POLITECNICO DI TORINO

Master's Degree course in Electronics Engineering

Master's Degree Thesis

Design and program implementation of a three axis measuring machine control system

Supervisors
Prof. Gianfranco Genta
Dr. Marco Pisani

Candidate
Andrea Giura
Student ID: 301253

Academic year 2024/2025

Summary

The aim of this thesis is to update an existing interferometric three-axis measuring machine, used for measuring diameters and optical scales, by completely re-designing and implementing a new control system. The project involves the integration of new motors and drivers, along with the development of a C++ software program to manage the instrumentation. The upgraded machine features an advanced movement system and instruments for reading environmental parameters, which are crucial for calculating the refractive index of air for the correction of the laser wavelength and for compensating thermal errors. By incorporating these innovations, this research aims to minimize uncertainty in both positioning and interferometric measurements of the samples, thereby enhancing the reliability, traceability to the International System of Units (SI) and precision of the measurements. This thesis will explore the challenges encountered during the modernization process, including the selection of appropriate components and the calibration of the system. Additionally, it will present the methodologies employed to ensure accurate data acquisition and processing. Finally, this work aims to optimize the performance of the renewed measuring machine, improving the quality and the timing of the calibration service provided and opening the way to novel precise geometric measurements in various research applications (e.g. semiconductor, automotive, energy, etc.).

Acknowledgements

Desidero esprimere la mia più sincera gratitudine a coloro che hanno reso possibile il raggiungimento di questo importante traguardo.

Un ringraziamento va al Professor Genta, per la preziosa guida e il supporto durante la stesura di questa tesi. Desidero esprimere la mia gratitudine anche al Dottor Pisani, per avermi seguito con professionalità nelle fasi di progetto.

Un ringraziamento speciale alla mia famiglia. A mio padre, per il suo supporto e per aver sempre creduto in me. A mia sorella, per aver condiviso ogni momento. A mia madre, che mi ha spinto ad intraprendere il percorso universitario senza insistenza e, finchè ha potuto, mi ha supportato nelle scelte e che, anche se da lontano, continua a farlo.

Ringrazio di cuore i miei amici, per essere stati una fonte inesauribile di supporto e per avermi sempre offerto momenti di spensieratezza e divertimento al di fuori del contesto universitario.

La mia gratitudine va anche ai miei colleghi di lavoro. La loro comprensione e disponibilità sono state preziose, permettendomi di gestire gli impegni tra lavoro e università, concedendomi flessibilità quando avevo bisogno di dedicare tempo allo studio.

Grazie a tutti.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Precision measurement of linear dimensions is a critical aspect in various scientific and industrial applications, particularly in fields such as semiconductor manufacturing [1], automotive engineering [2], and energy production [3]. Located in INRiM (Istituto Nazionale di Ricerca Metrologica) laboratory, is the Moore interferometric three-axis measuring machine (Figure 1.1), this instrument has been key in metrological research and calibration services for decades, providing high-accuracy measurements essential for ensuring the quality and reliability of linear samples [4].

The Moore machine was acquired by INRiM in the 1970s, marking the beginning of its long-standing role in precision metrology. It is located in an underground building designed ad hoc to minimize environmental disturbances, ensuring that the machine operates under optimal conditions. The laboratory is equipped with a temperature and humidity control system, which is crucial for maintaining the stability of the measurement environment [5].

Initially, this machine was manually operated, requiring trained technicians to perform measurements through a series of intricate adjustments and calibrations. This manual operation not only made the process long and time-consuming, but also introduced a significant potential for human error [6]. Operators had to rely heavily on their expertise and experience to achieve accurate results, which could vary based on individual handling and technique. As a result, the measurement process was often slow and labor-intensive, limiting the machine's efficiency. The challenges associated with manual operation underscored the need for automation and modernization, this led to subsequent developments that would enhance the machine's capabilities and usability.

In the late 1970s to early 1980s, the Moore machine was updated with an interferometer and an HP 85 computer [7] for acquiring environmental parameters and interferometer data. This marked the beginning of its transition towards automation.

In 1990, the machine underwent a significant upgrade with the addition of optical scales and a control program written in Turbo Pascal [8] on an Olivetti 386 [9]

Figure 1.1: Moore measuring machine in the INRiM laboratory.

computer running DOS 3.31. This upgrade allowed for fully automated measurements, retaining the original direct current motors but adding the capability for control via console, joystick, and remote control. The program implemented is still in use and it lacks a graphical user interface (GUI) and is not user-friendly, presenting challenges for operators who require a more intuitive and efficient means of interacting with the machine. The absence of a modern interface has hindered the machine's usability and so it is now a priority to perform a comprehensive redesign of its control system.

By 1995, the DC motors were replaced with stepper motors, further enhancing the machine's precision and control.

In 2003, the machine was equipped with a digital microscope head for measurement purposes, adding the functionality to calibrate optical scales.

In 2010, significant advancements have been made, particularly through the integration of a multiple mirror setup into the existing laser interferometer system

[10]. This innovative approach aims to address the challenges posed by small deformations in the measurement machine, which can affect the accuracy and reliability of the measurements obtained. The differential measurement approach not only enhances the precision of the measurements but also increases the system's robustness against environmental fluctuations. This is particularly important in high-precision applications where even the slightest deviation can lead to significant errors. The integration of the multiple mirror setup allows for real-time compensation of these deformations, ensuring that the measurements remain consistent and reliable.

In 2013, procedures for measuring threads and screws were introduced [11], expanding the range of applications for the Moore.

The Moore machine is primarily used for the calibration of linear samples, including sphere radii, diameter samples [12], linear gauges, and optical scales. These measurements are crucial for customers who rely on precision in their products and processes. As such, the machine must exhibit good reliability and maintain low error margins in positioning [13] to ensure that the calibration results are both accurate and traceable.

In light of these considerations, the aim of this thesis is to modernize the Moore machine by completely redesigning and implementing a new control system. The project involves the integration of new motors and drivers, alongside the development of a robust C++ software program to manage the instrumentation effectively. The upgraded machine will feature an advanced movement system, enhancing its operational capabilities, speed and allowing for more precise control over the measurement process.

The Moore machine is equipped with advanced instruments designed to read environmental parameters, which play a crucial role in ensuring precise laser interferometer readings. These parameters include temperature, humidity, and atmospheric pressure, all of which can significantly influence the refractive index of air. By accurately measuring these environmental conditions, the system can calculate the refractive index with precision, allowing for the necessary corrections to be applied to the laser wavelength during measurements. This correction is vital, as even minor fluctuations in environmental conditions can lead to substantial errors in the calculated dimensions of the samples being measured [14].

In the next chapters of this thesis, we will delve into the design and implementation of the measurement system, beginning with a detailed examination of the mechanical components, including the continuous screw system of the axes and the specifications of the stepper motors chosen, as discussed in Chapter 2. This section will cover motor drivers, microcontroller specifications, command handling and square wave generation. We will discuss the various position measuring instruments employed in the system, such as laser interferometers, optical encoders, sensors, and the integration of a Nilox webcam for the definition of the reference systems in Section 2.5. The C++ Moore controller [15] section will introduce the programming objectives and the implementation of acceleration ramps, coordinate

system definitions using OpenCV, and the use of multithreading for simultaneous axis movement and environmental parameter monitoring in Section 2.7. Furthermore, we will address the need of measuring environmental parameters, including the calculation of the air refractive index for wavelength correction, and the communication interfaces for the thermometer, barometer, and hygrometer in Section 2.6. The graphical user interface section will outline the essential controls, discussing the libraries considered for the project, including Clay, Raylib, and WxWidgets, which was selected for its advantages in Section 2.8. Finally, we will present the testing and uncertainty budget chapters, where we will calculate the positioning and measurement uncertainties using the GUM and Monte Carlo methods, providing a comprehensive understanding of the system's performance and reliability in Chapters 3 and 4. The thesis will conclude with a summary of the findings and reflections on the project's outcomes.

1.1 Moore measuring machines

The Moore Special Tool Co. Inc. [16] was founded in 1924 by Richard F. Moore, who began working in the milling department at the Remington Arms Company at the age of 18 thanks to his uncle Ira Moore, who was a foreman there. Moore worked for six years in various shops, eventually landing in the toolroom of the Singer Sewing Machine Company for three years before starting his own business.

Figure 1.2: Moore company logo.

In 1924, Moore started his business with a small collection of used machines: a Hendey Lathe, a Brown & Sharpe Miller, a Producto Drill, and a LeBlond Universal Grinder [17]. His only new machine was an American Shaper. In 1926, Bill Angell joined the company and worked as Moore's "right-hand man" until 1957. By 1930, the company grew to 12 employees and eventually moved to a larger space owned

by Rudolph Bannow and Magnus Dannow, the future designers and manufacturers of the Bridgeport milling machine.

The Moore Company later partnered with Dallas Optical Systems to build ultra-precision, computer numerical controlled (CNC) diamond-turning machines. These machines, such as the M-40, were used to create precise contours on large telescope mirrors and shafts, the Moore Nanotechnology Systems Company was established to produce these machines.

In 1957 the M-18 (No. 3) measuring machine (Figure 1.3) was produced, the machine is composed of a base, a table, and a cross slide assembly. Table movement, longitudinal and transverse, is made by means of precision lead screws. These are operated, on the standard machine, by handwheel and read on large dials graduated to 5.0 μm, read against verniers to 0.5 μm. As optional equipment, the handwheel could be replaced by a motorized drive unit. In addition electrical limit switches are present to shut off power if the table reaches the end of its travel.

For the Z axis, the vertical movement is done in two stages. The first stage is a coarse adjustment made by manually moving the entire Z assembly up or down along the vertical column. The second stage is a fine adjustment achieved through a precision gear, which is operated by a handwheel that moves a cylinder placed inside the Z assembly. This fine adjustment allows for precise control of the vertical position.

1.2 Normal machine operation

In this section we will discuss the typical machine setup and operation to perform a measurement, the goal is to explain why the modernization of the machine control system is really necessary and what are the current problems that are inherent to the old machine implementation.

The machine is primarily used to measure one dimensional feature samples such as diametral samples, optical scales, spheres and linear encoders. A measurement is obtained by combining the reading of the interferometer, mounted on the x-axis, with the reading of the cary (LVDT sensor), mounted on the z-axis but sensible to the x displacement of the probe. In practice this means that the machine must make contact between the sample side and the probe ball, read both instruments and repeat the operation in all the positions of interest needed to obtain the measurement result.

To perform a measurement, the following steps are typically followed:

1. **Mounting the Sample**: The sample must be mounted firmly on the machine to ensure stability and accuracy during the measurement process. Any movement or instability can lead to significant errors in the measurement results.

2. **Calibrated Gauge Block**: A calibrated gauge block of 10 mm must be mounted next to the sample in line with the x-axis of the machine.

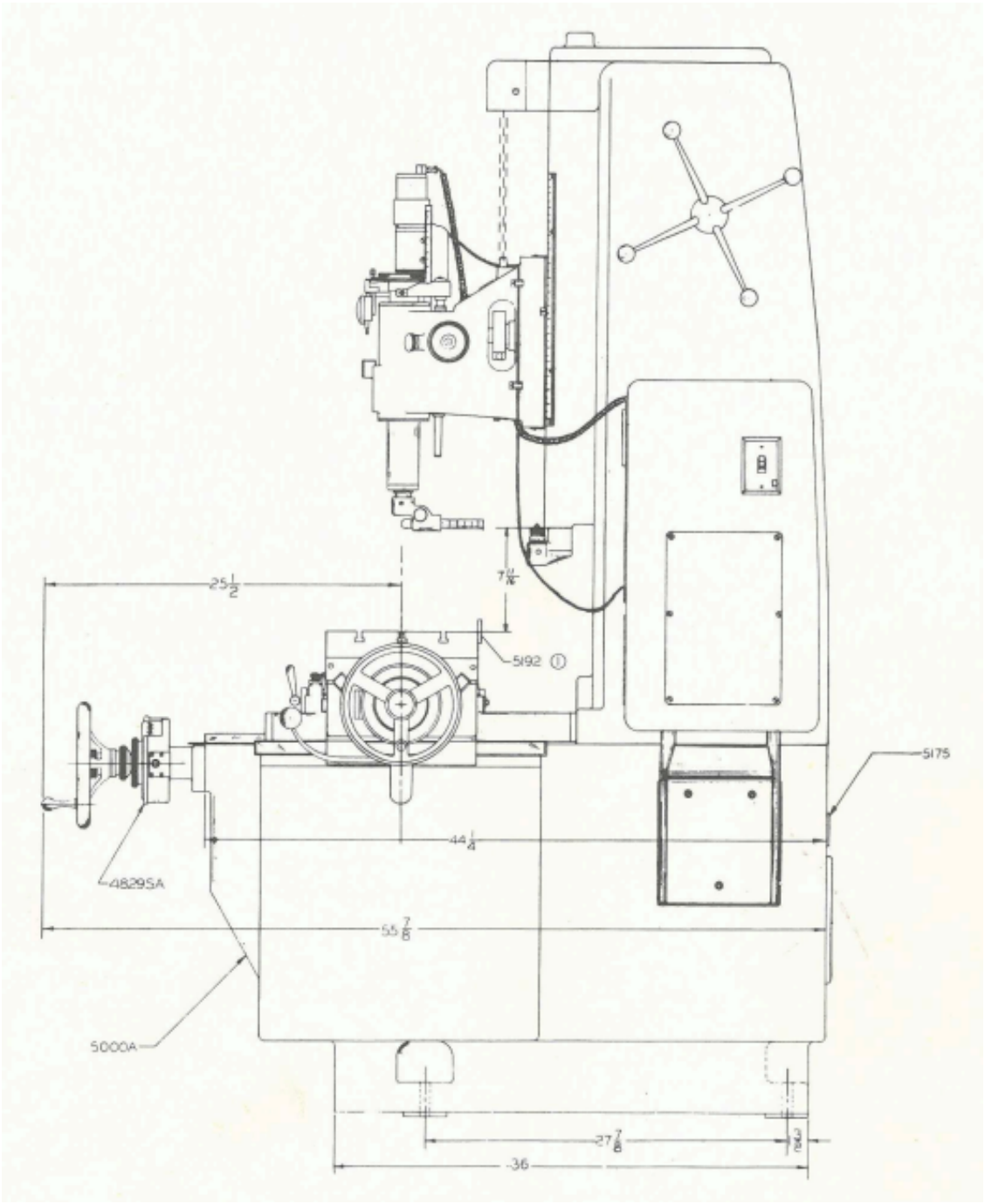


Figure 1.3: Moore model 3 outline and foundation layout.

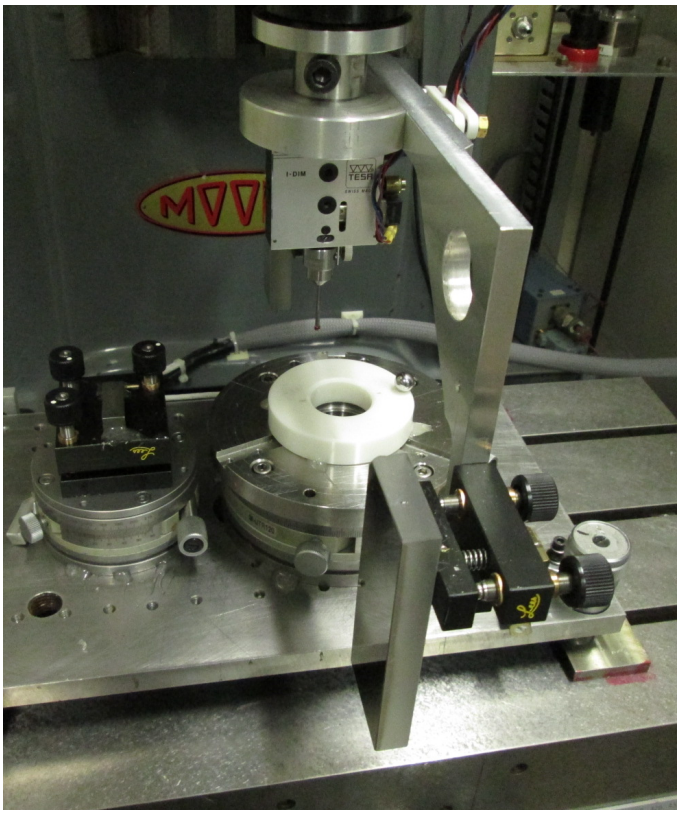


Figure 1.4: Setup for diameter measurement of a ring sample with 10 mm gauge block mounted next to the sample.

3. **Positioning the Spherical Steel Ball**: A spherical steel ball must be positioned on top of the sample (in case of a ring sample) to find the reference z starting quote of the cylinder side wall. This step is crucial for determining the initial position of the sample in the z-axis, which is necessary for accurate measurements.

4. **Manual Positioning**: Manually moving the machine, the operator makes contact with the block and the sample to tell the program that controls the machine where the pieces are situated. This step involves careful adjustments to ensure that the machine's sensors are correctly aligned with the sample and the gauge block.

5. **Setting Parameters**: The operator sets the parameters on the program, including the probe radius, sample nominal value, probe length, and other relevant settings. These parameters are essential for the machine to accurately interpret the measurements and perform the necessary calculations.

6. **Starting the Measurement**: The operator starts the measurement process.

The program begins by refining the position of the sample, scanning the side walls of the sample to find the maximum and measure the radius in the correct y and z position (Section 3.4).

7. **Measuring the Gauge Block**: The measurement proceeds by measuring the block to determine the correct ball radius of the probe and the deformation of the axle of the probe as explained in section 3.2. The steel axle, which is used to measure the sample, can deform due to the pressure exerted during the measurement process even if the force used in the contact is minimal. This deformation must be accounted for in the calculations to ensure accurate results.

8. **Measuring Environmental Parameters**: The environmental parameters of the laboratory, such as temperature, humidity, and atmospheric pressure, are measured. These parameters can significantly influence the accuracy of the measurements, so it is essential to account for them in the calculations.

9. **Measuring the Sample**: The sample is measured multiple times (n times) to ensure accuracy and repeatability. Repeating the measurements helps to identify any inconsistencies or errors and provides a more reliable result.

In conclusion, the current operation of the Moore machine involves a series of meticulous steps that require significant manual intervention and expertise. While the machine has undergone various upgrades over the years, the need for a modernized control system is evident. The existing control program lacks a user-friendly interface and automation capabilities, making the measurement process time-consuming and error-prone. By redesigning and implementing a new control system, the machine's efficiency, accuracy, and usability can be significantly enhanced, paving the way for more advanced metrological research and calibration services.

1.3 The INRiM framework

INRiM is the Italian National Metrology Institute [18], which is responsible for the realization and dissemination of the national measurement standards. It is a public research institute. INRiM plays a crucial role in ensuring the accuracy and reliability of measurements in various fields, including length, mass, time, temperature, electricity, and more. The institute collaborates with international organizations to maintain and improve measurement standards and practices, contributing to the global metrology community.

There are three main goals of the institute: the scientific research, the metrology services and the dissemination of the metrology culture. The scientific research is focused on the development of new measurement standards and techniques, while

the metrology services are aimed at providing calibration and testing services to industry and other organizations. The dissemination of the metrology culture (third mission or knowledge transfer) is focused on promoting the importance of metrology and its impact on society, industry, and research. This includes educational programs, workshops, and collaborations with other institutions to raise awareness about the significance of accurate measurements in various fields.

The work here presented is related to the first two goals, since the Moore machine is used to perform metrology services for accredited laboratories and to provide reliable measurements for research goals at the macro scale, for example measuring automotive or energy system mechanical parts.

The national metre standard is the basis for all length measurements in Italy and is maintained by INRiM, the Moore machine mounts an interferometer that is used to measure the length of the sample. This instrument is calibrated against the national standard to ensure the traceability chain in the measurement (Figure 1.5).

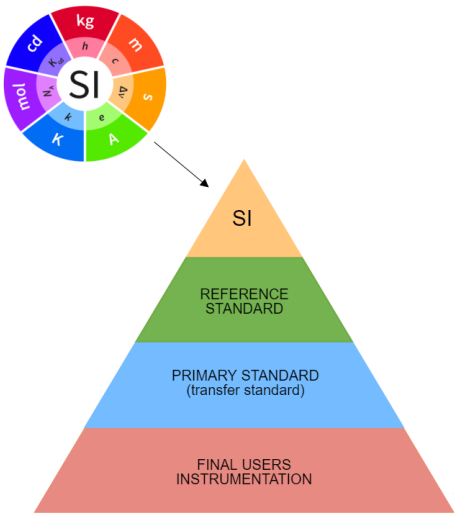


Figure 1.5: Traceability chain pyramidal representation. The moore machine is used to transfer the reference national standard to the final users primary standard (diametral sample).

Theoretical note 1 *Traceability is the property of a measurement result that allows it to be related to a national or international standard through an unbroken chain of comparisons, each contributing to the measurement uncertainty. This ensures that measurements are consistent and comparable across different laboratories and applications.*

Chapter 2

Design and implementation

The design of the control system begins with a comprehensive understanding of the requirements and constraints of the Moore machine. This chapter provides an in-depth look at the hardware components that form the backbone of the system, detailing their roles and interactions to achieve precise and reliable operation.

