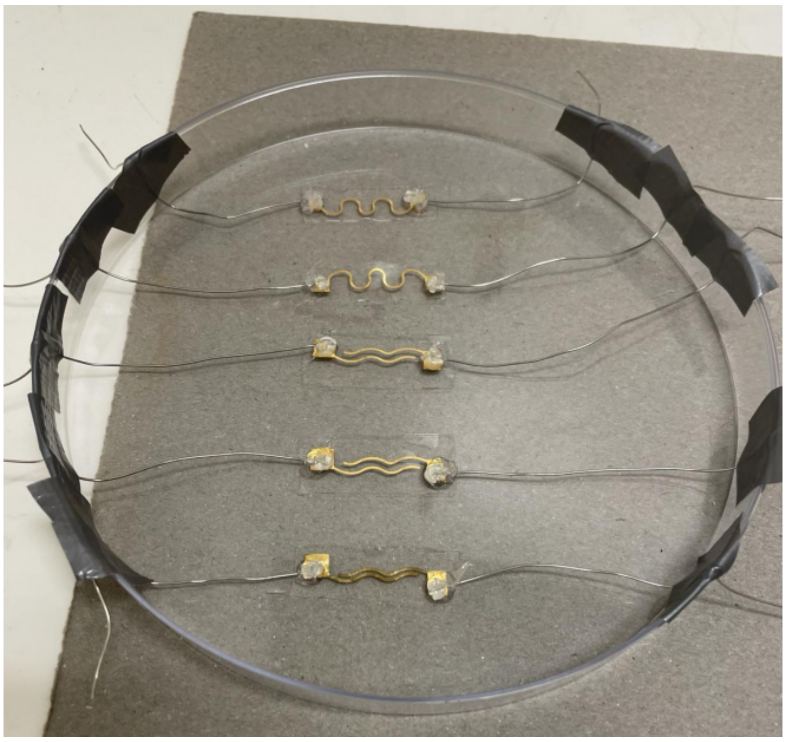


**Figure 6.6:** Electrodes with silver paste and silicon layers in the pads position while drying.

strong protective layer, keeping attention to leave some portion of the wire outside the PDMS.

The final result is showed in Figure 6.7, each electrodes is covered by a thicker PDMS layer that offer protection to the structure.

## 6.2 Results and Discussion

In this paragraph, the tests conducted on the developed electrodes are summarized and the obtained results are discussed. Firstly, it should be noted that not all the produced samples yielded successful results; in fact, several electrodes ended up to be not conductive. This is likely attributed to issues during the electrode construction phase and