In Figure 2.1 is reported the block diagram of all the hardware components used for the Moore project for normal machine operation. Other external devices can be connected to allow for different measurements and expand the machine capabilities.

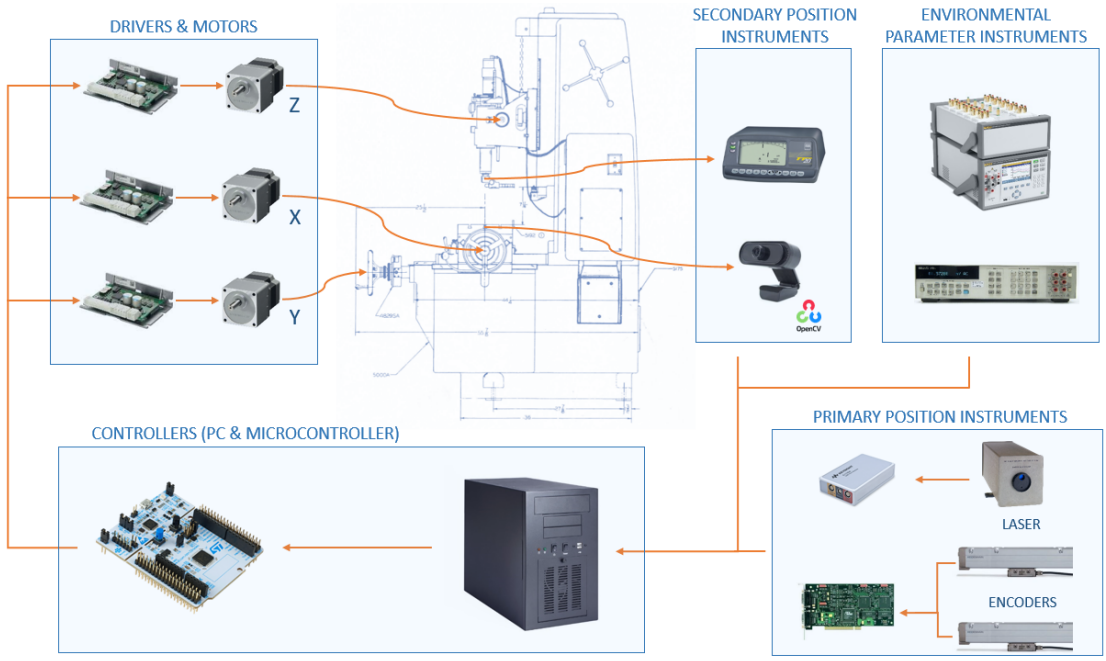


Figure 2.1: Block diagram of hardware components present for normal machine operation.

The system is composed of two main sections: the motion control section and the environmental parameter control section. Each of these sections plays a crucial role in ensuring the overall functionality and precision of the measurement system.

The motion control section is primarily responsible for the accurate positioning and movement of the machine. This section is further divided into two key components: position measurement instruments and actuators. The position measurement instruments are essential for providing real-time feedback on the location of the moving parts. These instruments can include laser interferometers, optical encoders, and other high-precision sensors that detect the position of the axes. The ability to utilize multiple position measurement instruments on the same axis allows for differential measurements.

On the other hand, the actuators consist of motor drivers, stepper motors, and a microcontroller. The motor drivers are responsible for controlling the power supplied to the motors, enabling precise control over their speed and position. The stepper motors, chosen for their high accuracy and repeatability, convert electrical signals into mechanical movement, allowing for fine adjustments in positioning. The microcontroller serves as the brain of the motion control system, processing input from the position measurement instruments and executing commands to the motor drivers based on this data. This closed-loop control system ensures that the motors are adjusted in real-time to maintain the desired position, compensating for any external disturbances or errors detected by the measurement instruments.

The second section of the system, dedicated to environmental parameter control, is equally important. This section monitors various environmental factors that can influence the measurement process, such as temperature, pressure, and humidity. By integrating sensors for these parameters, the system can make necessary adjustments to the measurements, ensuring that they remain accurate under varying conditions.

The PC plays a pivotal role in this entire setup, acting as the central hub that integrates both sections. It processes the data received from the position measurement instruments and environmental sensors, executing algorithms that determine the optimal motor commands. This feedback loop is critical for maintaining the precision of the system, as it allows for continuous monitoring and adjustment based on real-time data.

The following sections explain in detail all the components and how they are configured.

2.1 X, Y and Z axes

The axis system of the machine is designed to provide precise and controlled movement in multiple dimensions, facilitating accurate measurements and operations. The X and Y axes are similar in their construction and functionality, each featuring

a continuous screw with a constant pitch. This design allows for a movement of 3 millimeters for every complete rotation of the screw.

The system consists of several key components: a sliding carriage, linear guides, and the continuous screw (Figure 2.4). The sliding carriage is mounted on the linear guides, allowing for smooth and stable movement along the X and Y axes. The continuous screw is responsible for converting rotational motion into linear displacement, enabling the carriage to move accurately in response to the motor's commands. This configuration ensures that the X and Y axes can operate seamlessly, providing the necessary positioning for various measurement tasks.

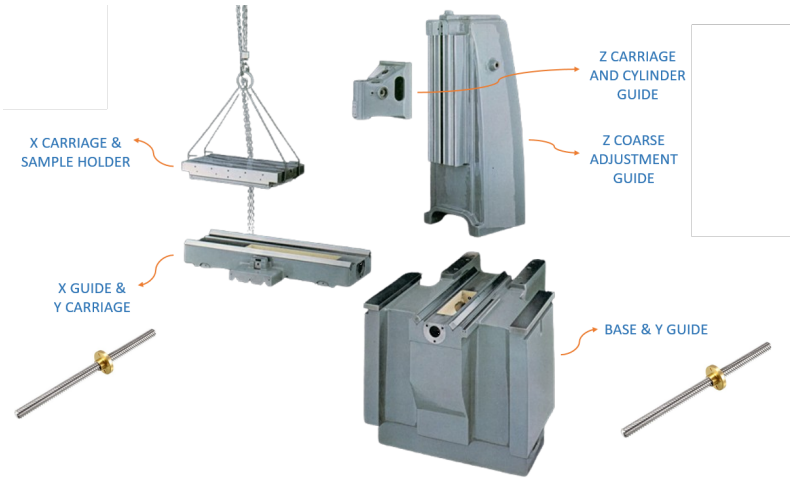


Figure 2.2: Composed view of the Moore machine parts, x, y and z carriages and slides.

In contrast, the Z axis presents a more complex system. It is designed to accommodate a mobile head that can traverse vertically, allowing for adjustments in height as needed. This mobile head is equipped with a free-moving cylinder that can be raised or lowered independently. The vertical movement of the cylinder is controlled by a knob located on the side of the machine, providing the operator with intuitive and direct control over the height adjustments.

The mechanism for controlling the vertical movement of the cylinder involves two belts, marked in red in Figure 2.3, that transmit power from the stepper motor to the knob. This configuration includes a transmission gear that increases the gear ratio, allowing for finer adjustments to the cylinder's height. By utilizing this gear system, the operator can achieve a greater degree of precision in height adjustments, as the increased gear ratio translates to smaller movements of the cylinder for each turn of the motor. This design effectively enhances the responsiveness of the Z axis, making it easier to position the measurement instruments accurately. To mount the gears a support is present on the machine to keep the belts in tension.

However, while this knob configuration offers advantages in terms of control, it also introduces some slack into the system. This slack can lead to a slight delay or

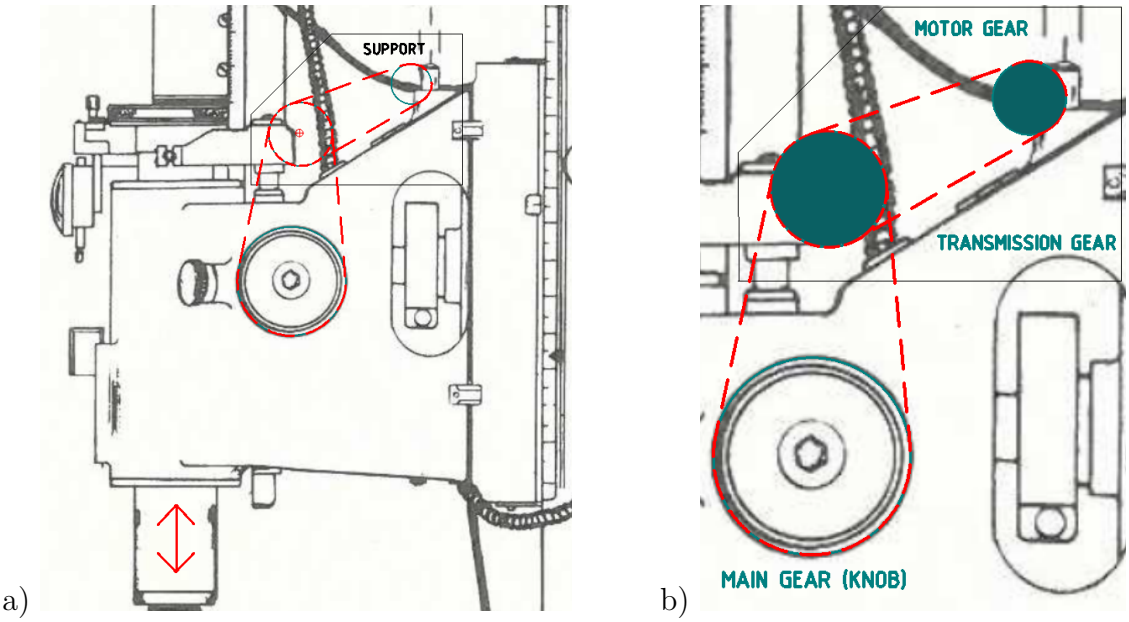


Figure 2.3: The Z axis movement assembly: (a) far view, (b) gear view.

imprecision in the cylinder's response to adjustments, which may be problematic in certain measurement scenarios where high accuracy is critical. Nevertheless, since the measurements are conducted parallel to the X axis, the impact of this Z axis slack is generally considered tolerable. The overall design ensures that, despite the potential for minor inaccuracies due to slack, the system remains effective for a wide range of applications, allowing operators to achieve reliable measurements without significant compromise in performance.

Figure 2.2 shows the assembly of the machine mechanical parts [16].

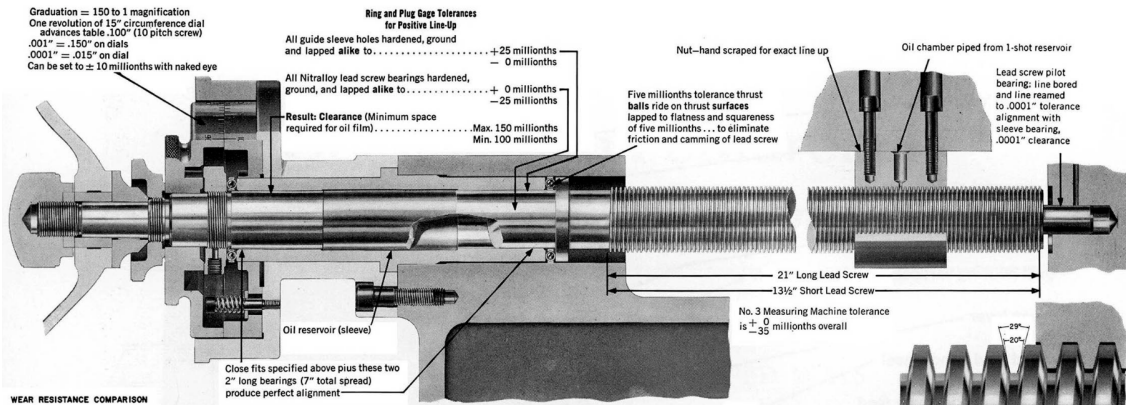


Figure 2.4: Internal screw of the Moore machine x and y axes.

2.2 Motors

For the design, we have chosen to use 5-phase stepper motors (Figure 2.5) to increase the precision of the system. A gear reduction ratio of 30:1 was selected to further enhance precision and reduce the torque load on the motor, as the rotation of the lead screws on the axes requires a considerable amount of torque.

Figure 2.5: Oriental Motors PKP564N28B2-TS30.

The specifications of the PKP564N28B2-TS30 motor [19] are reported in Table 2.1.

The speed-torque curve is shown in Figure 2.6.

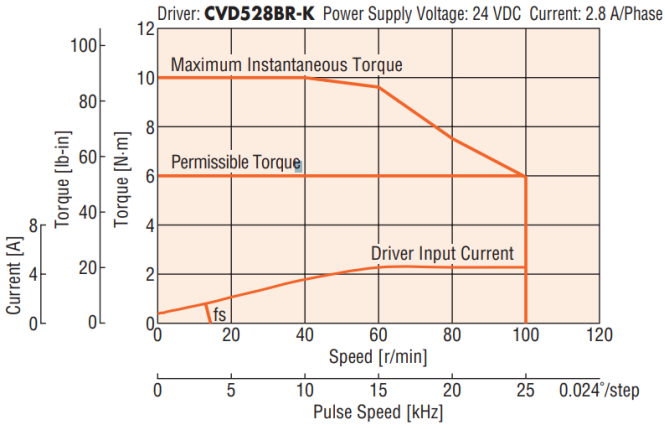


Figure 2.6: Speed-Torque Curve

Table 2.1: Specifications of the PKP564N28B2-TS30 motor.

Specification	**Value**
Output Shaft Diameter	10 mm
Motor Length	83 mm
Basic Step Angle	0.72º
Holding Torque	6 $N \cdot m$
Current per Phase	2.8 A/phase
Type	Geared
Shaft/Gear Type	Spur Gear (Offset Shaft)
Gear Type	Spur Gear, Low Backlash (up to 45 arc min), Offset Shaft
Gear Ratio (X:1)	30:1
Backlash	10 arcmin (0.17°)
Motor Connection Type	Flat Connector
Shaft	Double
Step Angle	0.024°
Connection Type	New Pentagon (Bipolar)
Lead Wires	5
Rotor Inertia	$140 \times 10^{-}7\ kg \cdot m^2$

2.2.1 Mounting and Adapter design

To facilitate the installation of new motors on the X and Y axes of the machine, it was necessary to create a custom 3D-printed adapter that effectively couples the motor shaft with the infinite screw shaft of the axes. This task was undertaken by a collaborator utilizing SolidWorks, a 3D modeling software, to design the adapter. The design process was not trivial due to the 30:1 reduction ratio of the motor shafts, which resulted in the rotation center of the motors being positioned higher than the axis shafts.

To achieve proper alignment between the two shafts, the holder for the motors was designed to position them lower than the centerline of the axis shafts. This strategic adjustment ensured that the motor shafts and the infinite screw shafts could align correctly, allowing for efficient power transmission and minimizing the risk of mechanical binding or misalignment during operation.

The 3D-printed adapter not only provided a robust coupling solution but also facilitated the integration of the new motors into the existing machine. The CAD design of the holder is shown in Figure 2.7.

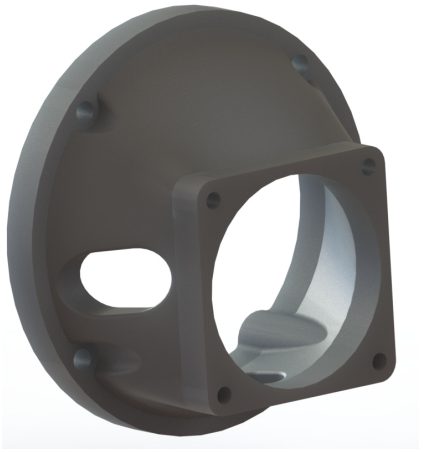
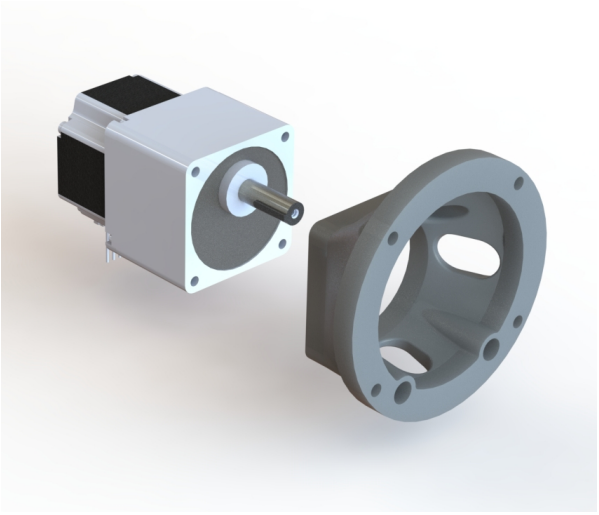
a)  b) 

Figure 2.7: a) The designed adapter and b) how it is interfaced with the motor.

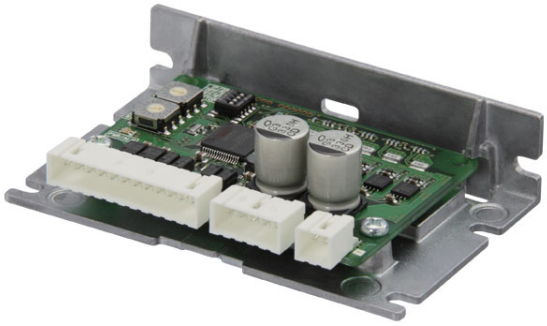


Figure 2.8: CVD528BR-K Driver.

2.3 Motor drivers

5-phase stepper motor drivers are specialized electronic circuits designed to control the operation of 5-phase stepper motors, which offer smoother motion and higher precision compared to their 2-phase counterparts. These drivers typically accept two primary inputs: a step signal and a direction signal. The step signal indicates when the motor should move to the next step, while the direction signal determines the rotational direction of the motor, allowing for precise control over its movement.

Many modern 5-phase stepper motor drivers also incorporate switching capabilities with microstepping functionality, which enables the motor to move in smaller increments than a full step. This microstepping capability enhances the resolution and smoothness of the motor's motion, improving overall performance in angular positioning of the axle.

Figure 2.8 shows the driver chosen for the control of the 5-phase motors.

Theoretical note 2 *Microstepping [20] is a technique used in stepper motor control that allows for finer resolution and smoother motion by dividing each full step of the motor into smaller increments. Stepper motors operate by moving in discrete steps, with each step corresponding to a specific angular movement. For example, a typical stepper motor might have 200 steps per revolution, meaning it moves 1.8 degrees with each step. In a 5-phase stepper motor, the motor has five coils, and the sequence in which these coils are energized determines the direction and position of the motor. Instead of moving the motor by a full step, microstepping allows the motor to move by fractions of that angle, such as 1/2, 1/4, 1/8, or even smaller increments. Instead of fully energizing a coil, the driver can apply varying levels of current to each coil, creating a smooth transition between steps. For example, to achieve a microstep between two full steps, the driver might energize one coil at 70% of its maximum current while energizing the adjacent coil at 30%. This results in a position that is not aligned with the full step but rather in between, allowing for finer control. Microstepping drivers often use sinusoidal control to achieve smooth motion.*

The driver [21] accepts in input several signals via the CN3 connector, the pinout is reported in Figure 2.9. The available input control signals are:

- Pulse (PLS): pulse signal that triggers a step (microstep if divided) of the motor
- Direction (DIR): TTL signal for setting the motor direction
- All windings off (AWO): TTL signal that switches on or off all the windings to enable or disable the motor, if the windings are turned off it is possible for the user to manually rotate the axis knob
- Chip select (CS): TTL signal that controls the subdivision of the step, two possible settings can be chosen. When the CS input is turned ON, the motor rotates at the predefined step angle, when the CS input is turned OFF, the motor rotates at the step angle set by the driver switch.

To set the step angle size a selector is present on the driver's board (Figure 2.10), the letter correspondence to the subdivisions are reported in the following table 2.2.

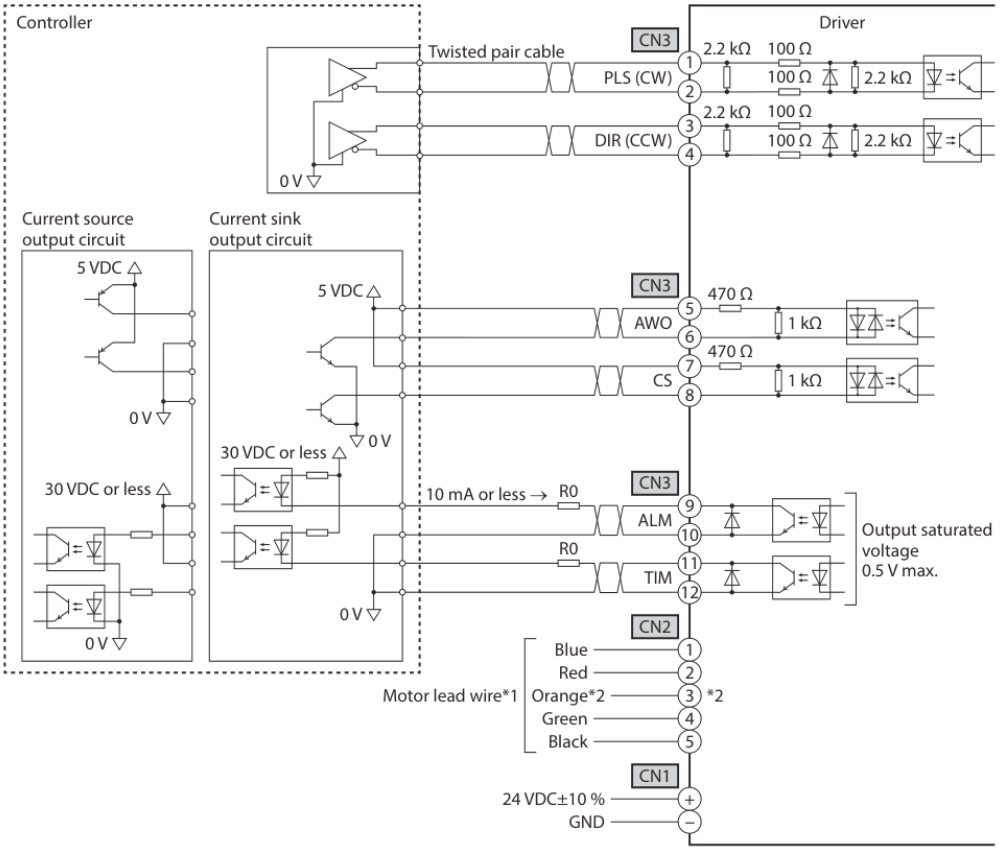


Figure 2.9: Driver connection scheme.

Table 2.2: Selector / division correspondence.

—	R2/R1 switch ON	
STEP switch	**Resolution (P/R)**	**Step angle**
0	500	0,72°
1	1000	0,36°
...	...	...
E	100000	0,0036°
F	125000	0,00288°

2.4 Microcontroller

STM32 Nucleo boards [22] are versatile development platforms that provide a powerful alternative to traditional Arduino boards [23]. While Arduino is well-known for its simplicity and ease of use, STM32 Nucleo boards offer enhanced performance and a wider range of features, making them suitable for more complex applications.

In Figure 2.11 is reported the microcontroller used for the Moore configuration.

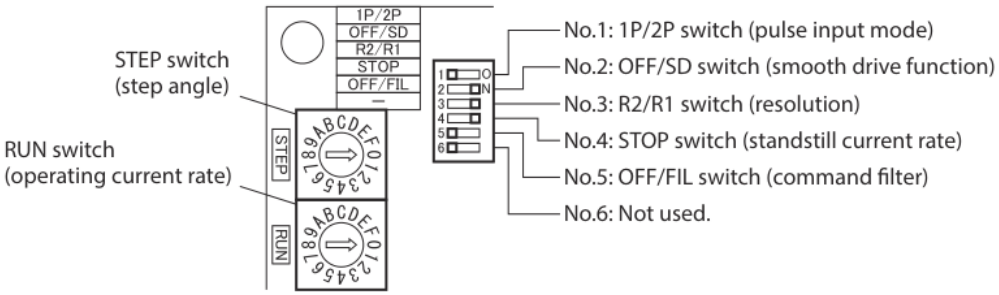


Figure 2.10: Driver switches and selectors for step, current and function setting.

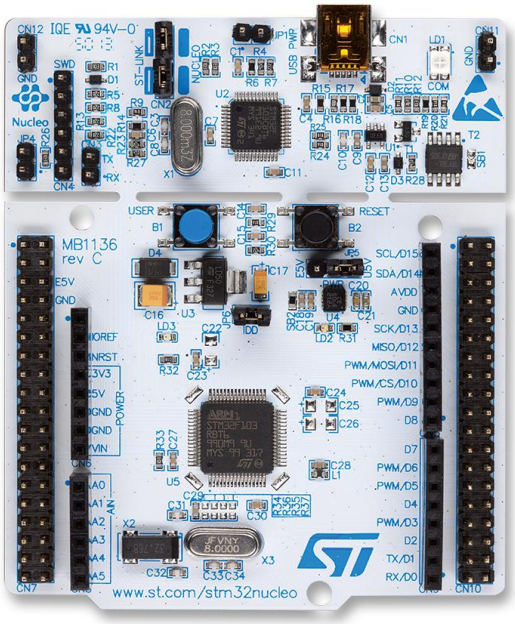


Figure 2.11: STM32 NUCLEO board.

These boards are built around the STM32 microcontroller family, which is based on the ARM Cortex-M architecture. This architecture allows for higher processing speeds, greater memory capacity, and advanced power management features. Nucleo boards come equipped with a variety of peripherals [24], including timers for precise timing control, NVIC (Nested Vectored Interrupt Controller) for efficient interrupt management, GPIO (General Purpose Input/Output) pins for interfacing with external devices, and USART (Universal Synchronous/Asynchronous Receiver-Transmitter) for serial communication. The versatility of STM32 Nucleo boards makes them ideal for a wide range of applications, including robotics, IoT

(Internet of Things) devices, automation systems, and embedded systems development.

In Figure 2.12 is reported the microcontroller configuration used for the Moore project to generate the signal needed by the motor drivers explained in the previous section.

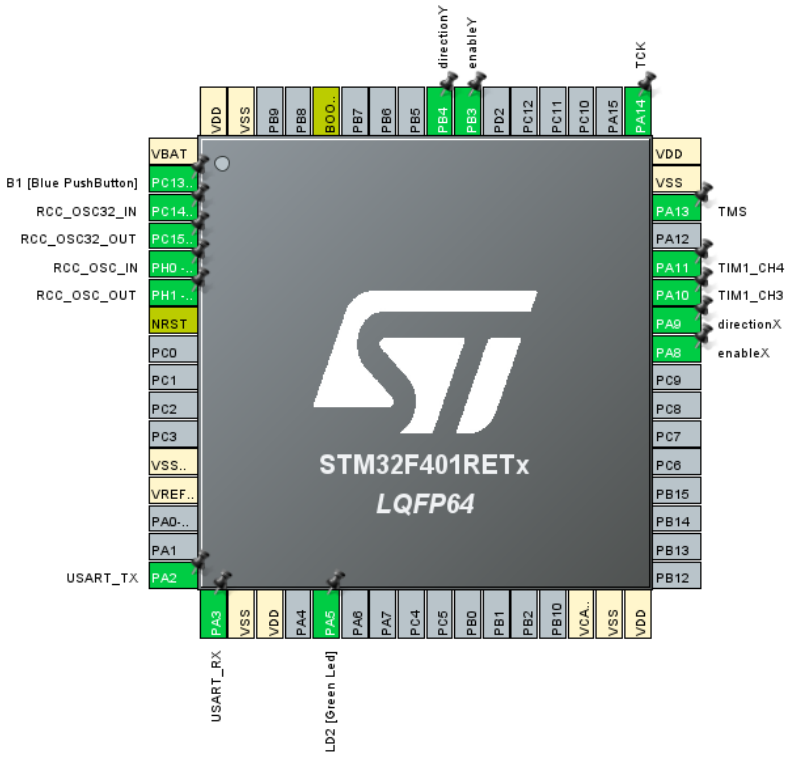


Figure 2.12: STM32 ioc, pinout and alternate function pins.

2.4.1 USART command handling

For the movement of the axes it is necessary that the PC coordinates the motor control with the reading of the position instrument present on the relative axis, to do this was used the RS232 protocol between the micro and the PC [25].

Theoretical note 3 *RS-232 is a standard protocol used for serial communication between devices, primarily in computer and telecommunications applications. It defines the electrical characteristics and timing of signals, as well as the physical interface and the format of the data being transmitted. RS-232 operates using a single-ended signaling method, where data is transmitted one bit at a time over a single wire. In an RS-232 communication setup, data is sent in frames, which*

consist of a start bit, a data byte (usually 7 or 8 bits), an optional parity bit for error checking, and one or more stop bits to indicate the end of the transmission. RS-232 supports various baud rates, which determine the speed of data transmission, and it can accommodate multiple devices through a point-to-point connection.

The configuration of the STM32 USART is shown in Table 2.3:

Table 2.3: STM32 USART configuration.

Parameter	**Value**
Baud Rate	9600
Word length	8
Stop Bits	1
Parity	None
Flow Control	None
Data direction	Rx and Tx
Oversampling	16 samples

The micro waits for each byte and generates an interrupt to call an interrupt subroutine to concatenate the bytes received until the reception of the 'CR' character. Once the command has been composed it is sent to the routines that parse the string and set the correct values to the timers and GPIO pins.

The commands for controlling the motor are in the following form:

$$'(x/y/z)(d/e)(b/f)nnnn\cdots n(CR)'$$

where:

- the first character indicates which axis the command refers to,
- the second indicates wether the windings of the motor should be enabled for motor control or disabled for freewheel of the axis and manual rotation,
- the third sets the direction of the movement 'b' if backward, 'f' if foreward,
- lastly a number of n digits represents the speed at which the motor is set instantly, the speed is not in measurable units but is a relative number ranging from minimum speed and maximum speed as explained in Section 2.4.2.

The command is then parsed from left to right and is used to modify the pins relative to the selected axis. This is done in code using a structure that represents the axis:

```
typedef struct
{
  uint8_t axisN;
```

```
  bool setV;

  uint16_t T;
  uint16_t MAXT, MINT;

  GPIO_TypeDef* axisEnableGPIO;
  uint16_t axisEnablePin;

  GPIO_TypeDef* axisDirectionGPIO;
  uint16_t axisDirectionPin;
} axisParams;

axisParams xaxis, yaxis, zaxis;
axisParams * axis = 0;  // active axis pointer
```

The `axis` variable is used to point to the current axis, while the `xaxis, yaxis, zaxis` variables are initialized with the specific GPIO and parameter values. Each time a new command is parsed the `axis` pointer is set to the desired axis variable, in this way we can change the correct pins when executing the USART ISR.

```
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
  // uart msg: axis, ena, dir, XXXXX, \0 -> 9 chars
  if (uart_buf != '\r') {
    uart_msg[msg_i] = uart_buf;
    msg_i++;
  } else {
    uart_msg[msg_i] = '\0';

    setAxis(uart_msg);
    msg_i = 0;
  }
  HAL_UART_Receive_IT(&hlpuart1, &uart_buf, 1);
}

void setAxis(const char * settings)
{
  switch (settings[0]) {
  case 'x':
    axis = &xaxis;
    break;
  case 'y':
    axis = &yaxis;
    break;
  case 'z':
    axis = &zaxis;
    break;
  }

  if (settings[1] == 'd') HAL_GPIO_WritePin(axis->axisEnableGPIO,
   axis->axisEnablePin, GPIO_PIN_SET);
```

```
31   if (settings[1] == 'e') HAL_GPIO_WritePin(axis->axisEnableGPIO,
      axis->axisEnablePin, GPIO_PIN_RESET);
32
33   if (settings[2] == 'b') HAL_GPIO_WritePin(axis->axisDirectionGPIO
      , axis->axisDirectionPin, GPIO_PIN_SET);
34   if (settings[2] == 'f') HAL_GPIO_WritePin(axis->axisDirectionGPIO
      , axis->axisDirectionPin, GPIO_PIN_RESET);
35
36
37   uint16_t speed = atoi(settings + 3);
38   axis->T = map(speed, MINspeed, MAXspeed, axis->MINT, axis->MAXT);
39 }
```

The speed setting is done using the `stoi` function from the standard library that enables the conversion from a string to an integer number, this number must be converted to a timing interval for the counter. How this is done is explained in the following Section 2.4.2. In order to make the `stoi` function work a '0' character is added to the assembled command as a NULL string terminator, as defined in the ISO C standard.

2.4.2 Square wave generation

To generate a square wave (step signal of the motor driver) using an STM32 microcontroller, we can utilize the Timer peripheral along with the GPIO (General Purpose Input/Output) pins. The process involves configuring the Capture/Compare Register (CCR), setting up the alternate function for the GPIO pin, and using an Interrupt Service Routine (ISR) to dynamically adjust the CCR value for the desired frequency.

Theoretical note 4 *Interrupts are signals that temporarily halt the normal execution of a program to allow the processor to respond to an event or condition that requires immediate attention. They serve as a mechanism for the microcontroller or processor to handle asynchronous events, such as input from a user, data arriving from a sensor, or a timer reaching a specific count. When an interrupt occurs, the current execution context is saved, and the processor jumps to a predefined location in memory where the interrupt service routine (ISR) is located. An interrupt service routine is a special function designed to handle the specific tasks associated with an interrupt. Within an ISR, the necessary actions are performed to respond to the interrupt, such as reading data from a sensor, processing input, or resetting a timer. After the ISR completes its execution, the processor restores the previous context and resumes the interrupted task, ensuring that the system continues to operate smoothly.*

The key steps in configuring the timer are:

- **Set the Prescaler**: The prescaler divides the input clock frequency to slow down the timer's counting speed.
- **Set the Auto-Reload Register (ARR)**: This register defines the period of the timer. This register must be set to the highest number possible in order to have the best resolution of the square wave frequency possible.

The timer counter continuously counts up from 0 to ARR, when it reaches ARR, it resets to 0 and the counting restarts. The output pin toggles based on the CCR value:

- **Counting**: The timer counts up at a fixed rate determined by the prescaler.
- **Matching**: When the counter value matches the CCR, the output pin toggles (if configured to do so) and the ISR is triggered.

To dynamically adjust the frequency of the square wave, an interrupt can be set up:

- **Enable Timer Interrupts**: Enable the interrupt for the timer, which will trigger when the timer counter matches the CCR value.
- **Set Interrupt Priority**: Use the NVIC (Nested Vectored Interrupt Controller) to set the interrupt priority to the highest level.

The ISR is executed every time the timer matches the CCR value and performs the following operations:

- **Calculate Next CCR Value**: Inside the ISR, the next CCR value can be calculated based on the desired frequency or duty cycle. For example, if an increase of frequency is needed, the ISR would decrease the CCR value.
- **Update CCR**: Write the new CCR value back to the CCR register to adjust the timing for the next toggle.

To calculate the T parameter used to update the CCR values we can proceed with the following reasoning. Motor speed must be limited between a maximum and a minimum value, it is therefore appropriate to define Equations 2.1.

$$\begin{aligned} T_{max} &= \frac{1}{f_{min}} \Rightarrow v_{min} \\ T_{min} &= \frac{1}{f_{max}} \Rightarrow v_{max} \end{aligned} \tag{2.1}$$

The microcontroller receives commands via serial interface, the microcontroller USART has been configured to receive data on 8 bits and generate an interrupt

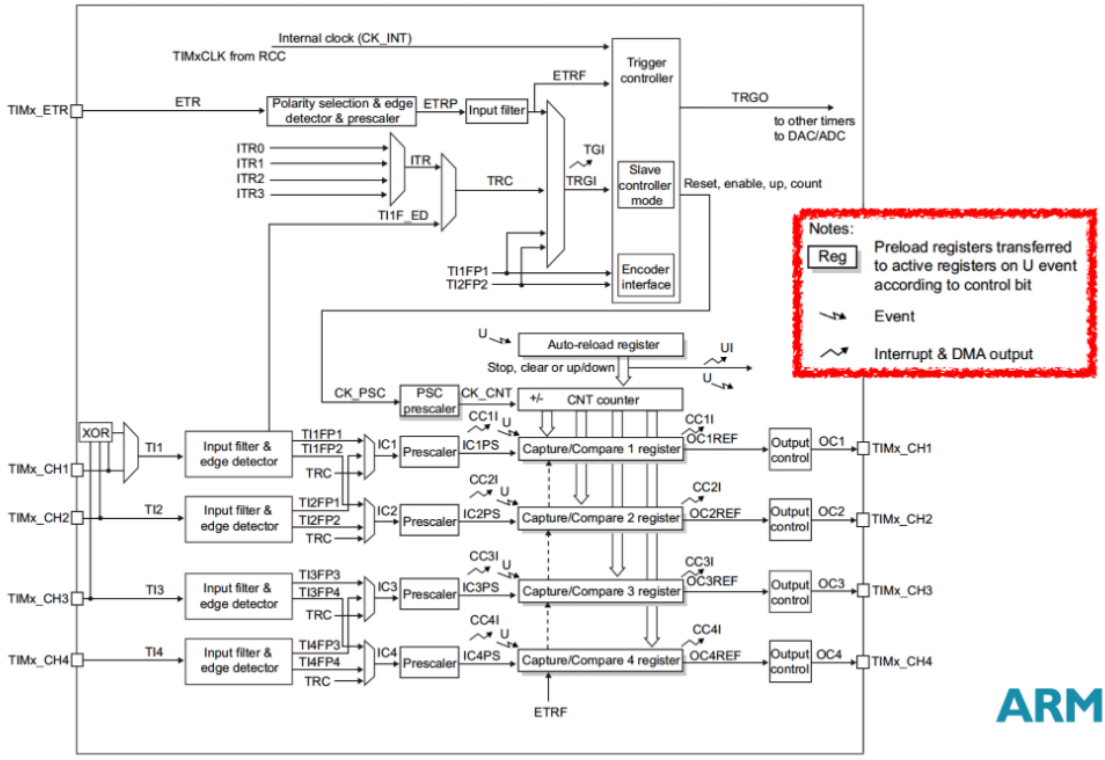


Figure 2.13: Internal register diagram of timer peripheral in STM32 microcontroller.

every time a byte is received. The motor speed has been mapped to values between D_{min} and D_{max}, it is possible to associate the corresponding value to T with Equations 2.2, 2.3:

$$f = \frac{1}{T} = (x-1)\frac{f_{max} - f_{min}}{D_{max} - D_{min}} + f_{min} = (x-1)\frac{\frac{1}{T_{min}} - \frac{1}{T_{max}}}{D_{max} - D_{min}} + \frac{1}{T_{max}} \tag{2.2}$$

$$T = \frac{1}{(x-1)\frac{1/T_{min} - 1/T_{max}}{D_{max} - D_{min}} + \frac{1}{T_{max}}} \tag{2.3}$$

It must be taken into account that T is represented as an integer on 16 bits, since the TIM1 CCR is itself a 16-bit register, this means that the formula must be modified to ensure that there are no data losses in the divisions. To do this it is sufficient to scale the numerator and denominator by a common factor, the optimal solution would be scaling by $T_{min} \cdot T_{max}$ but this value cannot be represented in a 16 bit variable so the final Equation 2.4 is only scaled by T_{max}.

$$T = \frac{1}{(x-1)\frac{1/T_{min} - 1/T_{max}}{D_{max} - D_{min}} + \frac{1}{T_{max}}} \cdot \frac{T_{max}}{T_{max}} = \frac{T_{max}}{(x-1)\frac{T_{max}/T_{min} - 1}{D_{max} - D_{min}} + 1} \tag{2.4}$$

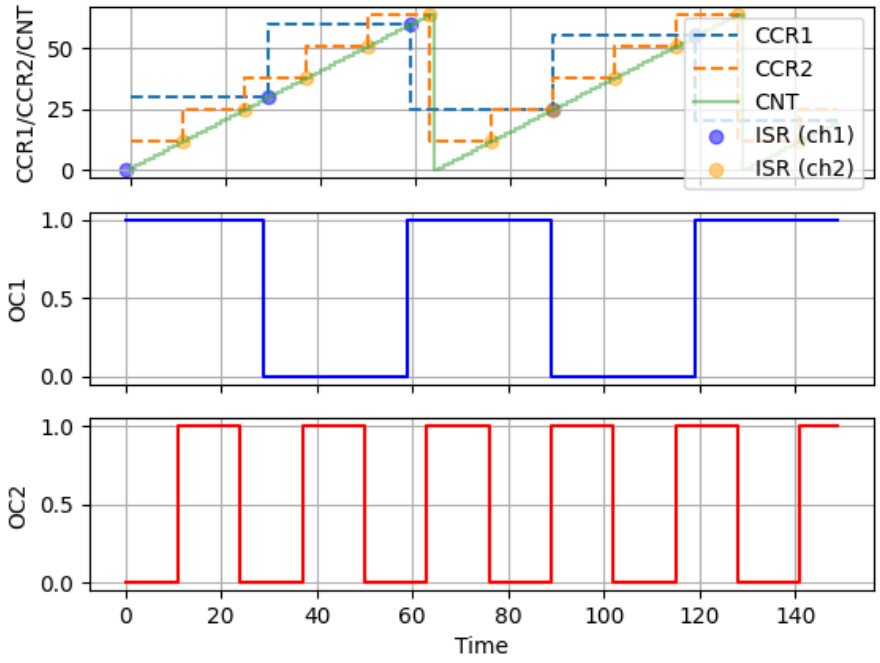


Figure 2.14: Counter value and CCR registers with double output square wave generation. The parameters of the simulation are: $T_1 = 30$, $T_2 = 13$, $CCR_{1start} = 0$ (The ISR is called immediately so it starts from T_1 and OC_1 is already set), $CCR_{2start} = 12$, $ARR = 64$.