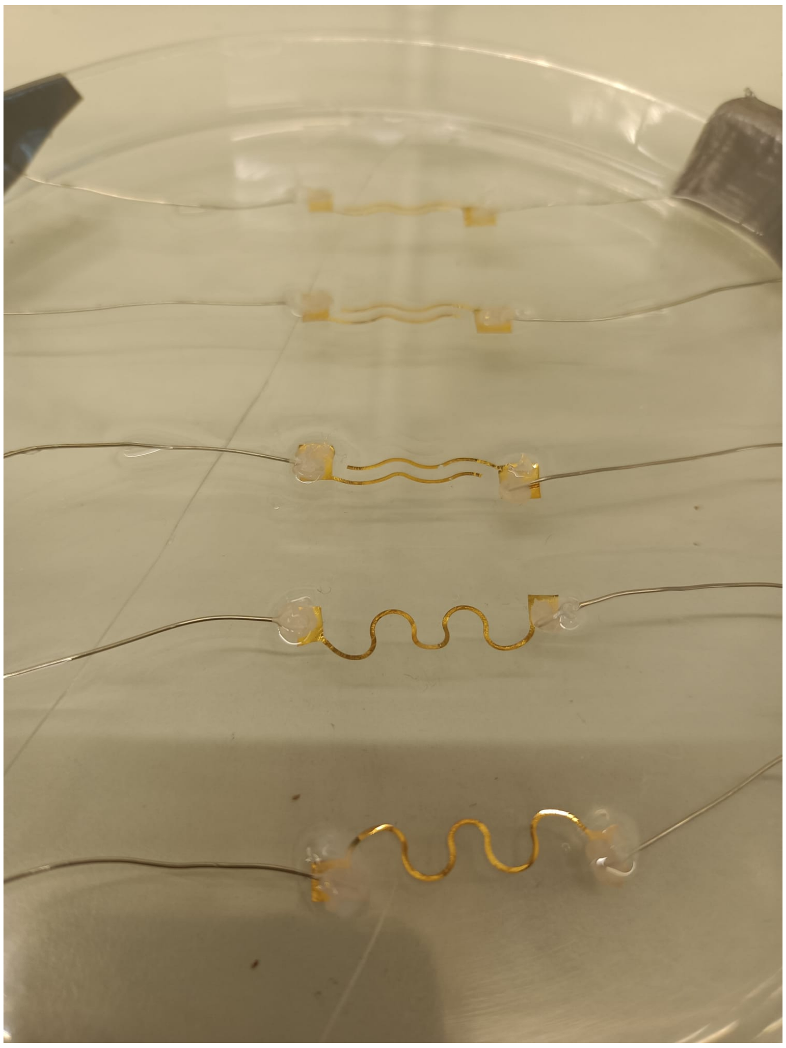


**Figure 6.7:** Electrodes after that the second PDMS encapsulation layer has been poured

preparation for the setup. As mentioned earlier, especially the latter step is crucial and undoubtedly requires further refinement to achieve more satisfactory outcomes. Furthermore, to ensure consistent and reliable results, a larger number of samples need to be tested, which, due to time constraints, was not feasible within this project. Therefore, the following discussion should be viewed as a foundation for future research concerning the fabrication and characterization of flexible electronics.

Before starting some preliminary considerations about the electrical model that approximates the electrodes electrical behaviour themselves has been made in order to understand which kind of measurements has to be done with the LCR. Looking again in the literature some interest insight can be taken, in [27] a flexible dry electrode is proposed and the proposed equivalent circuit to approximate the device is composed by a resistor and a stray capacitor in parallel. Also in [28] the proposed electrode is approximated with the same scheme. Starting from these basis some further considerations have been made on the developed electrodes. Indeed considering the serpentine shape a simple model composed by just a resistor and a capacitor cannot be satisfactory. To take into account the shape of the electrode a stray inductance in parallel to the circuit is considered. This inductance is due to the wavy shape of the gold serpentine, and its contribution can vary depending on the frequency at which the electrode operates.

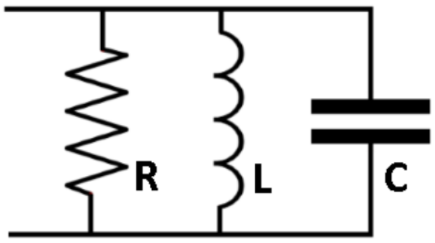


**Figure 6.8:** Resistor equivalent circuit model

Hence , in a first attempt, a frequency sweep was performed on the larger resistor by connecting the two wires at the ends of the electrode to the terminals of the LCR meter and running the corresponding LabVIEW program.
The same test has been executed on three different serpentine-shaped resistors, the frequency has been swept from 1 kHz to 2 MHz at step of 10 kHz applying 1 V signal to the DUTs.
Both resistors show the same behaviour among the frequency range. They start from the same range of values [90-110 Ω] and remain more