Once the T parameter has been calculated the CCR value can be set as in Equation 2.5:

$$CCR_i = CCR_{i-1} + T \tag{2.5}$$

in the board firmware this update was implemented in the timer ISR.

```
void HAL_TIM_OC_DelayElapsedCallback(TIM_HandleTypeDef *htim) {
  if (htim -> Channel == HAL_TIM_ACTIVE_CHANNEL_2)
      __HAL_TIM_SetCompare(&htim1, TIM_CHANNEL_2,
    __HAL_TIM_GetCompare(&htim1, TIM_CHANNEL_2) + xaxis.T);

  else if (htim -> Channel == HAL_TIM_ACTIVE_CHANNEL_1)
      __HAL_TIM_SetCompare(&htim1, TIM_CHANNEL_1,
    __HAL_TIM_GetCompare(&htim1, TIM_CHANNEL_1) + yaxis.T);

  else if (htim -> Channel == HAL_TIM_ACTIVE_CHANNEL_3)
      __HAL_TIM_SetCompare(&htim1, TIM_CHANNEL_3,
    __HAL_TIM_GetCompare(&htim1, TIM_CHANNEL_3) + zaxis.T);
}
```

For TIM1 the CCR, ARR, CNT and T registers and variables are of `uint16_t` type which can hold values from 0 to 65535. Since we are summing continuously the next value to the CCR, it is possible for the CCR to exceed the maximum 16 bit range. However, this is not a problem for the following reasons:

- **Overflow Behavior**: When the CCR value exceeds 65535, it wraps around from 0 due to the nature of unsigned integers. For example, if the CCR value is 65535 and 1 is added, it will overflow and become 0. This behavior is similar to how the timer counter works.
- **ARR Configuration**: By setting the ARR to its maximum value (65535), we can ensure that the timer counter will also wrap around when it reaches this value. This means that both the CCR and the timer counter will wrap around similarly.

This means that using `uint16_t` variables it is possible to solve at once all the problems related to the overflow of the registers.

Since we have three motor drivers to control we need to generate in parallel three square waves independent to each other, we can utilize the different Capture/Compare channels (CCR1, CCR2, CCR3) available on the timer. Each channel can be configured to toggle the output pin at different intervals, allowing us to produce distinct square waves with varying frequencies. This approach is efficient and resource-saving, as it eliminates the need for multiple timers. If we had chosen to modify the Auto-Reload Register (ARR) instead of the Capture/Compare Register (CCR) to generate the square waves, we would have encountered limitations, as changing the ARR would affect the timing for all channels simultaneously. Consequently, we would have required three separate timers to achieve the desired functionality, complicating the design and increasing resource usage. By leveraging the CCR channels, we can effectively manage multiple square wave outputs with a single timer, simplifying our implementation while maximizing the capabilities of the STM32 microcontroller.

Half Period resolution

To move the stepper motor smoothy during the acceleration and deceleration phases, the square wave frequency must be adjusted in small increments. In fact if the frequency is changed too abruptly the motor will not be able to follow the changes and will lose steps or the speed change will be too abrupt and the motor will vibrate. The period resolution of the square wave is determined by the PSC register value, which sets the count frequency for the timer. The period resolution is calculated as in Equation 2.6:

$$\text{Period Resolution} = \frac{1}{CK\_INT/PSC} \tag{2.6}$$

where CK_INT is the internal clock frequency of the microcontroller.

A trade-off between resolution and frequency range must be considered when selecting the PSC value. To start the motor we need to achieve low frequency square waves, this can be a problem if the PSC value is too low, because to achieve the low frequency we are limited by the counter upper bound ARR. In fact the minimum frequency that can be achieved is given by the following Equation 2.7:

$$f_{min} = \frac{CK\_INT}{PSC \cdot CCR_{max}} = \frac{CK\_INT}{PSC \cdot ARR} \tag{2.7}$$

Given the two relations between PSC, Period Resolution and minimum frequency we can observe that the minimum frequency is inversely proportional to the PSC value, this means that to achieve low frequencies we need to set a high PSC value. On the other hand the Period Resolution is directly proportional to the PSC value, this means that to achieve a high resolution we need to set a low PSC value. The solution to this problem is to set the PSC value to the lowest possible value that still allows us to achieve the minimum frequency required to start the motor. In this way we can achieve a good resolution and the correct operation of the machine.

A better solution would be to use a timer with a higher ARR value, this has been done in the firmware of a test NUCLEO board with a STM32F446RE microcontroller but has not been implemented yet in the machine since it is a minor improvement to the system and it requires a redesign of the PCB connection described in the following section. The test board has been configured to use the TIM2 peripheral with a PSC value of 1, since the TIM2 operates on 32 bits the ARR value can be set to $2^{32} - 1$ and the minimum frequency can be achieved with a large CCR value but maintaining the highest resolution possible.

To perform the test a simple Python 3 script was implemented to send the commands to the microcontroller and control the motor, the script has a GUI to set the speed and the settings of the driver and was really useful to test the setup and to set the constants in the microcontroller firmware.

2.4.3 Printed circuit board

The printed circuit board (PCB) was designed using KiCAD [26], an open-source software for electronic design automation. The PCB was designed to connect the STM32 microcontroller to the drivers. The GPIO outputs used in alternate function mode to generate the step and direction signals for the drivers are used in open-drain mode, pull-up resistors have been added to the connection scheme to ensure the correct signal levels.

The PCB was designed with the following features:

- **STM32 Connection**: The PCB features a connector for the STM32 microcontroller, allowing for easy connection and communication between the microcontroller and the driver.

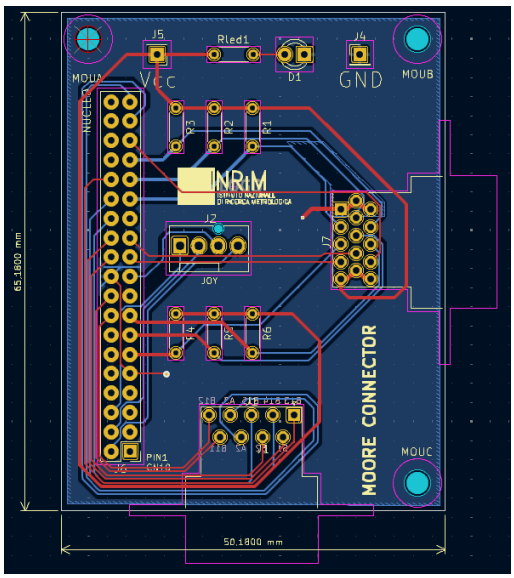
- **Driver Connection**: The PCB includes a connector for the three drivers, enabling the driver to receive signals from the microcontroller and control the stepper motors.

- **General purpose connector**: The PCB includes a general-purpose connector for connecting additional sensors or peripherals, providing flexibility for future expansions or modifications. This was included to exploit the majority of the I/O pins of the STM32 that are not used for the driver connection.

- **Compact Design**: The PCB is compact in size, the height of the board has been designed to be smaller than the microcontroller.

- **Ground plane**: The PCB features a ground plane to ensure proper grounding and reduce noise interference in the circuit, even if the frequency range of the signals is not high, the ground plane is a good addition to avoid any possible interference.

The PCB design (Figure 2.15) was then exported to a Gerber file format and sent to a PCB manufacturer for fabrication.

a) 

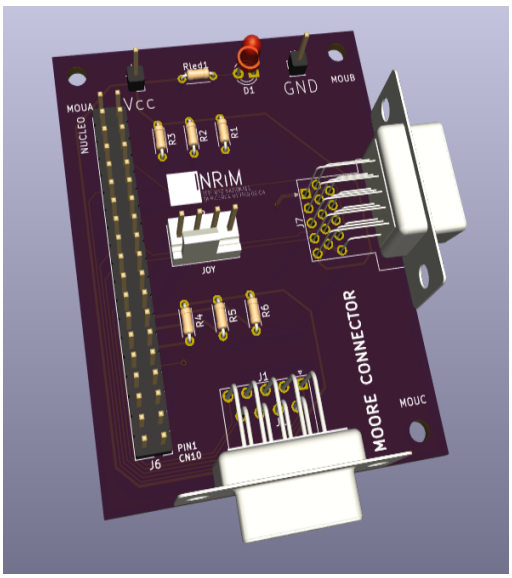
b) 

Figure 2.15: Printed circuit board for STM32 and driver connection. (a) KiCAD 2D top view, (b) 3D view.

2.5 Position measuring instruments

2.5.1 Laser interferometer

The Moore machine is equipped with a laser interferometer system that plays a crucial role in measuring linear dimensions with high precision. The interferometer is mounted on the x-axis of the machine and is used to determine the displacement of the mechanical probe along the x-axis. The wavelength of the laser light is calibrated using the national standard, ensuring that the measurements obtained from the interferometer are accurate and traceable.

Theoretical note 5 *An interferometer is a device that uses the interference of light waves to measure small displacements, distances, or changes in refractive index. Its operation is based on the principle of superposition, where two waves combine to form a resultant wave whose intensity is determined by the phase difference between the two. In the case of a laser interferometer, a laser beam is split into two paths: a reference path and a measurement path. The reference path serves as a stable reference for comparison, while the measurement path changes its length based on the displacement being measured (x-axis in the Moore machine). When the two beams are recombined, interference occurs, tha intensity of the light is detected using a photodiode and the phase difference is calculated to determine the displacement which is represented by fractions of the wavelength of the laser light.*

As illustrated in Figure 2.16, two mirrors function as reference mirrors in a double-pass plane-mirror interferometer. These mirrors are securely attached to a z-stage, which allows the mechanical probe to move up and down. The supporting structure is specifically designed to hold the mirrors symmetrically and in line with the mechanical probe, which helps to counteract any thermal drifts along the x-axis that could affect measurements. This symmetric arrangement ensures that both the measuring and reference beam paths align correctly, meeting the Abbe condition for lateral and vertical alignment when probing the middle height section of the gauge.

The rectangular shape of the two mirrors in the reference path permits vertical movements of approximately ±45 mm and lateral movements of about ±17 mm. This design allows the interferometer to accurately measure x-displacements while maintaining a stable signal. Continuous monitoring of the x-positions of the contact probe is essential throughout the entire measurement process. This includes moving the probe from one end face to another, across the outer surfaces of external diameter gauges, or laterally to locate the maximum diameter of cylindrical gauges.

Additionally, significant vertical movements are necessary for calibrating the gauge at various heights and for other measurement tasks, such as checking straightness. The current setup is capable of accommodating diameter gauges with a lateral size of up to 200 mm, a height of about 100 mm, and linear artifacts measuring

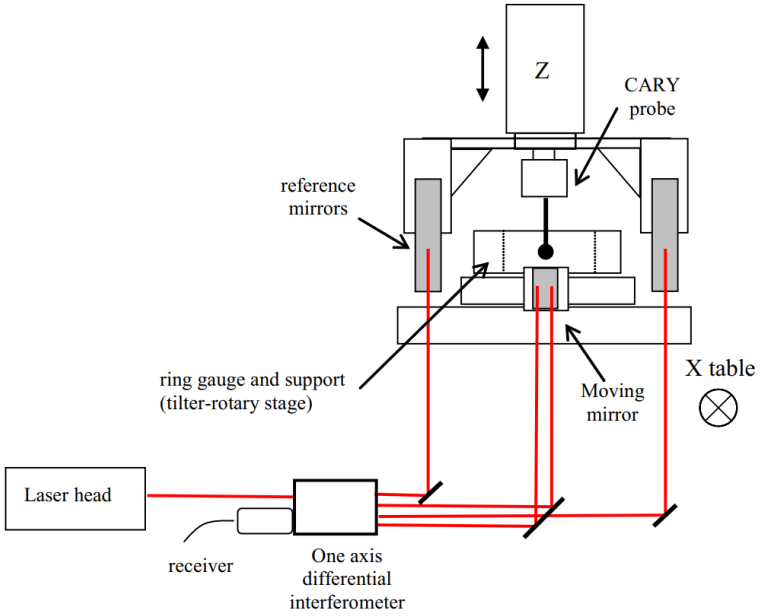


Figure 2.16: Moore machine interferometer setup. In red the optical path of the laser beam which is parallel to the x-axis movement. The machine view is from the side with the z-axis movement in the vertical direction and the x-axis movement entering the page.

up to 420 mm in length [10]. This flexibility makes the system suitable for a wide range of measurement applications.

The reading of the interferometer is performed by a dedicated device produced by KeySight [27], which is connected to the computer via a USB interface. The manufacturer provides the necessary dynamic libraries to interface the device with the computer and the software. In the following code snippet, we show how to read the interferometer using the KeySight dynamic libraries in C++.

```
void Keysight::connect()
{
    int ok = Initialize_E1735A_DLL();
    if (ok == 0) {
        E1735A_SelectDevice(0);
    } else {
        std::cerr << "Failed to init" << std::endl;
    }
}

double Keysight::readInstr()
{
    double sample = E1735A_ReadSample() / 2;
    return sample;
}
```

Figure 2.17: E1735A laser reader.

```
16
17 double Keysight::readSamplesAvg(int n_samples_to_read)
18 {
19     TLaserSample samples[n_samples_to_read];
20
21     E1735A_SetupTimer(0.0001);
22     E1735A_SetSampleTriggers(TB_SOFTWARE | TB_TIMER);
23     E1735A_StartTimer();
24
25     std::this_thread::sleep_for(std::chrono::milliseconds(1));
26
27     E1735A_StopTimer();
28
29     int num_sample = E1735A_ReadAllSamples(samples,
    n_samples_to_read);
30     double sum = 0.0;
31     double readAverageSamples = 0.0;
32     double stan_deviation = 0.0;
33
34     calculateAvgAndStanDeviation(samples, num_sample, sum,
    readAverageSamples, stan_deviation);
35     if(stan_deviation > 0.001)
36     {
37         return this->readSamplesAvg(n_samples_to_read);
38     }
39
40     return readAverageSamples;
41 }
```

The code snippet demonstrates the initialization of the device, the configuration of the measurement parameters, and the acquisition of the interferometer reading. There are two methods that can be used to read the value of the x position, the first

method reads only the last available sample to perform a fast acquisition, this is used for movement control. The last method implemented allows to read the value averaging over a specified number of samples and repeat the acquisition until the value of the standard deviation is below a certain threshold, this is used to perform the actual measurement. In both functions the value of the x position is returned as a double and it must be divided by two since the double interferometric setup doubles the light travel distance. All the functions starting with `E1735A` are part of the dynamic library provided by KeySight, the following are the ones used in the keysight class:

- `Initialize_E1735A_DLL()`: This function is called once at the beginning to initialize the E1735A DLL. This is necessary for using all the other functions within the DLL. It returns an integer which is 0 upon success.
- `E1735A_SelectDevice(0)`: This function is used to select the active E1735A device. The argument 0 indicates the first connected device. This function is needed to be called before accessing any other function of the DLL.
- `E1735A_ReadSample()`: This function reads the current laser position. It returns a double value representing the measured position. The return value depends on the interferometer type and its parameters. By default, it returns linear displacement in millimeters. In your code, the returned value is divided by two to account for the double pass of light in the interferometer. If no sample can be read, NAN is returned and an error code is saved which can be accessed with `E1735A_ReadLastError()`.
- `E1735A_SetupTimer(0.0001)`: This function sets the interval of the internal timer, which is used to trigger measurements. The argument 0.0001 sets the interval to 0.0001 seconds or 100 microseconds.
- `E1735A_SetSampleTriggers(TB_SOFTWARE | TB_TIMER)`: This function enables or disables the triggers for sample collection. In this case, it enables both software and timer triggers using the constants `TB_SOFTWARE` and `TB_TIMER`. Other trigger types include `TB_REMOTE`, `TB_AQB` and `TB_EXTERNAL`.
- `E1735A_StartTimer()`: This function starts the internal timer, initiating the sampling process based on the set interval. It is necessary to call `E1735A_SetupTimer()` first.
- `E1735A_StopTimer()`: This function stops the internal timer. This function should be called after measurements are completed.
- `E1735A_ReadAllSamples(samples, n_samples_to_read)`: This function reads all available samples from the buffers and stores them into a user-defined buffer. It returns the number of samples read.

- `E1735A_SetParameter(int Index, double Value)`: This function is used to set parameters of the interferometer and environment. The Index parameter specifies which parameter to change, such as air temperature `OP_AIRTEMP`, air pressure `OP_AIRPRES`, laser wavelength `OP_WAVELENGTH`, relative humidity `OP_RELHUMI`, or dead path `OP_DEADPATH`. The Value parameter is the value to which the corresponding parameter should be set.

- `E1735A_ReadSampleCount()`: This function returns the number of samples currently available in both the hardware and software buffers. This function also moves the samples from the hardware buffer to the software buffer.

In the C++ implementation a class for managing the old laser reader was also implemented for compatibility reasons, the class is called `Laser` and it uses the GPIB protocol to communicate with the device. A single command `"XPOS?"` can be sent to the instrument to read the x position. To use the GPIB protocol a PCI card is needed, the card is connected to the computer via a PCI slot and to the instrument via a GPIB cable. The library from National Instruments `ni4882.h` is used to interface the card with the computer.

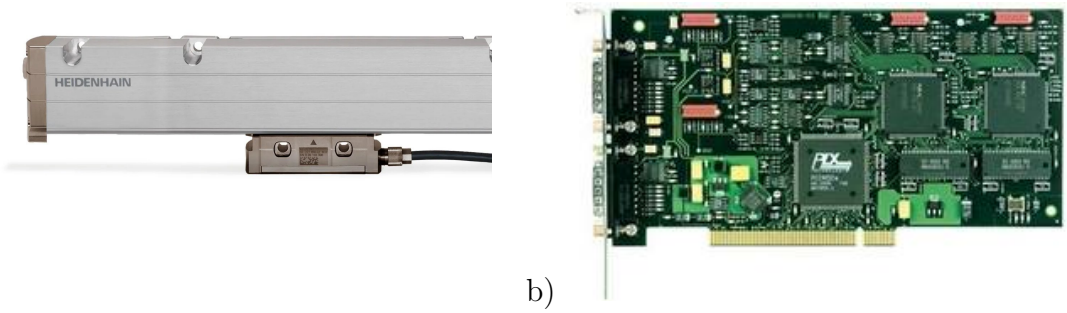
2.5.2 Optical scales (linear encoders)

To precisely determine the position of the mechanical probe along the z-axis, and the crosslide along the y-axis, the Moore machine is equipped with optical scales. These scales are used to measure the displacement of the probe and the crosslide with high accuracy, providing feedback to the control system for position control and error correction.

Theoretical note 6 *Optical scales are linear encoders that use light to determine the position of a moving object. The scale consists of a glass substrate with a series of fine lines etched onto its surface. A light source and a photodiode are positioned on opposite sides of the scale, with the light passing through the lines and being detected by the photodetector. As the object moves, the light intensity changes due to the lines blocking or allowing the light to pass through. By measuring these intensity changes, the position of the object can be accurately determined.*

To read the values from the two devices a PCI board is used, the board is connected to the computer via a PCI slot and to the scales via two cables. The board is capable of reading the sine and cosine signals from the scales and to convert them into a position value. A dynamic library provided by the manufacturer is used to interface the board with the computer and the software.

The IK220 [28] provides a set of functions to interface with Heidenhain optical encoders, the following is a summary of the functions used in the implemented class:



a) b)

Figure 2.18: Optical scales setup: a) Optical scale, b) IK220 PCI board.

- `IK220WritePar`: This function is used to write a parameter to the IK220 card for a specified axis. The function takes the axis number `(USHORT Axis)`, the parameter number `(USHORT ParNum)`, and the parameter value `(ULONG ParVal)` as input. In your code, this is used to set the encoder type and signal type. For example, the code uses `IK220WritePar(Ax, 1, ENC_INCREMENTAL)` to set the encoder type to incremental and `IK220WritePar(Ax, 2, SIG_11UA)` to set the signal type to 11uA.

- `IK220ReadPar`: This function reads the value of a parameter from the IK220 card for a given axis. The function takes the axis number `(USHORT Axis)`, the parameter number `(USHORT ParNum)`, and a pointer to a variable where the parameter value will be stored `(ULONG* pParVal)`.

- `IK220Init`: This function loads the firmware into the IK220 card and starts it. It is called for each axis before other functions can be used. It takes the axis number `(USHORT Axis)` as input.

- `IK220Set`: This function sets the position value of a specified axis. It takes the axis number `(USHORT Axis)` and the new position value `(double SetVal)` as input.

- `IK220Read48`: This function reads the 48-bit counter value of a specified axis. The function takes the axis number `(USHORT Axis)`, the latch register `(USHORT Latch)`, and a pointer to a variable where the counter value will be stored `(double* pData)`. The counter value represents the position of the axis.

- `IK220Get48`: This function also reads the 48-bit counter value of a specified axis. It is similar to `IK220Read48` but requires that the counter value be saved in a register before reading it.

- `IK220Latch`: This function saves the counter value in the specified register. It takes the axis number `(USHORT Axis)` and the register number `(USHORT`

`Latch)` as input. This is used to save the current position before reading it with `IK220Get48`.

These functions provide the basic functionality to configure, read, and control the Heidenhain optical encoders through the IK220 interface card. The parameters for the encoders, like the type and signal, are set using `IK220WritePar`. The position of the axes can be set using IK220Set and read using `IK220Read48` or `IK220Get48` after saving the value with `IK220Latch`.

The IK220 card can support encoders with sinusoidal current signals (11 µAPP), voltage signals (1 VPP), EnDat, or SSI interfaces. It is designed for applications that require high resolution of encoder signals and fast measured value acquisition. The card also has a 12-bit interpolation value combined with a 32-bit period counter, which creates a 44-bit measured value. The measured values are stored in a 48-bit data register.

The y and z axes are configured with different scaling values by setting different signal periods, which are specific to each axis, and by setting the encoder and signal types using the `IK220WritePar` function. The code snippet below shows how the parameters are set for each axis using the `setParams` function, and the `IK220WritePar` function is used within this function to set the encoder type and signal type:

```
void Scale::setParams(ULONG encoderType, ULONG signalType, double
    signalPeriod)
{
    if (!IK220WritePar(ax, 1, encoderType)) std::cout << "Error in
    writePar";
    else std::cout << "Setting Encoder Type to incremental" << std
    ::endl;
    if (!IK220WritePar(ax, 2, signalType)) std::cout << "Error in
    writePar";
    else std::cout << "Setting Signal Type to 11uA" << std::endl;

    scaleParams.encoderType = encoderType;
    scaleParams.signalType = signalType;
    scaleParams.signalPeriod = signalPeriod;
}

void Moore::init()
{
  ...
    // INIT THE 3 AXIS
    Xaxis.init((PosInstr*)&key, ser, x_lab);
    Yaxis.init((PosInstr*)&yScale, ser, y_lab);
    Zaxis.init((PosInstr*)&zScale, ser, z_lab);

  ...

    yScale.setParams(ENC_INCREMENTAL, SIG_11UA, M_SIG_PERIOD_Y);
```

```
24      zScale.setParams(ENC_INCREMENTAL, SIG_11UA, M_SIG_PERIOD_Z);
25    ...
26  }
```

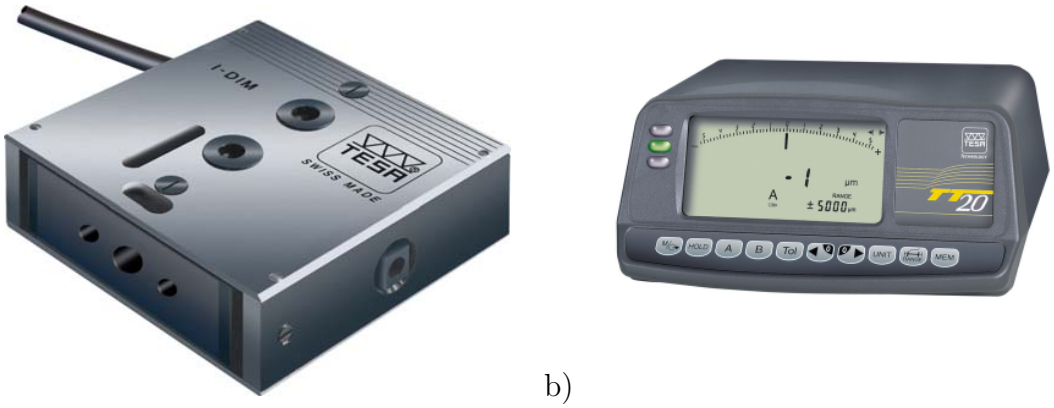
- The `setParams` function takes three arguments:
  - `encoderType`: This sets the type of encoder being used, which is set to `ENC_INCREMENTAL`.
  - `signalType`: This sets the type of signal the encoder is using, which is set to `SIG_11UA`.
  - `signalPeriod`: This parameter, which is not used in the code snippets provided but is available in the `Scale` class, allows to set a scaling factor related to the signal period of the encoder.
- The code uses the `IK220WritePar` function to set the encoder type and signal type parameters for the specified axis.
- The `scaleParams` struct stores the values of encoder type, signal type, and signal period.
  - This struct can then be used to retrieve the parameters set using the `getParams()` method.
- The `Moore::init` function initializes the axes and sets the parameters for the y and z axes.
- The `yScale.setParams` line sets the encoder type to `ENC_INCREMENTAL`, the signal type to `SIG_11UA` and the signal period to `M_SIG_PERIOD_Y` for the y-axis.
- The `zScale.setParams` line sets the encoder type to `ENC_INCREMENTAL`, the signal type to `SIG_11UA` and the signal period to `M_SIG_PERIOD_Z` for the z-axis.
- `M_SIG_PERIOD_Y` and `M_SIG_PERIOD_Z` are macros that hold the specific scaling factors for the y and z axis respectively.

In summary, the y and z axes are configured with different scaling values by passing different `signalPeriod` values to the `setParams` function and setting up the axes with the IK220 parameters to `ENC_INCREMENTAL` and `SIG_11UA`, which are specific to each axis. Although the `signalPeriod` is not used in the code, it is available to be set in the `Scale::setParams` and can be used to set different scaling values for each axis.

2.5.3 Cary LVDT sensor

To make contact with the sample and measure its dimensions, the Moore machine is equipped with a Cary Linear Variable Differential Transformer (LVDT) sensor. The LVDT sensor is mounted on the z-axis of the machine and is used to detect the displacement of the mechanical probe along the x-axis. Figure 2.19 shows the setup of the head of the LVDT sensor and the reader of the sensor.

Theoretical note 7 *A (LVDT) is a type of electromechanical sensor used to convert mechanical motion or vibration into an electrical signal. The LVDT consists of a primary coil and two secondary coils wound on a hollow cylindrical core. A movable ferromagnetic core is placed inside the coil assembly, which moves in response to the mechanical motion being measured. When an alternating current is applied to the primary coil, it induces a voltage in the secondary coils that varies with the position of the core. By measuring the difference in voltage between the two secondary coils, the LVDT can accurately determine the displacement of the core and, by extension, the mechanical motion being measured.*



a) b)

Figure 2.19: Moore machine LVDT sensor setup: a) Cary LVDT sensor head, b) Tesa LVDT sensor reader.

The interface between the PC and the reader is done via a special optical USB cable. The connection is not simple since the cable is not a standard USB cable, some control flow signals (DTR and RTS) of the RS232 protocol are used to give the internal diode of the cable the correct power supply. The parameters of the communication are reported in Table 2.4.

The instrument is controlled by sending commands to the reader, the commands are sent as strings and the reader responds with a string containing the requested information. Also in this case two different read functions are implemented, the

Table 2.4: LVDT sensor communication parameters.

Parameter	Value
Baud Rate	4800 Baud/s
Byte Size	7 bits
Stop Bits	2 bits
Parity	Even
DTR Control	Enabled
RTS Control	Disabled
Port	COM1

first one reads only the last available sample to perform a fast acquisition while the second one reads the value averaging over a specified number of samples as explained for the interferometer in section 2.5.1.

The possible commands that can be sent to the reader are:

- `?`: This command is used to read the current measurement value from the instrument. It retrieves the latest data available from the sensor.
- `RNG?`: This command reads the current range setting of the instrument. It returns the range that is currently configured on the device.
- `RNG n`: This command sets the desired range on the instrument. The parameter n' specifies the range value, where n' can be a specific range value or 0' for auto range.

All the commands must be terminated with a carriage return character.

2.5.4 Nilox webcam

In order to automate the sample recognition process, the Moore machine is equipped with a Nilox webcam (Figure 2.20). The webcam is mounted on the z-axis of the machine and is used to capture real time images of the measurement setup. The images are processed by the software to identify the sample and determine its position. The webcam is connected to the computer via a USB interface and is controlled using the OpenCV library as explained in section 2.7.2.

2.6 Environmental monitoring

The calculation of the refractive index of air is essential in many scientific and engineering applications, particularly in optics and metrology. The refractive index of air affects the propagation of light and, consequently, the wavelength of the light itself. Since the speed of light varies depending on the medium through which it

Figure 2.20: Nilox webcam

travels, it is necessary to correct the wavelength of light based on the refractive index of air to obtain accurate measurements.

Edlen's formula is one of the most commonly used expressions to calculate the refractive index of air as a function of pressure, temperature, and humidity [29]. It is therefore necessary to accurately measure the environmental conditions while performing the measurements with the machine, this is done throug various instrumentation explained in the following sections.

The coefficient of thermal expansion (CTE) is a critical parameter that quantifies how much a material expands or contracts in response to changes in temperature. This property is particularly significant when dealing with materials such as ceramics, glass, and metals, which are commonly encountered in various artifacts and industrial applications. Accurate temperature measurements are essential when measuring artifacts, even slight temperature variations can lead to significant changes in dimensions, potentially affecting the fit and function of components in assemblies.

Theoretical note 8 *The Edlén equation is a mathematical formula used to calculate the refractive index of air as a function of various atmospheric conditions, such as temperature, pressure, and humidity. This is crucial for applications such as laser ranging, astronomical observations, and the calibration of optical devices, where even small discrepancies in the refractive index can lead to significant errors in measurements.*

Figure 2.21: Thermometer FLUKE Super-DAQ with multiple channel expansion.

2.6.1 Thermometer

The model used for temperature measurement is the FLUKE super-DAQ (Figure 2.21), which is actually a multimeter equipped with six PT-100 sensors connected to six different channels. All six probes are periodically calibrated, and the calibration curve is set directly within the instrument. The control of the thermometer from the PC is performed via a LAN connection, necessitating the creation of a C++ class to open and manage the communication.

Theoretical note 9 *A Local Area Network (LAN) connection allows devices to communicate with each other over a limited geographical area, such as a home, office, or campus. In the context of communicating with instruments like the FLUKE super-DAQ, a LAN connection enables the instrument to be controlled and monitored remotely via a networked computer. This is typically achieved using the Transmission Control Protocol (TCP), which provides reliable, ordered, and error-checked delivery of data between applications running on devices connected to the network. To establish a TCP connection, a socket must be created. A socket is an endpoint for sending and receiving data across a network.*

Command Explanations

The following commands are sent to the FLUKE super-DAQ instrument to configure it for temperature measurement:

- ‘*RST’: This command is used to reset the instrument to its default state. It clears any previous settings and configurations, ensuring that the device starts from a known baseline. It helps in avoiding unexpected behavior and ensures that the device is ready for new commands.
- ‘*RCL 04’: This command is used to recall a specific setup or configuration stored in the instrument’s memory. The number ‘04‘ indicates the specific memory location from which the settings will be retrieved. By using the ‘*RCL’ command, the user can quickly restore a predefined configuration that may include calibration settings, measurement parameters, or other relevant settings.
- ‘DATA:LOG:DEST MEM’: This command sets the destination for logged data to the internal memory of the instrument. It tells the FLUKE super-DAQ to store the measurement data in its onboard memory rather than sending it directly to an external device or display. This is useful for batch processing or when the data needs to be retrieved later for analysis. It allows for more flexible data management, especially in scenarios where continuous monitoring is required.
- ‘DATA:LOG:AUTO OFF’: This command disables automatic logging of data. When set to "OFF," the instrument will not automatically log data at predefined intervals or conditions. Disabling automatic logging allows the user to have more control over when data is recorded. This is useful in situations where the user wants to manually trigger data logging based on specific events or conditions.
- ‘TRIG:COUN 1’: This command sets the trigger count to 1, meaning that the instrument will take a single measurement when triggered.
- ‘ROUT:SCAN (@201:206)’: This command configures the instrument to scan specific channels, in this case, channels 201 to 206. The ‘@’ symbol indicates that the channels specified in parentheses are to be included in the scan.
- ‘STAT:OPER?’: This command queries the operational status of the instrument. It returns a status code that indicates the current state of the device, such as whether it is ready to take measurements or if there are any errors. The ‘STAT:OPER?’ command is crucial for ensuring that the instrument is in a proper operational state before proceeding with measurements. By checking the operational status, the user can avoid attempting to read data when the instrument is not ready or it is still measuring, thus preventing errors.
- ‘FETC?’: This command is used to fetch the most recent measurement data from the instrument. It retrieves the temperature readings from the active channels and returns them as a comma-separated string. By sending this

command, the user can access the latest temperature readings, which can then be parsed and processed for further analysis.

TCP Client creation

Below is reported the class methods implemented to setup a client server connection. The code first checks if the socket has already been created. If not, it creates a new socket using the `socket` function with the parameters `AF_INET` (indicating IPv4) and `SOCK_STREAM` (indicating a TCP socket).

If the provided address is not a valid IP address (checked using `inet_addr`), it attempts to resolve the hostname using `gethostbyname`. If the hostname is successfully resolved, the code retrieves the IP address from the `h_addr_list`. If the address is a plain IP address, it is directly assigned to `server.sin_addr.s_addr`.

The code sets the address family (`AF_INET`) and port number in the `server` structure and finally, it attempts to connect to the remote server using the `connect` function.

```
bool tcp_client::conn(std::string address, int port)
{
    ...
    sock = socket(AF_INET, SOCK_STREAM, 0);
    ...

    if (inet_addr(address.c_str()) == -1)
    {
        struct hostent* he;
        struct in_addr** addr_list;

        //resolve the hostname, its not an ip address
        if ((he = gethostbyname(address.c_str())) == NULL)
        ...
    }
    // plain ip address
    else server.sin_addr.s_addr = inet_addr(address.c_str());
    ...
    //Connect to remote server
    if (connect(sock, (struct sockaddr*)&server, sizeof(server)) <
    0)
    ...
    std::cout << "Connected\n";
    return true;
}


bool tcp_client::send_data(std::string data)
{
    //Send some data
    if (send(sock, data.c_str(), strlen(data.c_str()), 0) < 0)
```

```
31      {
32          perror("Send failed : ");
33          return false;
34      }
35
36      return true;
37  }
38
39
40  std::string tcp_client::receive(int size)
41  {
42      char buffer[size];
43
44      //Receive a reply from the server
45      std::string reply;
46
47      int nch = recv(sock, buffer, sizeof(buffer), 0);
48      ...
49      reply = buffer;
50      return reply;
51  }
```

2.6.2 Voltmeter

To measure the pressure and humidity, we use a voltmeter with a GPIB (General Purpose Interface Bus) interface that reads the voltage signals coming from the barometer and igrometer. The voltmeter is an Agilent 3458A model (Figure 2.22), which is a 8.5 digit multimeter.

Theoretical note 10 *The General Purpose Interface Bus (GPIB), also known as IEEE 488, is a standard interface used for connecting and controlling multiple electronic instruments, such as oscilloscopes, multimeters, and signal generators, in a laboratory or industrial setting. GPIB allows for communication between a computer and various instruments in parallel, enabling automated data acquisition, control, and analysis.*

A C++ class derived from the `ambInstr` class has been created to manage the GPIB communication with the voltmeter which is done using a custom command set provided by the instrument manufacturer.

The following commands are used to control the voltmeter:

- `BEEP ONCE`: This command is sent in the `setParams` function. It instructs the voltmeter to emit a single beep sound, possibly for user feedback upon initialization.

Figure 2.22: Agilent 3458A voltmeter.

- `ID?`: This command is sent in the `setParams` function. It is a system query that requests the instrument to return its identification string, allowing the user to verify the model or version of the voltmeter.

- `NRDGS 10`: This command is used in `setParams`. It sets the number of readings to be taken per trigger to 10, meaning that whenever a trigger occurs, the voltmeter will take 10 measurements.

- `DCV AUTO`: This command appears in `setParams`. It likely refers to DC Voltage measurement, with `AUTO` indicating that the measurement range should be set to automatic, allowing the voltmeter to select the appropriate range based on the input signal.

```
void Voltmeter::setParams()
{
  ask("BEEP ONCE",  false);
  ask("ID?",       true);

  ask("NRDGS 10",   false);  // do 10 readings @ trig
  ask("DCV AUTO",   false);  // 10 V range DC
}
```

- `MATH STAT`: This command is used in both `readPressure` and `readHumidity` functions. It enables statistical functions on the measured data, allowing the user to perform calculations such as mean, standard deviation, etc., on the collected measurements.

- `TERM REAR`: This command is used in `readPressure` and `readHumidity`. It selects the rear input terminals for measurement, indicating where the probes should be connected for accurate readings.

- `CHAN 5` (in `readPressure`) and `CHAN 0` (in `readHumidity`): These commands specify the channel to be measured. The voltmeter is capable of measuring multiple channels, and channels 5 and 0 are designated for pressure and humidity measurements, respectively.

- **`TRIG SGL`**: This command is used in both `readPressure` and `readHumidity`. It is related to triggering the measurement, with `SGL` indicating that a single trigger will initiate one set of readings, as configured by `NRDGS`.
- **`RMATH MEAN`**: This command is used to retrieve the measured value in both `readPressure` and `readHumidity`. It queries the voltmeter to return the mean value of the measurements taken after the trigger, providing a summary statistic of the collected data.

Barometer

Figure 2.23: Rosemount barometer.

The Rosemount Pressure Transducer RMT 1201 F1B1A1A converts a pressure reading into a voltage signal, which is then read by the voltmeter. The transducer is designed to measure air pressure in the range of 0.8 to 1.1 bar and it is equipped with a 0-5V output signal, which is proportional to the pressure reading.

Tha calibration of the transducer is performed in INRiM, and the calibration curve is set directly in the program using two predefined macros `PRESSURE_A` and `PRESSURE_B` that are used to calculate the pressure from the measured voltage.

```
double Voltmeter::readPressure()
{
  ask("MATH STAT", false);  // enable statistics
  ask("TERM REAR", false);
  ask("CHAN 5", false);
  ask("TRIG SGL", false);  // Trigger the multimeter
  std::this_thread::sleep_for(std::chrono::seconds(5));
  double val = ask("RMATH MEAN", true);   // Ask the mean value
```

```
9
10    val = val * PRESSURE_A + PRESSURE_B;
11    return val;
12  }
```

Dewpoint igrometer

Figure 2.24: Michell Instruments S3000 igrometer monitor.

The Michell Instruments S3000 igrometer, specifically designed for measuring humidity, is an instrument for dew point measurement, its Automatic Balance Compensation (ABC) system periodically heats the measurement mirror to eliminate condensation and counteract contamination, ensuring that the surface remains dry for accurate readings. Additionally, the device is equipped with a 'CLEAN OPTICS' warning light that activates when the optics have accumulated significant contamination, prompting users to perform maintenance. A TTL signal output is also available for remote alarm notifications.

The operating range of the Series 3000 is influenced by the ambient temperature of the sensor and the efficiency of heat sinking, with a maximum achievable dew point equal to the ambient temperature or 80°C, whichever is lower, and a minimum dew point of -65°C at an ambient temperature of -40°C.

The instrument calibration is performed in INRiM and results in a linear calibration curve, it is possible to set the calibration curve directly in the program using two predefined macros `HUMIDITY_A` and `HUMIDITY_B` that are used to calculate the dew point temperature from the measured voltage.

```
1  double Voltmeter::readHumidity()
2  {
3    ask("MATH STAT", false);  // enable statistics
4    ask("TERM REAR", false);
5    ask("CHAN 0", false);
6    ask("TRIG SGL", false);  // Trigger the multimeter
7    std::this_thread::sleep_for(std::chrono::seconds(5));
```

```
8   double val = ask("RMATH MEAN", true);   // Ask the mean value
9
10  val = val * HUMIDITY_A + HUMIDITY_B;
11  return val;
12 }
```

2.7 Moore controller

In order to control all the components fo the Moore machine an industrial computer was chosen. The computer specs are reported in Table 2.5. The components marked in bold where the ones necessary for the project and the reason why this computer was chosen, some legacy devices present in the current machine configuration can only be interfaced with COM ports or PCI cards with specific characteristics.

Table 2.5: Industrial PC Specifications.

Component	**Specification**
Chassis	AX60810WM Desktop Chassis for Mainboard ATX
Power Supply	PS400W_ATX Power Supply 400W ATX
Processor	LGA1151 socket 9th_8th Gen Intel Core Processor
Motherboard	Intel C246 PCH DDR4 (ECC)
Connectivity	USB 3.1(Gen2) **COM PORT** PCIex16 2 PCIex4 **4 PCI**
Model	IMB525RVDHGGA_C246
COM Port Cable	2 x COM port cable with bracket, P=2.54 mm, L=300 mm
CPU	Intel Core i5_8500 Coffee Lake 9 MB Cache, 3GHz
CPU Cooler	FAN/CPU CPU Cooler
RAM	DDR4 _ 2133MHz DIMM 288pin (16GB)
Storage	Solid State Disk 2.5" SATA 500GB
TPM Module	AX93515 TPM 2.0 module for IMB/MMB

The PC runs the Moore controller software [15], which is responsible for managing the operation of the machine. The software is designed to provide a user-friendly interface for controlling the machine's movements, setting parameters, and monitoring the measurement process. The industrial PC offers the necessary processing power and connectivity to ensure smooth and reliable operation of the Moore machine. The software was written in C++ leveraging the power of object-oriented programming to create a modular and extensible system. The software architecture is based on the Moore class, which encapsulates all the functionality required to control the machine. The moore controller program structure is as follows:

- **Core System Classes:**
  - **`Moore:`** Manages the overall operation of the measuring machine.

    - Includes instances of `Asse` for each axis (X, Y, and Z).
    - Manages communication with optical scales using an `IkOptical` object.
    - Handles communication with the Keysight laser interferometer using a `Keysight` object.
    - Manages communication with the LVDT sensor using a `Cary` object.
    - Uses a `SimpleSerial` object to communicate with the microcontroller.
  - **`Asse`:** Represents a single axis of the machine.
    - Uses a `PosInstr` interface to read position data from the instrument the axis is associated to to close the feedback movement loop.
    - Uses a `SimpleSerial` object to communicate with the microcontroller.
  - **`PosInstr`:** Abstract base class for position measuring instruments.
    - Implemented by `Keysight`, `Scale`, `Laser`, `Cary`, and `CHRocodile`.
  - **`IkOptical`:** Manages communication with the IK220 PCI board for optical scales.
  - **`Scale`:** Configures and manages the optical scales.
  - **`SimpleSerial`:** Utility for handling serial communication.