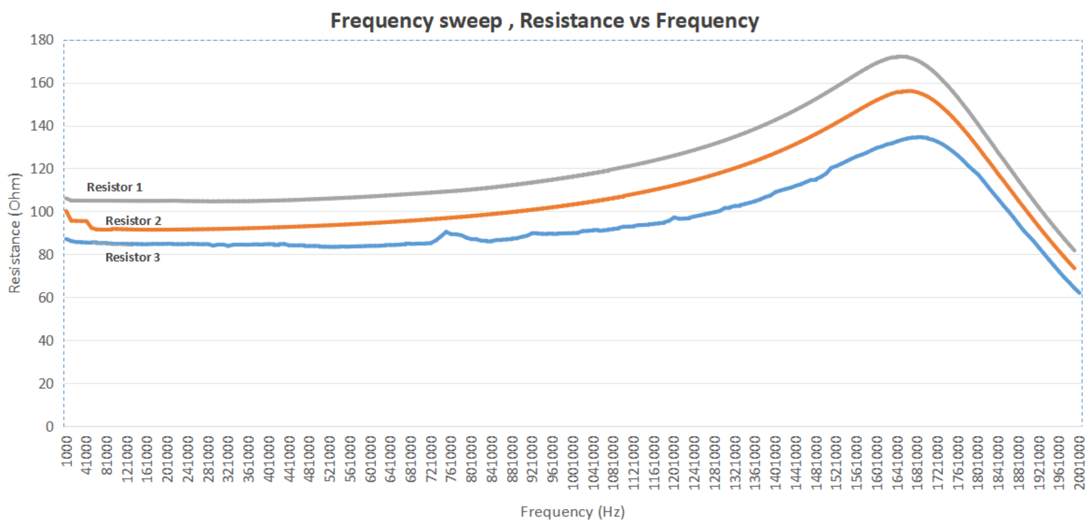


**Figure 6.9:** PDMS-Au resistors frequency sweep, resistance evaluation

or less constant till 60 kHz were all of them start to slightly increase till a maximum value reached at 1.6 MHz. After that the resistance suddenly drop to restore to the original value around 2 MHz.
Next, to further investigate the behaviour of the sample in respect to different frequencies, other sweeps have been made to measure the change of the stray component and which is their contribution.
In Figure 6.10 are showed three different graphs, the blue and the black one represent the trend of the reactance of the resistor 1 and 3 respectively. The orange plot instead is the contribution of the series stray inductance to the Resistor 2.
The trend of the two reactance show that the parasitic component is initially positive and then drop very rapidly to negative values around 1.3 MHz , reaching a maximum value of 130 Ω.
The orange line representing the series stray inductance remains constant to 1 $\mu$H among all the frequency range confirming that the theoretical model was correct.
The blue and black plots are significant as they provide insights into

the nature of the parasitic component and its absolute value. Indeed,

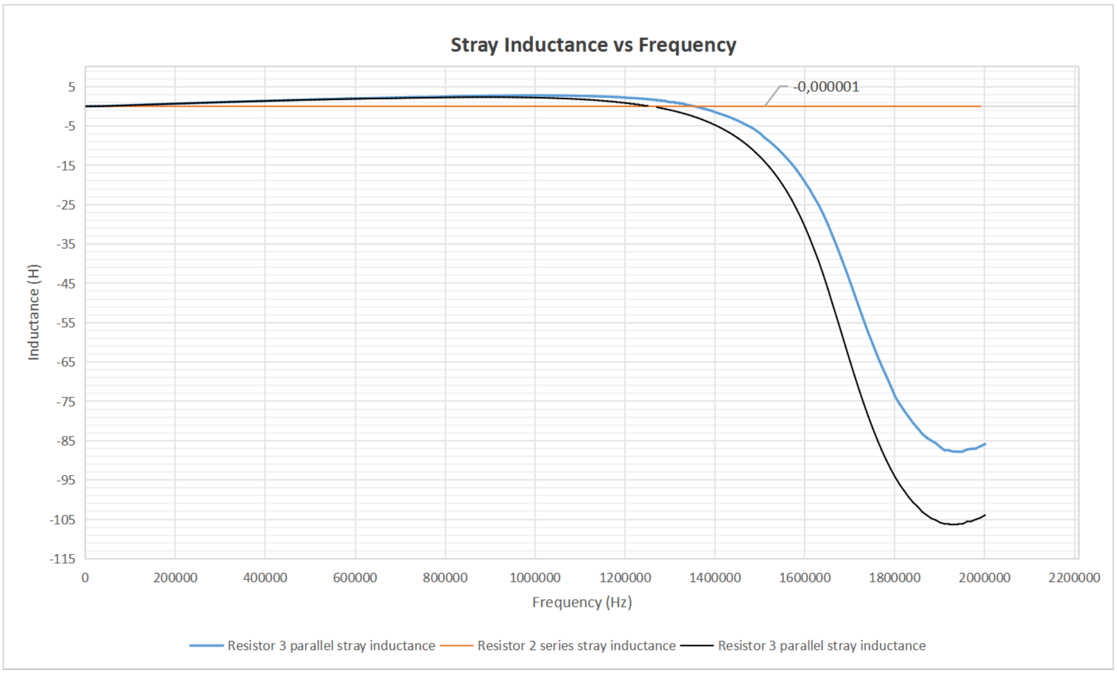


**Figure 6.10:** PDMS-Au resistors frequency sweep, reactance evaluation

with the reactance being positive between 1 kHz and 1.3 MHz, it should be considered an inductance. However, at frequencies above 1.3 MHz, the capacitive component predominates, as the reactance values become negative.