- **Environmental Parameter Classes:**
  - **`Paramb`:** Manages the environmental parameter measurement.
  - **`Fluke`:** Manages communication with the FLUKE thermometer via LAN.
  - **`Voltmeter`:** Abstract class for pressure and humidity sensors.

- **GUI Related Classes:**
  - **`MyApp`:** Main application class.
  - **`PosFrame`:** Base class for the position control GUI frame.
  - **`AxisFrame`:** GUI frame for setting the parameters of each axis.
  - **`CHRMeasFrame`:** GUI frame for CHR measurements.
  - **`AmbFrame`:** GUI frame for environmental parameter monitoring.

- **Other Classes:**
  - **`Nilox`:** Manages the Nilox camera for machine vision and marker detection.
  - **`pos`:** Represents a 3D position and defines the most common operators to handle point operations.

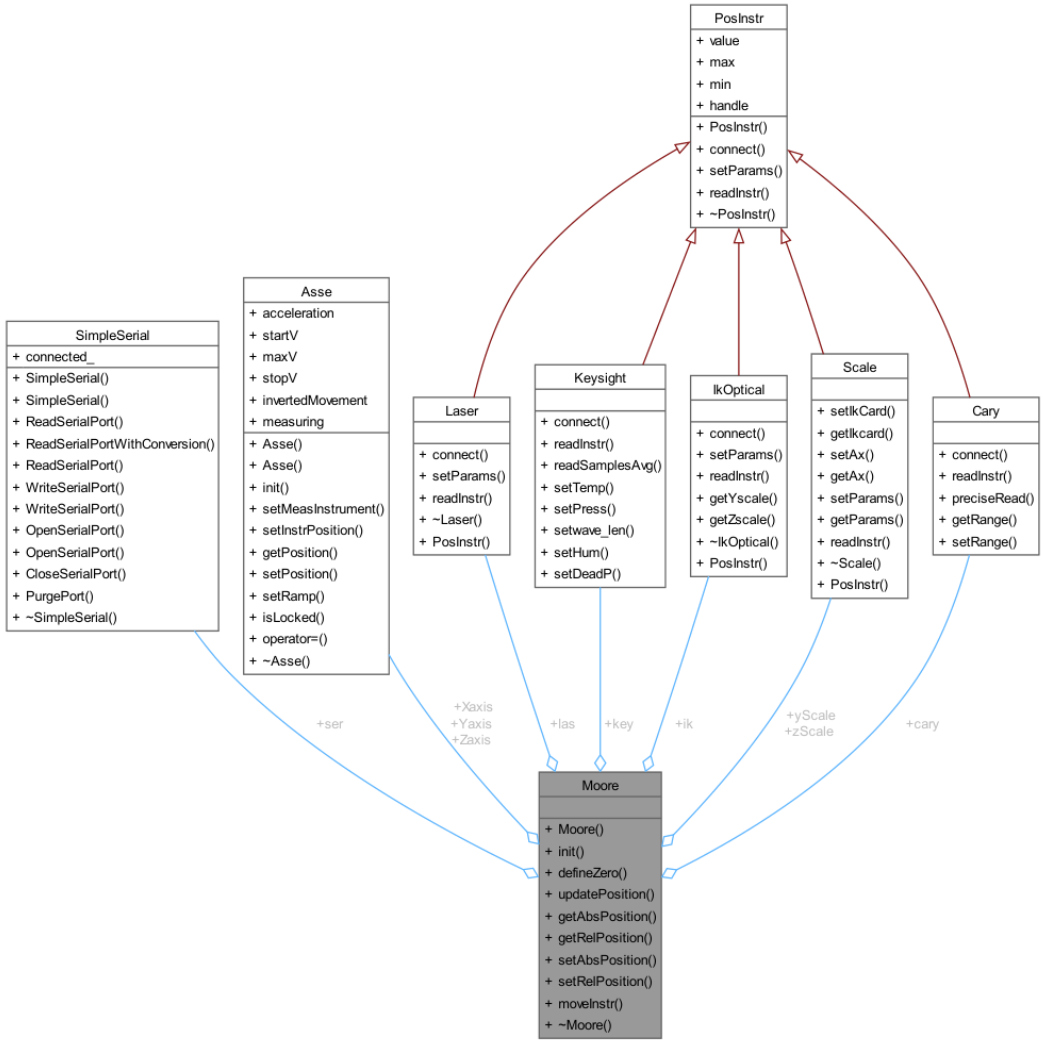


Figure 2.25: Moore core system classes and dependency / inheritance structure.

2.7.1 Acceleration ramps

To guarantee the correct movement of the motors it is necessary that the axis speed follows a linear trend, both in acceleration and in deceleration, to do this the PC must calculate the instantaneous speed and send it to the microcontroller. This is done by the `Asse::setVelocity` function, which calculates the instantaneous speed based on the distance traveled by the axis and the parameters set by the user. In Equation 2.8 the calculation of the istantaneous speed is reported where a is the axis acceleration value, t is the traveled distance and d is the total distance the axis needs to be moved to reach the destination.

$$v = \begin{cases} a \cdot t, & \text{if } t < \frac{d}{2} \\ a \cdot (d - t), & \text{if } t > \frac{d}{2} \end{cases} \tag{2.8}$$

After calculating the speed, the function limits the result to the maximum and minimum values to be sent to the microcontroller, in this way the speed curve shown in Figure 2.26 is obtained.

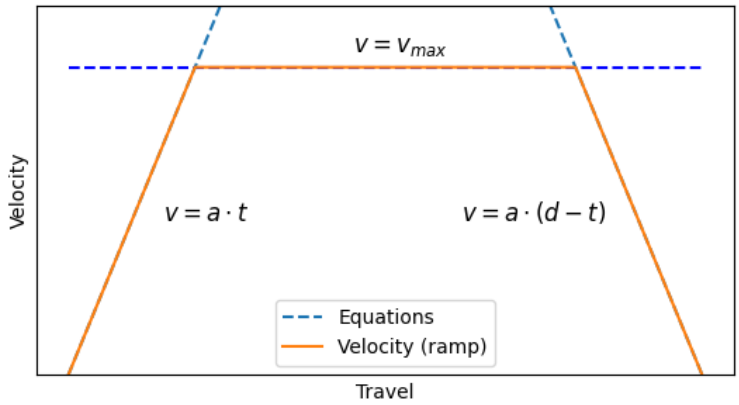


Figure 2.26: Acceleration ramps for motor control.

It is important to clarify that the acceleration ramps are calculated based on the position and not based on time since only in this way it is really possible to guarantee a correct positioning after closing the loop.

To set the acceleration parameters function `Asse::setRamp` can be used, this function also sets the starting speed and end speed of the motion. The start speed cannot be too low since the infinite screw has some slack that needs to be covered in a short amount of time in order to have fast movement.

```
double distance = abs(targetPosition - startP);
double travel = 0;

while (distance - travel > 0)
{
    setVelocity(distance, travel);
    P = position;
    if (instr == 'm')
        P = measPT->readInstr();
    travel = abs(P - startP);

    std::this_thread::sleep_for(std::chrono::milliseconds(10));
};

velocity = 0;
```

```
16 lock = true;
17 sendVelocityToMicro();
```

2.7.2 Coordinate system definition

In a coordinate measurement system (CMS), the ability to define multiple reference systems is crucial for achieving accurate and flexible measurements. Different workpieces may require distinct coordinate systems to accommodate their unique geometries and orientations. The use of rotation and translation matrices enables the transformation of coordinates from one reference system to another, facilitating the alignment of measurements with the machine's fixed coordinate system. This capability is essential for ensuring that measurements are consistent and comparable, regardless of the orientation or position of the workpiece.

Theoretical note 11 *Rotation and translation matrices are essential tools in linear algebra used for transforming geometric objects in space, particularly in fields like computer graphics, robotics, and computer vision. A rotation matrix is a square matrix that rotates points around a specified axis. In two dimensions, it can be represented as:*

$$R(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$$

In three dimensions, rotation matrices can rotate points around the X, Y, or Z axes, preserving the length of vectors due to their orthogonal properties.

A translation matrix, on the other hand, is used to move points in space without changing their orientation. In two dimensions, it is expressed in a homogeneous coordinate system as:

$$T = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

where tx and ty are the translation distances along the X and Y axes.

The measurement machine operates at a relatively slow pace, and the manual definition of workpiece coordinate systems can be a time-consuming process. This manual method requires the physical contact of a probe with the objects, which not only prolongs the setup time but also introduces the potential for human error and inconsistencies in the measurements. To address these challenges and enhance the efficiency of the measurement process, a webcam has been installed on the machine, equipped with the capability to detect markers using the OpenCV library.

Theoretical note 12 *OpenCV, or Open Source Computer Vision Library, is a powerful and widely used open-source software library designed for computer vision and image processing tasks. The library provides a comprehensive set of tools and*

functions that enable users to perform a wide range of operations, including image manipulation, object detection, facial recognition, motion tracking, and machine learning. Its extensive documentation and active community further contribute to its popularity, making it a go-to resource for both beginners and experienced practitioners in the field of computer vision.

The integration of the webcam allows for a significant reduction in the time required to establish the coordinate systems for the workpieces. By strategically placing markers at key points on the objects, the system can determine their positions without the need for manual probing. This approach leverages computer vision technology to automate the detection and localization of the markers, speeding up the entire measurement process.

The webcam is programmed to map the 2D space of its field of view (FOV) with the machine's coordinate system. It captures images of the work area and processes them using OpenCV. The software identifies the markers within the captured images and calculates their positions relative to the machine's coordinate system, which is derived from readings obtained from the laser and optical encoders. This allows for precise localization of the workpieces without the need to physically move the machine or make contact with the objects.

The use of a webcam in the measurement system introduces several challenges that must be addressed to ensure accurate and reliable performance. One significant issue is the lens distortion inherent in many webcams, particularly those with a fisheye design. This distortion can lead to inaccuracies in the captured images, as straight lines may appear curved and the proportions of objects can be misrepresented. To mitigate this effect, calibration techniques must be employed to correct the distortion and ensure that the measurements derived from the images accurately reflect the true geometry of the workpieces.

Another challenge arises from the coordinate system of the webcam's field of view (FOV), which does not directly correspond to the actual dimensions of the samples being measured. The mapping between the camera's coordinate system and the physical dimensions of the workpieces must be robust enough to maintain accuracy even if the webcam is repositioned. This requires the implementation of a reliable calibration process that can adapt to changes in the camera's orientation or distance from the objects. The system must be designed to consistently translate the coordinates detected by the webcam into meaningful measurements that align with the machine's fixed coordinate system, ensuring that the data remains valid regardless of any adjustments made to the camera's position.

It is important to note that the marker detection for the localization of the workpiece is intended to be only an approximation of the precise location, the real position must be calculated by the machine automatically by making contact with the sample under analysis starting from the approximate position detected.

Fisheye and distorsion correction

OpenCV provides a comprehensive framework for working with fisheye cameras, which are characterized by their wide field of view and significant lens distortion. The key components involved in modeling fisheye cameras, including the camera matrix, distortion matrix, and the process of undistorting images. The camera matrix, often denoted as K, is a 3×3 matrix that contains the intrinsic parameters of the camera. It is defined as follows:

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

where: - f_x and f_y are the focal lengths in pixels along the x and y axes, respectively. - c_x and c_y are the coordinates of the principal point, typically located at the center of the image.

Fisheye lenses introduce significant distortion to the captured images, which can be modeled using a distortion matrix. In OpenCV, the distortion coefficients for fisheye cameras are typically represented as a vector D:

$$D = \begin{bmatrix} k_1 & k_2 & k_3 & k_4 & k_5 \end{bmatrix}^T$$

where: - k_1, k_2, k_3, k_4, k_5 are the distortion coefficients that describe the radial and tangential distortion effects.

The radial distortion can be modeled as:

$$r_d = r \cdot (1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

where r is the distance from the center of the image, and r_d is the distorted radius.

The chessboard calibration method [30, 31] is a widely used technique in computer vision for calibrating cameras. This method involves capturing multiple images of a chessboard pattern from different angles and distances. By analyzing these images, OpenCV can compute the camera matrix and distortion coefficients. The chessboard pattern consists of a grid of alternating black and white squares. The corners of these squares serve as calibration points. The number of inner corners in the chessboard pattern is specified, typically denoted as (n_x, n_y), where n_x is the number of squares along the width and n_y is the number of squares along the height. The total number of inner corners is $n_c = n_x \times n_y$.

The calibration process involves several key steps:

- Image Acquisition: Capture multiple images of the chessboard pattern from different perspectives. It is important to cover a range of angles and distances to ensure robust calibration.

- Corner Detection: For each captured image, OpenCV detects the corners of the chessboard pattern using the function `findChessboardCorners`. This function returns the pixel coordinates of the detected corners.

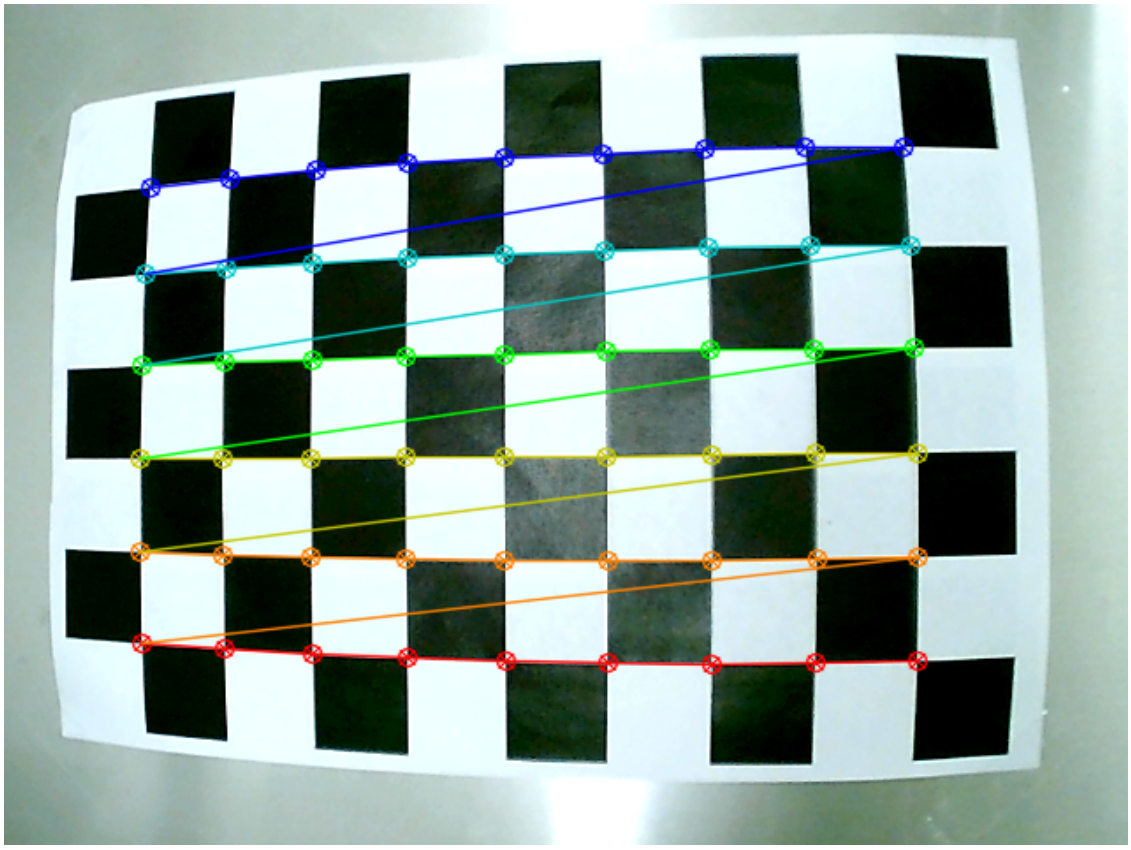


Figure 2.27: Chessboard pattern for camera calibration.

- Object Points Generation: The 3D coordinates of the chessboard corners in the real world are defined in a known coordinate system.

  To correct for the distortion introduced by the fisheye lens, OpenCV provides functions to undistort images. The undistortion process involves mapping the distorted image coordinates back to the ideal pinhole camera model coordinates.

- Camera Matrix and Distortion Coefficients Calculation: Using the detected corner points and the corresponding object points, OpenCV applies the camera calibration function `calibrateCamera`. This function computes the camera matrix K and the distortion coefficients D by minimizing the reprojection error, which is the difference between the observed image points and the projected object points.

After calculating the matrices, the undistortion can be performed as described in [32].

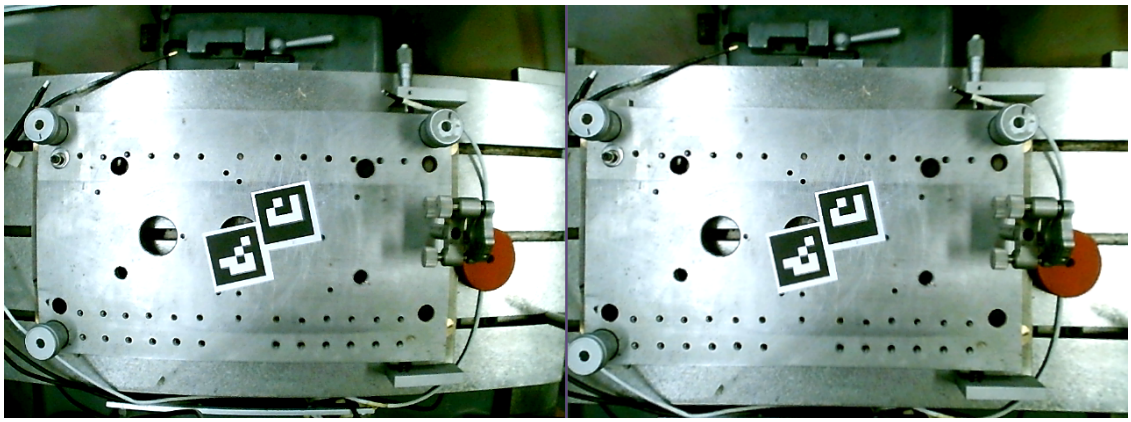


Figure 2.28: Image undistortion process, on the left the original image from the Nilox camera, on the right the undistorted image.

In code this can be done in the following way:

```
void Nilox::calibrate()
{
    // Define the dimensions of the chessboard
    cv::Size boardSize(9, 6); // Number of inner corners per chessboard row and column
    float squareSize = 25.0f; // Size of a square in your defined unit (e.g., meters)

    // Prepare object points (3D points in real world space)
    std::vector<cv::Point3f> objectPoints;
    for (int i = 0; i < boardSize.height; i++)
    {
        for (int j = 0; j < boardSize.width; j++)
        {
            objectPoints.emplace_back(j * squareSize, i * squareSize, 0);
        }
    }
    ...
    std::cout << "Press 'a' to capture an image for calibration." << std::endl;
    std::cout << "Press 's' to start calibration and save parameters." << std::endl;

    while (key != 'q')
    { // Press 'q' to quit
        ...
        // Find the chessboard corners
```

```
24          bool found = findChessboardCorners(gray, boardSize, corners
    );
25          if (key == 'a')
26          { // If a key is pressed
27              objectPointsList.push_back(objectPoints);
28              imagePoints.push_back(corners);
29              std::cout << "[INFO] Captured image for calibration."
    << std::endl;
30          }
31          if (key == 's')
32          {
33              // Calibrate the camera
34              std::vector<cv::Mat> rvecs, tvecs;
35              calibrateCamera(objectPointsList, imagePoints, gray.
    size(), cameraMatrix, distCoeffs, rvecs, tvecs);
36              break;
37          }
38      }
39  }
```

By understanding the camera matrix, distortion matrix, and the mathematical principles behind undistorting images, we can effectively correct for lens distortion and improve the accuracy of their computer vision applications.

Camera FOV to machine coordinate mapping

ArUco markers [33] are a type of fiducial marker used in computer vision applications for robust and accurate pose estimation. OpenCV provides a dedicated module for detecting and tracking these markers, which can be utilized in various applications, including augmented reality, robotics, and camera calibration.

The ArUco marker detection process involves several key steps:

- Marker Generation: ArUco markers are square-shaped patterns that contain a unique binary code. Each marker is defined by a specific size and can be generated using OpenCV's functions. The markers are typically printed on a flat surface.

- Image Acquisition: A camera captures images of the scene containing the ArUco markers. The quality of the captured images is crucial for accurate detection.

- Detection: OpenCV provides the function `detectMarkers` to identify ArUco markers in the captured images. This function returns the corners of the detected markers and their corresponding IDs. The detection process involves thresholding the image, finding contours, and identifying the square shapes that correspond to the markers.