Finally, the stretchability of the same 3 electrodes was tested using the stretching setup. Each electrode was elongated starting from a strain of 5%, gradually increasing to reach a 40% strain. The change in resistance relative to the applied strain was measured and plotted in Figure 6.11 for each of the three resistors. The trend observed for each of the three samples analyzed aligns with expectations, with resistance increasing as elongation increases.

Overall, the results obtained are in line with expectations, but it should be noted that only a small number of devices have been tested, and testing on a larger scale is necessary to fully consider the results

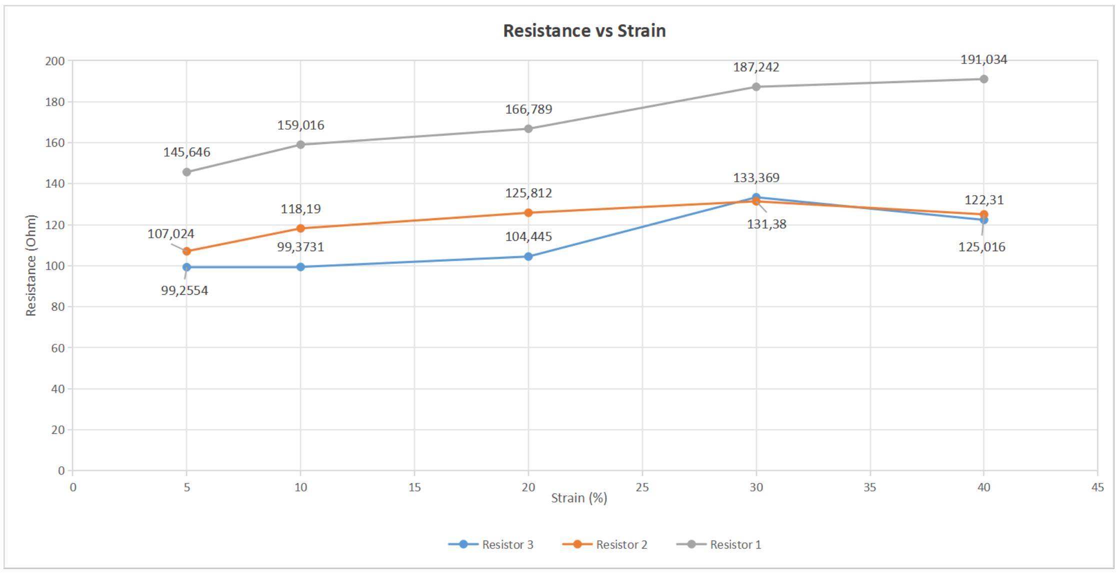


**Figure 6.11:** PDMS-Au electrodes stretching from 5 to 40 %

reliable. Furthermore, the measured values, although reasonable in magnitude, need to have the noise and resistance components introduced by the measurement connections subtracted.
As for the electrodes composed of smaller dimension resistances, the aforementioned issues regarding fabrication, noise, and contact resistance have led to the inability to obtain any significant measurements. This can be attributed, in particular, to several factors in the fabrication phase. Firstly, the very small thickness of the serpentine and the number of folds composing it cause the traces to be almost in contact at the folding points, greatly increasing the effect of the parasitic inductive component. Furthermore, the silicone used during the setup preparation phase to strengthen the contact point between the wire and the pad, upon drying, caused the thin layer of PDMS serving as the substrate to contract and bend slightly, inducing the same movement on the gold serpentine. This resulted in microfractures already present on the gold even before performing the measurements. The same issues related to noise and conductivity were encountered with the capacitive electrodes.

In fact, by calculating the theoretical capacitance, the expected value is in the range

$$C = \frac{\epsilon \cdot l \cdot t}{d_{PDMS}} = \frac{2.3 \cdot 20 \cdot 10^{-3} \cdot 100 \cdot 10^{-9}}{1.5 \cdot 10^{-3}} = 3.0667\mu F \tag{6.1}$$

where C is the capacitance of the capacitor, $\epsilon$ is the dielectric constant of the material between the plates, in this case PDMS, l is the length of the electrode, t is the thickness of the electrode and $d_{PDMS}$ is the distance between the two metal traces.
In Figure 6.12, frequency sweeps conducted on three different capacitors are displayed. Each sweep is repeated four times on each device and goes from 20 Hz to 2 MHz at each cycle.

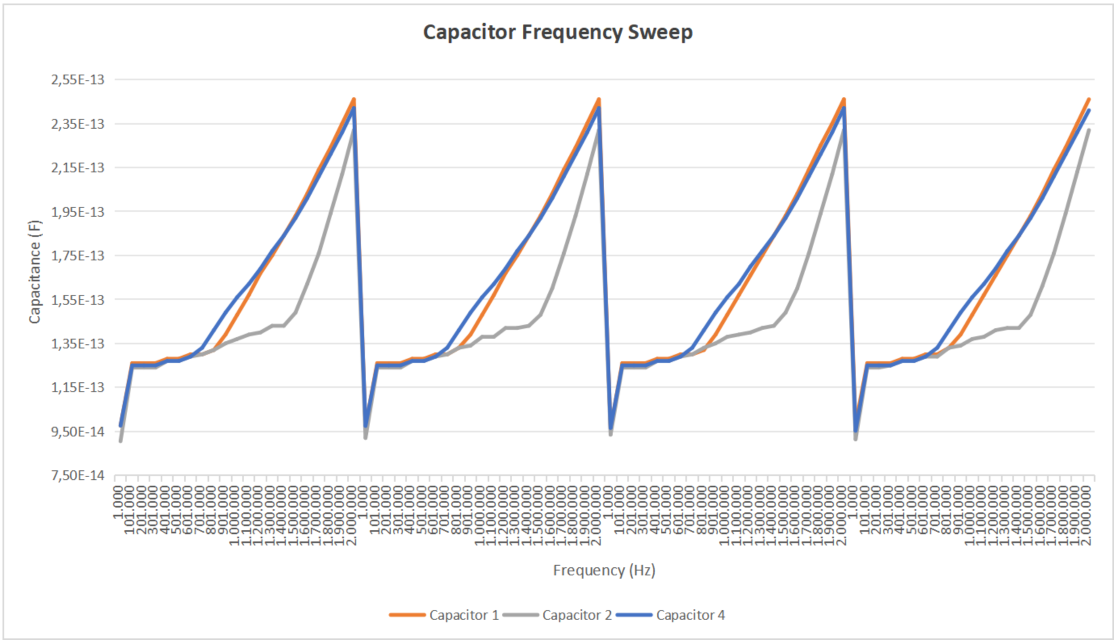


**Figure 6.12:** Capacitors Frequency sweep

As it is clearly visible, the range of values in which the measured capacitance oscillates is on the order of hundreds of femtofarads, significantly distant from the expected capacitance. This is likely due to the aforementioned fabrication issues with the electrodes and noise factors arising from contacts and the setup, which became even more predominant in the measurement of capacitance.

# Chapter 7

# Conclusion

As previously mentioned at the beginning of this work, this project is part of a larger one aimed at creating a cardiac patch. Since the project had just started, there was a need to develop and model each component from scratch, and this is where my work fits in. The ultimate goal of my work was to develop a tool for the characterization of biodegradable electrodes that will be part of the aforementioned cardiac patch. The objective can be considered achieved as the developed setup allows for stretching and measurement of the electrical parameters of the DUTs as the applied strain varies. The setup is highly flexible as it enables various tests to be performed while varying a multitude of parameters with good resolution.

That being said, there are still some limitations and areas for improvement in order to achieve even better performance. As shown in Chapter 5, when pushing the system to its limits and measuring data with very high precision (fraction of a millimeter), some errors occur during sampling. These errors are not irreversible and do not significantly affect the measurement overall, but they do affect the visualization of the data through the LabVIEW interface to some extent. In order to improve this aspect, a position sensor can be added to the system. Some attempts have already been made in this regard, but the sensor used was not precise enough for the purposes of this project.