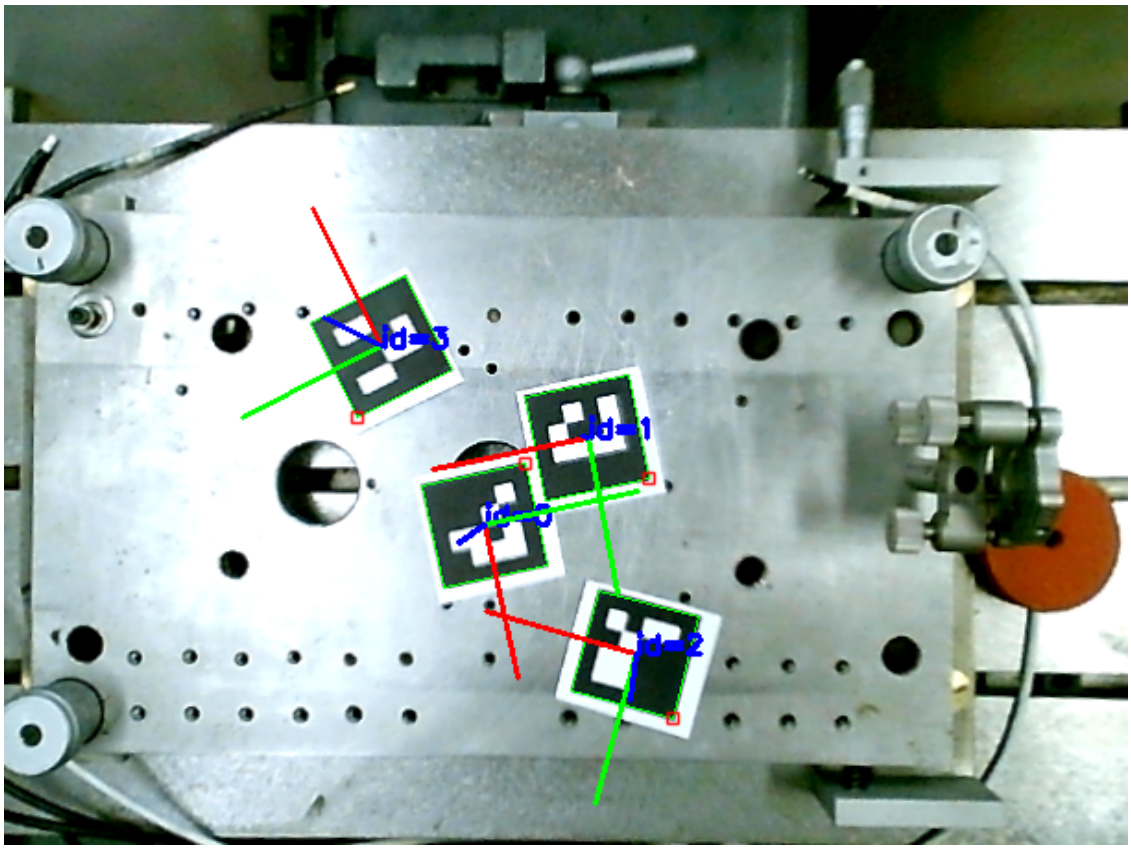


Figure 2.29: AruCo marker detection.

- Position Calculation: Once the markers are detected, the next step is to calculate their positions in the FOV. By knowing the size of the marker in millimeters, the pixel coordinates of the corners can be mapped to real-world 2D coordinates (X, Y). The relationship between the marker size and its pixel representation allows for accurate distance measurements, even if the camera moves. The position of each marker can be calculated using the following equations:

$$X = \frac{(x_{marker} - c_x) \cdot \text{size}}{f_x}, \quad Y = \frac{(y_{marker} - c_y) \cdot \text{size}}{f_y}$$

  where (x_{marker}, y_{marker}) are the pixel coordinates of the marker's center, size is the physical size of the marker in millimeters, and (c_x, c_y) are the coordinates of the principal point i n the camera matrix.

- Displacement Calculation: After calculating the positions of all detected markers, the displacements required to center the probe on each marker can be

determined. This involves comparing the current position of the probe with the calculated positions of the markers.

In code this is done in the following way:

```
float Nilox::detect(std::vector<cv::Point2f> &markerPos, std::
    vector<int> &ids, float markerSize, bool estimatePose)
{
     ...
     while (inputVideo.grab())
     {
          cv::Mat image, undistorted, imageCopy;
          inputVideo.retrieve(image);
          cv::undistort(image, undistorted, this->cameraMatrix,
    this->distCoeffs);
          ...
          // detect markers and estimate pose
          detector.detectMarkers(undistorted, corners, ids,
    rejected);
          ...

          char key = (char)cv::waitKey(waitTime);
          if (key == 'q')
          {
               if(!ids.empty()) {
                    float sum = 0;
                    int n = 0;
                    for (auto corner: corners) {
                         n += 4;
                         sum += cv::norm(corner[0] - corner[1]);
                         sum += cv::norm(corner[1] - corner[2]);
                         sum += cv::norm(corner[2] - corner[3]);
                         sum += cv::norm(corner[3] - corner[0]);
                    }
                    float l = sum / n;
                    float scale = markerSize / l;

                    for (auto center: centers) {
                         markerPos.emplace_back(center - cv::Point2f
    (image.size()) / 2);
                    }
                    return scale;
               }
          }
     }
     return 0;
}
```

Once the displacements are calculated, the machine can be programmed to move its three axes accordingly. By adjusting the positions of the X, Y, and Z axes based on the calculated displacements, the probe can be accurately centered on

the detected markers. This process allows for precise positioning and measurement without the need for manual adjustments, enhancing the efficiency and accuracy of the overall system.

2.7.3 Multithreading

Multithreading is a programming technique that allows multiple threads to exist within a single process, enabling concurrent execution of tasks. Each thread represents a separate path of execution, allowing a program to perform multiple operations simultaneously. Additionally, multithreading can improve resource utilization by allowing a program to perform background operations, such as file I/O or network communication, without blocking the main execution flow. In the specific case of the Moore controller, this technique comes particularly handy in a variety of applications:

- concurrent movement of multiple axes
- loop reading of instrumentation
- other temporary background processes (keyboard control etc.)

In the program, there are two threads running in an infinite loop, each serving a distinct purpose to ensure the efficient operation of the system. The first thread is responsible for continuously reading the environmental parameter instruments, such as temperature, humidity, and pressure. This thread performs an iteration every five minutes, gathering and updating the relevant environmental data for laser correction. This periodic update is sufficient for monitoring changes in the environment, as these parameters typically do not fluctuate rapidly.

The second thread, on the other hand, is dedicated to monitoring the instantaneous position of the three axes X, Y, and Z of the machine. This thread operates at a much higher frequency, executing an iteration every 10 milliseconds. This rapid polling is crucial for ensuring that the system has up-to-date information about the position of the machine at all times. The instantaneous position data is essential for precise motion control, as it allows the system to make real-time adjustments to the motor commands based on the current state of the machine.

```
1 std::thread ambThread{ &MooreLayAmbFrame::UpdateAmb, this }; //
    create ambParam infinite
2 void MooreLayAmbFrame::UpdateAmb()
3 {
4   ambVals v;
5   while (true)
6   {
7     v = this->amb->scanParams();
8     std::this_thread::sleep_for(std::chrono::seconds(300));
9   }
```

```
}

std::thread posThread{ &MoorePosFrame::UpdatePosition, this }; // create pos infinite

void MoorePosFrame::UpdatePosition()
{
  while (true)
  {
    this->moore->updatePosition();
    std::this_thread::sleep_for(std::chrono::milliseconds(10));

    pos abs = this->moore->getAbsPosition();
    pos rel = this->moore->getRelPosition();
  }
}
```

When a movement command is issued to move the machine to a target point, (p_{target}), three separate and simultaneous threads are created to control the movement of each axis concurrently. Each thread is dedicated to managing the motion of one specific axis, allowing for coordinated and efficient movement. This multi-threaded approach enables the system to execute complex motion commands while maintaining responsiveness. The movement loop relies heavily on the instantaneous position data provided by the second thread to generate the acceleration ramps for the motors. Since the position data is updated every 10 milliseconds, the system can dynamically adjust the motor commands based on the latest position information, facilitating smooth and precise movements.

```
void Moore::setAbsPosition(pos target)
{
  std::thread xt(&Asse::setPosition, &Xaxis, target.x); // create single axis movement
  std::thread yt(&Asse::setPosition, &Yaxis, target.y); // create single axis movement
  std::thread zt(&Asse::setPosition, &Zaxis, target.z); // create single axis movement

  xt.join();
  yt.join();
  zt.join();
}
```

2.7.4 Documentation and Doxigen

Documenting the code is essential for maintaining clarity and facilitating collaboration among developers. Well-documented code helps others understand the purpose and functionality of various components, making it easier to modify, debug, and extend the codebase. One effective tool for generating documentation is Doxygen

[34], which automates the process of creating comprehensive documentation from annotated source code.

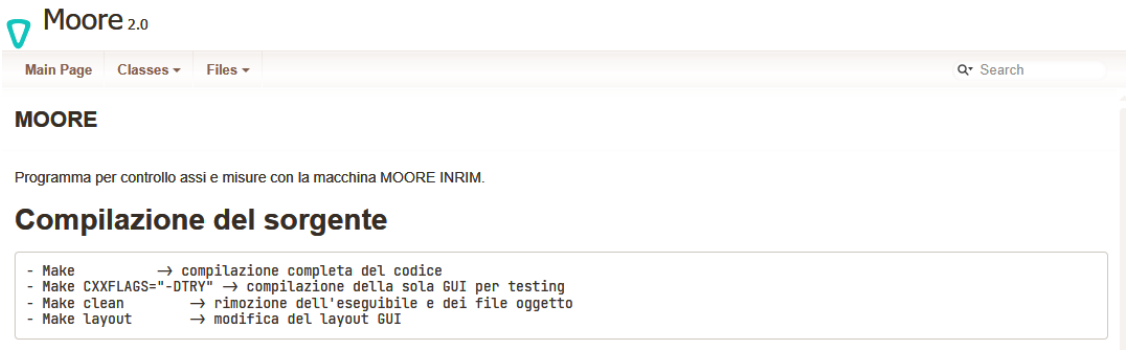


Figure 2.30: Section of index.html page generated by Doxygen.

Doxygen works by parsing specially formatted comment strings embedded within the code. These comments can include descriptions of classes, functions, parameters, return values, and more. By following a consistent format, developers can provide detailed explanations that Doxygen will convert into structured documentation. The output can be generated in multiple formats, including HTML for web-based documentation and LaTeX [35] for high-quality printed documents.

Using Doxygen in conjunction with comment strings not only enhances the readability of the documentation but also ensures that it remains up-to-date with the code. As the code evolves, the documentation can be regenerated easily, reflecting any changes made.

In the moore project each header file includes an explanation for each function declaration, the comments are defined in the following format:

- @brief: explanation of the functionality of the implemented method
- @param: function input parameter explanation and type
- @return: the return type and explanation of the value
- @throws: the exception that the user should handle when calling the function

An example of generated documentation is reported in Figure 2.31, in the upper part there is the comment stub with all the doxigen keyword, while in the bottom part there is the generated html document on the browser, the Moore documentation main page is reported in Figure 2.30. Also all class diagrams (Figures 2.25, 3.4) present in this thesis were generated automatically based on the code itself using Doxigen.

```
/** @brief Finds the real roots of a cubic equation.

The function solveCubic finds the real roots of a cubic equation:
-   if coeffs is a 4-element vector:
\f[\texttt{coeffs} [0] x^3 +  \texttt{coeffs} [1] x^2 +  \texttt{
    coeffs} [2] x +  \texttt{coeffs} [3] = 0\f]
-   if coeffs is a 3-element vector:
\f[x^3 +  \texttt{coeffs} [0] x^2 +  \texttt{coeffs} [1] x +  \
    texttt{coeffs} [2] = 0\f]

The roots are stored in the roots array.
@param coeffs equation coefficients, an array of 3 or 4 elements.
@param roots output array of real roots that has 1 or 3 elements.
@return number of real roots. It can be 0, 1 or 2.
*/
CV_EXPORTS_W int solveCubic(InputArray coeffs, OutputArray roots);
```

◆ solveCubic()

int cv::solveCubic (**InputArray** coeffs,
OutputArray roots
)

Python:
cv.solveCubic(coeffs[, roots]) -> retval, roots

`#include <opencv2/core.hpp>`

Finds the real roots of a cubic equation.

The function solveCubic finds the real roots of a cubic equation:

- if coeffs is a 4-element vector:

$$\texttt{coeffs}[0]x^3 + \texttt{coeffs}[1]x^2 + \texttt{coeffs}[2]x + \texttt{coeffs}[3] = 0$$

- if coeffs is a 3-element vector:

$$x^3 + \texttt{coeffs}[0]x^2 + \texttt{coeffs}[1]x + \texttt{coeffs}[2] = 0$$

The roots are stored in the roots array.

Parameters

coeffs equation coefficients, an array of 3 or 4 elements.

roots output array of real roots that has 1 or 3 elements.

Returns

number of real roots. It can be 0, 1 or 2.

Figure 2.31: An example of code documentation generated by Doxygen.

2.8 Graphical user interface

A graphical user interface (GUI) is a type of user interface that allows users to interact with electronic devices through graphical icons and visual indicators, instead of text-based interfaces, typed command labels or text navigation. GUIs

were introduced in reaction to the perceived steep learning curve of command-line interfaces (CLIs), which are more flexible and powerful in many ways, but are also more difficult for new or occasional users to learn. The actions in a GUI are usually performed through direct manipulation of the graphical elements. The elements of a GUI are often organized as a hierarchy of graphical control elements, including windows, dialog boxes, buttons, text fields, labels, checkboxes, radio buttons, and other widgets.

When designing the Moore application two main options were considered for the GUI: WxWidgets and Clay. The following sections will provide an overview of these two libraries and the reasons why WxWidgets was chosen for the project.

2.8.1 Clay and Raylib

Clay [36] is a powerful stb style library for C and C++ developed by NicBarker that provides a simple and easy-to-use API for creating graphical applications.

Theoretical note 13 *Stb libraries [37] are a collection of single-file public domain libraries for C/C++ that are designed to be easy to integrate into an existing codebase. They are lightweight, fast, and easy to use, making them ideal for small projects or prototyping. The single file is divided in two parts: the implementation and the header. This allows the user to include the header file in their code and choose if compile a separate dynamic library or include the implementation in their code by defining a preprocessor macro.*

Clay has zero dependencies, including no C standard library, which makes it highly portable and easy to integrate into various projects. It allows to layout an intuitive UI and then render it using Raylib [38], WebGL Canvas, or even as HTML (wasm) [39]. This flexibility in output makes it easy to composite the UI in custom engines or environments.

Theoretical note 14 *Wasm or web assembly is a binary instruction format for a stack-based virtual machine hosted on a web browser. It is designed as a portable target for compilation of high-level languages like C/C++/Rust, enabling web applications to run at near-native speed on the web.*

Clay uses a flexible and readable declarative syntax with nested UI element hierarchies. Developers can mix elements with standard C code, such as loops, conditionals, and functions, providing a powerful and versatile way to create complex UIs. Despite its simplicity, Clay is fast enough to recompute the entire UI every frame (immediate UI [40]), ensuring smooth and responsive interfaces. Additionally, Clay includes built-in "Chrome Inspector"-style debug tooling, allowing developers to view their layout hierarchy and configuration in real-time. This feature greatly aids in debugging and optimizing the UI.

To have such a fast library with a simple API and no dependencies has a significant drawback, Clay does not support differentiated callbacks or event handling. There are two sides to this issue. On one hand, the refresh rate of the UI is so high that it is not necessary to have callbacks to update the UI, on the other end the handling of all the user actions must be done manually in a polling fashion. This can be a significant drawback for complex applications that require a lot of user interaction. Furthermore Clay is a recent open source project and it is not as mature as other libraries like WxWidgets.

Theoretical note 15 *A callback is a function that is passed as an argument to another function, allowing the second function to call the first function at a later time. In GUI programming, callbacks are often used to handle events such as button clicks, mouse movements, and keyboard input. When an event occurs, the GUI framework calls the appropriate callback function to handle the event. This allows developers to create interactive applications that respond to user input in real time.*

On a positive note Clay is a very promising library, the real time re-computation of each frame does not allow inconsistencies in the GUI. This is a common issue in other libraries where the UI is not updated in real time and the user can see the changes in the UI happening in a non-sequential order.

To test the library, a simple application was created to test the UI and the interaction with the user. GUI widgets are graphical user interface elements that allow users to interact with software applications. They serve as building blocks for user interfaces and include components such as buttons, text boxes, sliders, checkboxes, and menus. Each widget has a specific function, enabling users to perform actions, input data, or navigate through the application. CLAY however does not provide complex widgets (e.g sliders) and so the software also required the custom implementation of a subset of widgets:

- Slider
- Button
- Radio Button
- Input Box

The program aimed to also test the motor control system and the serial communication with the microcontroller.

Slider

The slider is a rectangular graphic which can be scrolled from either left to right and vice versa using the mouse. See Figure 2.32. It is used to control the speed

Figure 2.32: Slider GUI element.

of the motor. By scrolling or clicking on any position of the slider with the mouse the motor will begin to spin at the speed indicated in the widget body. The slider is a structure that contains a callback function, a minimum, maximum, real and a boolean value.

```
typedef struct
{
    int min;
    int max;
    int value;
    int real;

    bool active;
    void (*onHoverFunction)(Clay_ElementId elementId,
        Clay_PointerData pointerData,
        intptr_t userData);
} slider;
```

When the slider is clicked and scrolled within the slider container, it also functions when the mouse is outside the frame. As long as the mouse remains pressed it will continue to function and only when it is released will it finally stop. This is done using the on hover function; "speedSliderOnHover" and "MainFrameHover". The speedSliderOnHover function is responsible for ensuring the slider functions when the mouse is pressed while inside the slider container. While the MainFrameHover is responsible for ensuring the slider functions even if the mouse is pressed outside the slider container.

```
void speedSliderOnHover(Clay_ElementId elementId, Clay_PointerData
    pointerData, intptr_t userData)
{
    // Pointer state allows you to detect mouse down / hold /
    release
    if (pointerData.state == CLAY_POINTER_DATA_PRESSED_THIS_FRAME)
    {
        printf("Slider activated\n");
        speed_slider.active = true;
    }
}

void MainFrameHover(Clay_ElementId elementId, Clay_PointerData
    pointerData, intptr_t userData)
{
    if (pointerData.state == CLAY_POINTER_DATA_PRESSED) // handle
    pressed mous
```

```
    {
        if (speed_slider.active == true)
        {
            speed_slider.value = pointerData.position.x - (3 *
    PADDING);
            if (speed_slider.value < MIN_SLIDER_W)
                speed_slider.value = MIN_SLIDER_W;
            if (speed_slider.value > MAX_SLIDER_W)
                speed_slider.value = MAX_SLIDER_W;

            int converted_value = map(speed_slider.value,
    MIN_SLIDER_W, MAX_SLIDER_W,
                speed_slider.min, speed_slider.max);

            Serial_SendCommand(fd,
                               radioOptions.press[0] ? 'f' : 'b',
                               radioOptions.press[1] ? 'w' : 'm',
                               radioOptions.press[2] ? 's' : 'r',
                               converted_value);
        }
    }
}
```

Input Box

The input box is a small text window where the user can interact with the GUI by entering text from the keyboard (See Figure 2.33). In particular, in this case the text must be a number ranging from 1 to 300, if the text input is not within that range or contains any characters, an error message will be displayed. The acquired digits, will be converted to an integer and used as a step value. This step value will be used to control the rate at which the motor and slider will go from zero to maximum speed and from maximum back to zero when the test ramp button is pressed. Below is the C code for the input box.

```
typedef struct{

    int n;
    bool textactive;
    Clay_String textContent;

    void (*onHoverFunction)(Clay_ElementId elementId,
                            Clay_PointerData pointerData,
                            intptr_t userData);

}inputbox;
```

For the programming of the input box the Raylib library was used in the main section of the code. To allow the user to type text to the GUI appending characters to the Clay-String is required. However, CLAY does not allow directly appending

Figure 2.33: Input Box GUI element.

characters to the "Clay-String" as it is a `const char *`. Therefore, a new char variable must be created and used to store the input text. Then a new Clay-String is created each time a character is appended to the created char variable.

```
if (text_box.textactive == true)
{
    int key = GetCharPressed();

    // NOTE: Only allow keys in range [32..125]
    if ((key >= 32) && (key <= 125))
    {
        char new_content[100];
        sprintf(new_content, "%s%c", text_box.textContent.chars, (
    char)key);

        text_box.textContent = (Clay_String){
            .chars = new_content,
            .length = text_box.textContent.length + 1};
    }
}
```

Buttons

These are static buttons and when they are clicked they carry out a specific task. For this particular GUI there are two buttons:

- the Zero button: This is used to make the slider go from the value it is currently positioned, down to zero. It also makes the motor go from the current speed it is spinning, down to zero.
- the Ramp button: This is used to make the slider and motor go from zero speed to maximum speed and back to zero.

Radio Buttons

The motor's graphical user interface (GUI) features six radio buttons that change color and activate upon being clicked. The first three buttons are labeled [**w-off**],

[**fore**], and [**fine-angle**], while the remaining three correspond to the x, y, and z axes.