Regarding the developed electrodes, these were not part of any specific project but were produced based on my proposal. As mentioned earlier, the purpose of this additional work was to compare the data measured from non-biodegradable electrodes made of PDMS gold with biodegradable electrodes composed of iron nanowires. Unfortunately, due to various complications during fabrication, this second objective was not achieved. However, beyond this, the fabrication and tests conducted on the gold electrodes were valuable for testing the setup in a real-use case scenario and for taking the first steps towards the characterization of flexible electronic components.

# Bibliography

[1] Manuel Franco, Richard S. Cooper, Usama Bilal, and Valentín Fuster. «Challenges and Opportunities for Cardiovascular Disease Prevention». In: The American Journal of Medicine,vol. 124 (2011), pp. 95–96 (cit. on p. 1).

[2] Xiaoyu Jiang et al. «A Bi-Layer Hydrogel Cardiac Patch Made of Recombinant Functional Proteins». In: Advanced Materials, vol.34 (2022), pp. 1–2 (cit. on p. 1).

[3] Ron Feiner, Leeya Engel, Sharon Fleischer, Maayan Malki, Idan Gal, Assaf Shapira, Yosi Shacham-Diamand, and Tal Dvir. «Engineered hybrid cardiac patches with multifunctional electronics for online monitoring and regulation of tissue function». In: Europe PMC Funders Group Author Manuscript, vol.15 (Sept. 2022), pp. 1–6 (cit. on p. 2).

[4] Jia Liua et al. «Intrinsically stretchable electrode array enabled in vivo electrophysiological mapping of atrial fibrillation at cellular resolution». In: PNAS, 117 (May 2020), pp. 14769–14772 (cit. on p. 2).

[5] Yuhao Liu, Matt Pharr, and Giovanni Antonio Salvatore. «Lab-on-Skin: A Review of Flexible and Stretchable Electronics for Wearable Health Monitoring». In: ACS Nano 2017, vol. 11 (Sept. 2017), pp. 9614–9619 (cit. on p. 5).

[6] Xin Tang, Yichun He, and Jia Liu. «Soft bioelectronics for cardiac interfaces». In: Biophysics Rev. 3 (Jan. 2022), pp. 3, 7 (cit. on p. 6).

[7] Hanjun Ryu et al. «Materials and Design Approaches for a Fully Bioresorbable, Electrically Conductive and Mechanically Compliant Cardiac Patch Technology». In: Advanced Science, vol. 10 (Oct. 2023), p. 10 (cit. on p. 7).

[8] John A. Rogers, Takao Someya, and Yonggang Huang. «Materials and Mechanics for Stretchable Electronics». In: Science, vol.327 (Oct. 2014), pp. 1605–1606 (cit. on p. 7).

[9] Wei Yuan, Xinzhou Wu, Weibing Gu, Jian Lin, and Zheng Cui. «Printed stretchable circuit on soft elastic substrate for wearable application». In: Journal of Semiconductors,vol.39 (Jan. 2018), pp. 1–5 (cit. on p. 8).

[10] Nitin Kumar Singh, Kazuto Takashima, and Shyam S. Pandey. «Fabrication, characterization and modelling of the fabric electrode-based highly stretchable capacitive strain sensor». In: Materials Today Communications, vol.32 (July 2022), p. 2 (cit. on p. 8).

[11] Amer Abdulmahdi Chlaihawia, Binu Baby Narakathua, Sepehr Emamiana, Bradley J. Bazuina, and Massood Z. Atashbara. «Development of printed and flexible dry ECG electrodes». In: Sensing and Bio-Sensing Research, vol.20 (May 2018), p. 3 (cit. on p. 8).

[12] F.J. Jimenez Romero, Jose R. Gonzalez-Jimenez, Felix García-Torres, Alvaro Caballero, and F.R. Lara Rayai. «A novel testing equipment based on Arduino and LabVIEW for electrochemical performance studies on experimental cells: Evaluation in lithium-sulfur technology». In: Measurement, vol.224 (Nov. 2024), pp. 2–3 (cit. on p. 8).

[13] Vishal Nayyar, K. Ravi-Chandar, and Rui Huang. «Stretch-induced stress patterns and wrinkles in hyperelastic thin sheets». In: International Journal of Solids and Structures, vol. 48 (Sept. 2011), p. 3472 (cit. on p. 10).

[14] *eTrack SERIES LINEAR STAGEdata sheet*. Newmark system inc. (cit. on p. 15).

[15] *Stepper motor driver TB6600 data sheet*. Sorotec (cit. on p. 17).

[16] *Agilent E4980A Precision LCR Meter data sheet*. Agilent technologies (cit. on p. 25).

[17] Mike McCauley. *Stepper motor library*. 2024. URL: `https://www.arduino.cc/reference/en/libraries/accelstepper/` (cit. on p. 29).

[18] David Austin. «Generate stepper-motor speed profiles in real time». In: EE Times-India (Jan. 2005), pp. 1–3 (cit. on p. 30).

[19] *LabVIEWTM Real-Time 1 Course Manual data sheet*. Austin, Texas: National Instruments (cit. on p. 57).

[20] Namsun Chou, Soonki Yoo, and Sohee Kim. «FABRICATION OF STRETCHABLE AND FLEXIBLE ELECTRODES BASED ON PDMS SUBSTRATE». In: MEMS 2012 (Feb. 2012), pp. 247–248 (cit. on p. 85).

[21] Florian Fallegger, Alix Trouillet, Florent-Valéry Coen, Giuseppe Schiavone, and Stéphanie P. Lacour. «A low-profile electromechanical packaging system for softto-flexible bioelectronic interfaces». In: APL Bioeng., vol. 7 (Aug. 2023), p. 2 (cit. on p. 86).

[22] Ines Miranda, Andrews Souza, Paulo Sousa, Joao Ribeiro, Elisabete M. S. Castanheira, Rui Lima, and Graca Minas. «Properties and Applications of PDMS for Biomedical Engineering: A Review». In: Journal of Functional Biomaterials, vol. 13 (Dec. 2021), pp. 2–4 (cit. on p. 88).

[23] Flaminio C. P. Sales, Ronaldo M. Ariati, Verônica T. Noronha, and João E. Ribeiro. «Mechanical Characterization of PDMS with Different Mixing Ratios». In: Procedia Structural Integrity, vol. 37 (2022), p. 384 (cit. on p. 88).

[24] Hao Zhang, Tingting Zhao, and Lunchao Zhong. «Design and fabrication of electrode array on curved surfaces for wearable electronics». In: 2023 International Conference on Applied Physics and Computing (ICAPC) (2023), pp. 559–561 (cit. on p. 90).

[25] Dhayalan Shakthivel, Nitheesh M. Nair, and Ravinder Dahiya. «Nanowires-Based Stretchable Strain Sensor for Wearable Applications». In: IEEE Sensor Council, vol. 7 , NO. 6 (June 2023), p. 2 (cit. on p. 90).

[26] Rui Li, Xin Gou, Chul Hee Lee, Haibo Ruan, Xiaojie Wang, Zhihao Zhou, Xin Huang, Zhongbang Liu, and Ping-an Yang. «Fe NWs/CNT/PUS composite constructed rigid-flexible coupling 3D porous structure with highly linear response and large strain for strain sensor». In: Sensors  Actuators: A. Physical vol. 353 (Sept. 2023), p. 2 (cit. on p. 94).

[27] Long-Fei Wang, Jing-Quan Liu, Bin Yang, and Chun-Sheng Yang. «PDMS-Based Low Cost Flexible Dry Electrode for Long-Term EEG Measurement». In: (Sept. 2012), p. 2900 (cit. on p. 97).

[28] Chin-Teng Lin, Lun-De Liao, Yu-Hang Liu, and Bor-Shyh Lin I-Jan Wang. «Novel Dry Polymer Foam Electrodes for Long-Term EEG Measurement». In: IEEE TRANSACTIONS ON BIOMEDICAL ENGINEERING, VOL. 58, NO. 5 (May 2011), p. 1202 (cit. on p. 97).