- The [**fore**] button controls the motor's rotation direction. When pressed, it changes to [**back**] reversing the motor's spin from clockwise to anti-clockwise.
- The [**w-off**] button indicates whether the winding is engaged. Pressing it changes the label to [**w-on**], activating the winding mechanism for the motor.
- The [**fine-angle**] button adjusts the motor's stepping mode. When clicked, it changes to [**default-angle**] reverting to the standard step size.
- The [**X**], [**Y**], and [**Z**] buttons are used to select which of the x, y, and z axis of the machine is being controlled by the slider.

When it comes to programming the radio buttons the challenging part was how to make the program identify which button was pressed. In CLAY, multiple buttons are seen as a single radio button entity and so it is necessary to distinguish each one when they are pressed. Fortunately, CLAY has a function called "Clay-HashString" which takes a key, an offset, and a seed. And so using this function, a for loop can be utilized to label and identify which of the radio buttons was pressed. See below the code the shows the use of "Clay-HashString".

```
1
2  radio radioOptions = {
3      .name = CLAY_STRING("fw"),
4      .labels = {CLAY_STRING("fore"), CLAY_STRING("w_off"),
       CLAY_STRING("fine_angle")},
5      .labels_change = {CLAY_STRING("back"), CLAY_STRING("w_on"),
       CLAY_STRING("default_angle")},
6      .active = 0,
7      .n = 3,
8      .press = {false,false,false},
9      .onHoverFunction = radioOptionsOnHover,
10 };
11
12 void radioOptionsOnHover(Clay_ElementId elementId, Clay_PointerData
       pointerData, intptr_t userData)
13 {
14     if (pointerData.state == CLAY_POINTER_DATA_PRESSED_THIS_FRAME) {
15         for (int i = 0; i < radioOptions.n; ++i) {
16             if (Clay__HashString(CLAY_STRING("fw"), i, 0).id ==
       elementId.id) {
17                 radioOptions.active = i;
18                 radioOptions.press[i] =! radioOptions.press[i];
19             }
20         }
21     }
22 }
```

Complete GUI

The complete GUI consists of several components arranged as shown in figure 2.34. At the top, there is an orange header displaying the label "Sending" along with the current speed indicated by the slider. Below the header is the green slider which allows users to adjust the current speed. Directly beneath the slider are three radio buttons labeled [**w-off**], [**fore**], and [**fine-angle**], accompanied by a ramp button. Further down, there are additional radio buttons for the x, y, and z axes.

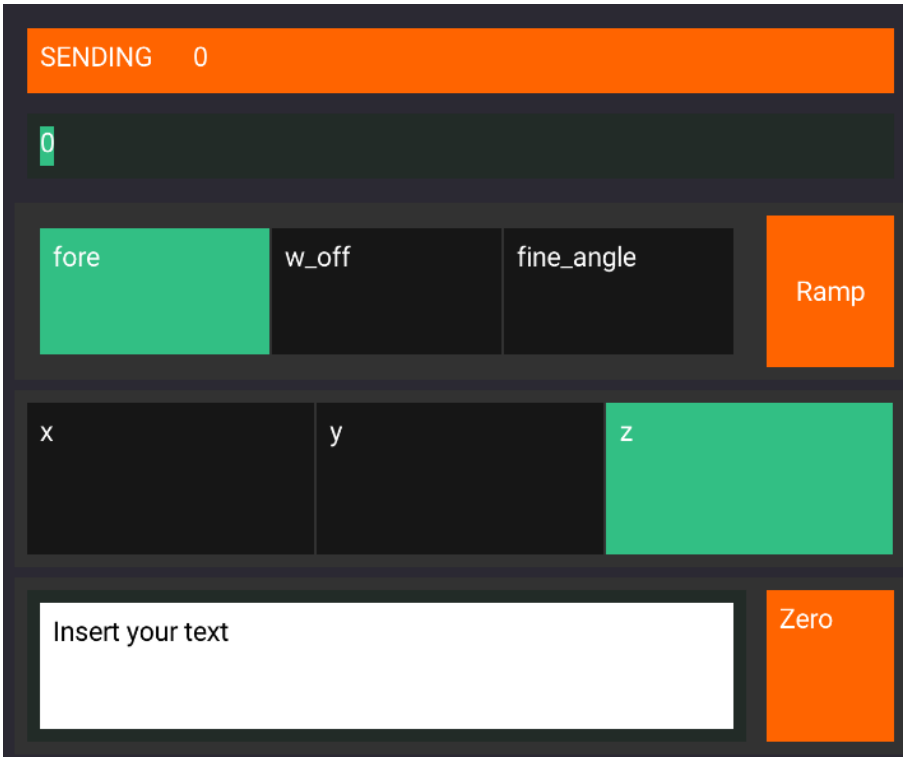
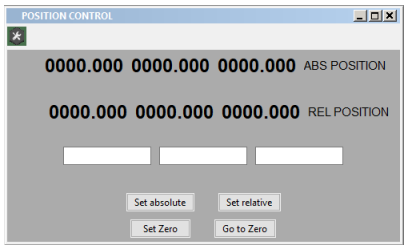


Figure 2.34: Full GUI for motor testing.

2.8.2 WxWidgets

The Moore project includes a GUI to allow the user to interact with the software in a more user-friendly way. The GUI is implemented using the WxWidgets library [41], which is a C++ library that lets developers create applications for Windows, macOS, and Linux. WxWidgets is a mature library that has been around for a long time and is widely used in the industry. It provides a wide range of widgets and tools to create GUI applications, and it is well-documented and supported by a large community of developers. The main frames of the application are the following:

- **Position control frame**: This frame allows the user to control the position of the three axes, the speed of the axes. The user can also set the position of the axes to a specific value, set the origin of the movement.

a) 

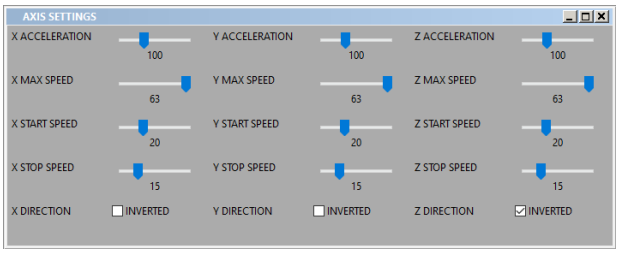
b) 

Figure 2.35: Position control frames (a) position indication and setting, (b) speed settings for each axis.

- **Environment parameters frame**: This frame allows the user to monitor the temperature and humidity of the environment. The window includes four temperature readings and two for humidity and pressure.

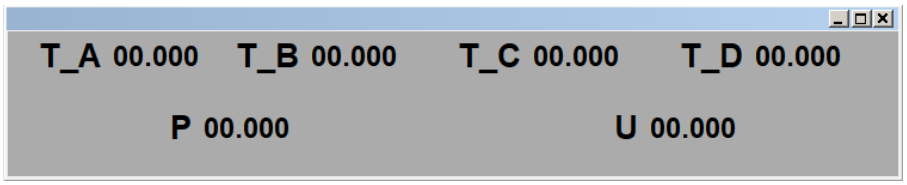


Figure 2.36: Environment parameters frame.

- **Measurement fame**: This frame allows the user to set the parameters of the measurement, different measurement types are available, for each measurement a different UI can be configured. Figure 2.37 shows the UI for the measurement of profiles with a chromatic confocal sensor.

The various frames and layouts can be created graphically using a program provided by WxWidgets called wxFormBuilder [42]. This program allows the user to create the layout of the GUI by dragging and dropping widgets onto a canvas. The program generates the code for the GUI, which can then be used in the application. This makes it easy to create complex GUIs without having to write a lot of code. The callbacks can be configured directly in the program, making it easy to link the widgets to the application logic. The generated code consists of a header file and a source file that define the main aspect of the GUI, furthermore for each frame an abstract class is defined that can be inherited to create the specific frame.

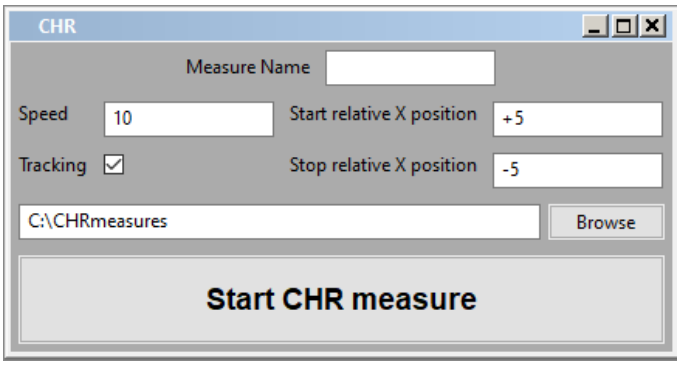


Figure 2.37: Measurement frame.

Chapter 3

Movement routines and testing

Given the designed control system and the machine's capabilities, the next step requires the implementation of the logic behind the machine movements for specific purposes. The machine must measure diameter of spheres, cylinders, optical scales, and the control system must be able to perform the necessary movements to achieve all these measurements. Some routines are general and can be used for any type of measurement, while others are specific to the type of measurement being performed.

3.1 Approach

The approach routine is used to move the probe in contact with the sample and is designed to move the probe in a controlled manner to ensure that it makes contact with the sample without causing damage or excessive force, which could damage the sample or the probe itself. The routine takes as input the target position (P3 in Figure 3.1), the speed of approach and the direction.

Starting from an arbitrary position P0, at first the probe is moved to position P1 above the sample (SAFE_Z) previously defined by the user. At this height the machine can be moved freely since no obstructions are present, and the probe is moved 100 μm (App in Figure 3.1) away from the sample side in the x direction of contact (P2). Lastly the z axis is lowered to reach the correct height needed to perform the approach.

Once this point is reached the actual approach starts and the x axis is moved slowly (speed set by the input parameter) until a certain reading of the cary (target position) is red from the instrument.

To know the precise location of the sample the machine can proceed in two ways:

- the traditional way allows the user to control using the keyboard the machine to set the contact position manually

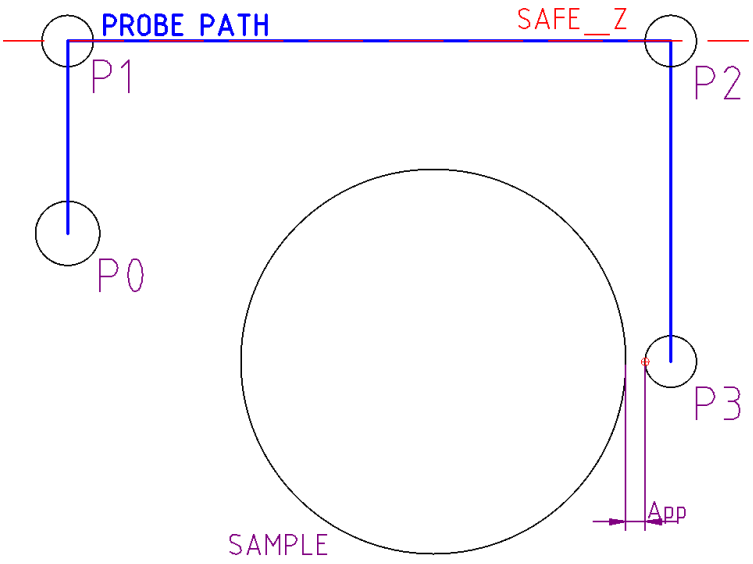


Figure 3.1: The approach path of the probe starting from an arbitrary position P0 to reach the position to start the approach.

- an aruCo marker can be placed solidarily to the sample to instruct the program using the webcam where it is located. This method must be refined in the final implementation since it does not allow for the initial position to be only 100µm from the side wall.

The routine also includes a safety check to ensure that the probe does not exceed a certain force threshold during the approach. If the force exceeds the threshold, the routine stops the movement and raises an exception.

3.2 Probe parameters

Two main parameters are used to define the probe: the probe diameter and the probe axle deformation. The probe diameter is the diameter of the probe tip, which is used to measure the diameter of the sphere. The probe axle deformation is the amount of deformation that occurs in the probe axle when a force is applied to it.

Knowing the exact length of the calibrated gauge block allows the system to determine the ball radius and the deformation of the axle. The difference between the measured value and the known block length provides the correction factor that will be used when measuring the sample under analysis. This calibration step is essential to correct any deviations and ensure the precision of subsequent

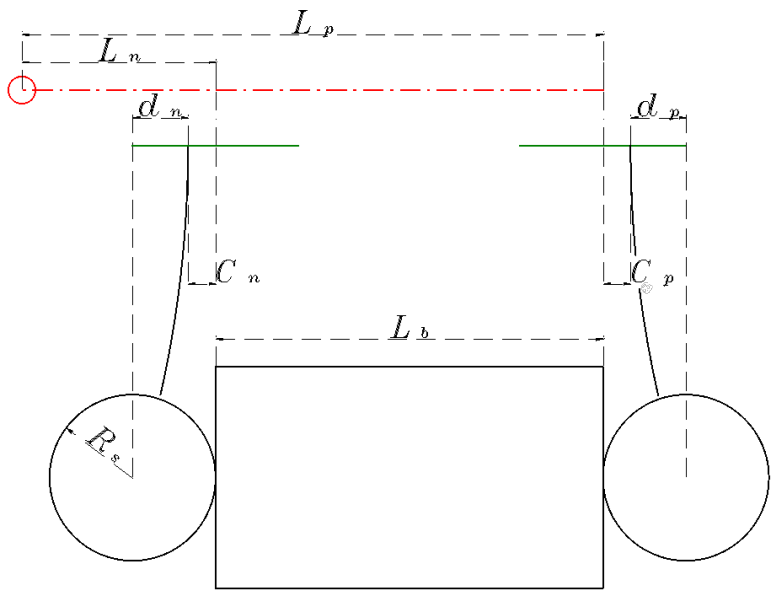


Figure 3.2: Calibration of probe radius and deformation using a calibrated gauge block.

measurements.

$$P_n = L_n - C_n - d_n + R_s \tag{3.1}$$

$$P_p = L_p + C_p + d_p - R_s \tag{3.2}$$

$$L_b = P_p - P_n \tag{3.3}$$

$$\text{Corr. Factor} = 2R_s - (d_p + d_n) = L_p + C_p - L_n + C_n - L_b \tag{3.4}$$

where R_s is the radius of the sphere, d_p is the positive deformation of the probe axle, d_n is the negative deformation of the probe axle, L_p is the laser measurement in the positive direction, L_n is the laser measurement in the negative direction, C_p is the cary measurement in the positive position, C_n is the cary measurement in the negative position and L_b is the length of the gauge block as shown in Figure 3.2.

3.3 Reading correction

The Cary reading is not calibrated once in a while as per usual practice for standard instrumentation, but it is calibrated every time a new measurement is performed by comparison with the interferometer reading. To do this a routine was created to perform a set of approaches of different forces (proportional to the approach target) and to store the Cary reading as well as the interferometer reading. The routine is called `caryFlex`. The Cary reading is corrected by a linear regression of the data acquired during the calibration routine [43].

First (as in Equation 3.5) the algorithm calculates the mean values of the input vectors x and y, where n is the size of the two vectors:

$$\begin{aligned} \bar{x} &= \frac{1}{n}\sum_{i=1}^{n} x_i \\ \bar{y} &= \frac{1}{n}\sum_{i=1}^{n} y_i \end{aligned} \tag{3.5}$$

Then the pearson correlation coefficient r (Equation 3.6) is calculated [44]:

$$\begin{aligned} W_{n1} &= \sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y}) \\ W_{n2} &= \sum_{i=1}^{n}(x_i - \bar{x})^2 \\ W_{n3} &= \sum_{i=1}^{n}(y_i - \bar{y})^2 \\ W_{n4} &= W_{n2} \cdot W_{n3} \\ r &= \frac{W_{n1}}{\sqrt{W_{n4}}} \end{aligned} \tag{3.6}$$

Finally in Equation 3.7 the α and β parameters can be calculated and they represent the regression line parameters with the line equation expressed as $y = \beta x + \alpha$:

$$\begin{aligned} S_y &= \sqrt{\frac{W_{n3}}{n-1}} \\ S_x &= \sqrt{\frac{W_{n2}}{n-1}} \\ \beta &= r \cdot \frac{S_y}{S_x} \\ \alpha &= \bar{y} - \beta\bar{x} \end{aligned} \tag{3.7}$$

The correction factor is calculated as the slope of the linear regression line both for right and left approaches, and it is used to adjust the Cary reading to match the interferometer reading, typical values of the correction range from $m = 0.98$ to $m = 1.02$ indicating that the Cary correction is minimal.

In the current implementation of the program, only the parameters alpha and beta are used to define the linear relationship established by the regression algorithm. It is important to note that also the parameters associated with the residuals, which quantify the discrepancies between the observed and predicted values, are not yet integrated into the program but they are already calculated in the regression routine. These residuals hold significant potential for future enhancements,

as they can be employed to assess the quality of the fit and evaluate the linearity correction of the probe.

3.4 Maximum determination

In order to measure the diameter it is necessary to position the probe in the maximum positions (left and right) of the sphere or cylinder relative to the probing direction. The program uses a method to find the maximum by moving the probe in the y and z directions. First the sample is approached on the right side at a predefined Cary reading, then the probe is moved in the positive y/z direction until 80% of the initial reading is reached, at this point the actual scan is performed. The motor that controls the scan axis switched direction and the istantaneous reading of the probe and of the axis is stored in two vectors.

The (findMax) algorithm is called to process the acquired values: first both the vectors are sorted based on the probe reading, from the highest to the lowest value. Then the scan axis sorted vector is parsed from the start until the value of the probe reading changes, at this point an average of all the scan positions found to have the max reading of the cary is calculated (Equations 3.8, 3.9).

$$\begin{aligned} x_{scan} &\in \mathbb{R}^n \rightarrow \text{The readings of the scan axis} \\ y_{scan} &\in \mathbb{R}^n \rightarrow \text{The readings of the cary during the scan} \\ x_m &\in \mathbb{R}^{n_m} = \{x_i \in x_{scan} \mid i = argmax(y_{scan})\} \end{aligned} \tag{3.8}$$

$$x_{max} = \frac{1}{n_m} \sum_{0}^{n_m} x_m \tag{3.9}$$

Where n_m is the number of points found at the maximum cary value.

This is the maximum position of the sphere for that specific scan. The same procedure is repeated in the other direction for both axes, and also for the other side of the sphere. A typical result of the algorithm is reported in figure 3.3.

3.5 Test programs

To test the various components of the machine multiple C++ programs were created. The programs are designed to test the various components of the machine, including the motors, encoders, and other sensors. Tests include:

- nilox_test.cpp: tests the webcam and the opencv routines used in the project
- moore_test.cpp: moves the machine to test all position instruments and motors
- keysight_test.cpp: tests the communication with the keysight interferometer

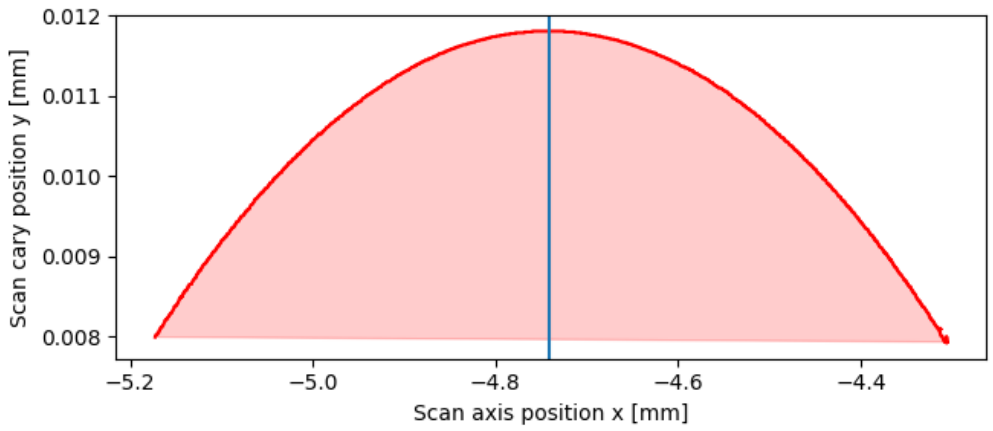


Figure 3.3: Maximum determination algorithm result. The red line is the probe reading, the blue line is the maximum position found.

- ik220_test.cpp: tests the communication with the ik220 controller for optical encoders
- cary_test.cpp: tests the communication with the cary controller and the reading of the cary probe
- axis_test.cpp: moves a single axis at the time
- marker_test.cpp: tests the marker recognition algorithm
- stm_test.py: performs a test of the STM32 microcontroller with a minimal GUI to control all motor parameters
- approach_test.cpp: tests the approach routine of the machine, moving the probe to a specific position in contact with a sample and reading the cary probe value
- caryFlex_test.cpp: tests the caryFlex algorithm to calibrate the cary probe reading
- cary_max_test.cpp: tests the cary probe maximum detection algorithm
- key_control_test.cpp: tests the keyboard functionality to move manually the machine

3.6 Measurement

To generalize the measurement process, the `Measurement` and `Sample` class hierarchy was implemented as shown in Figure 3.4.

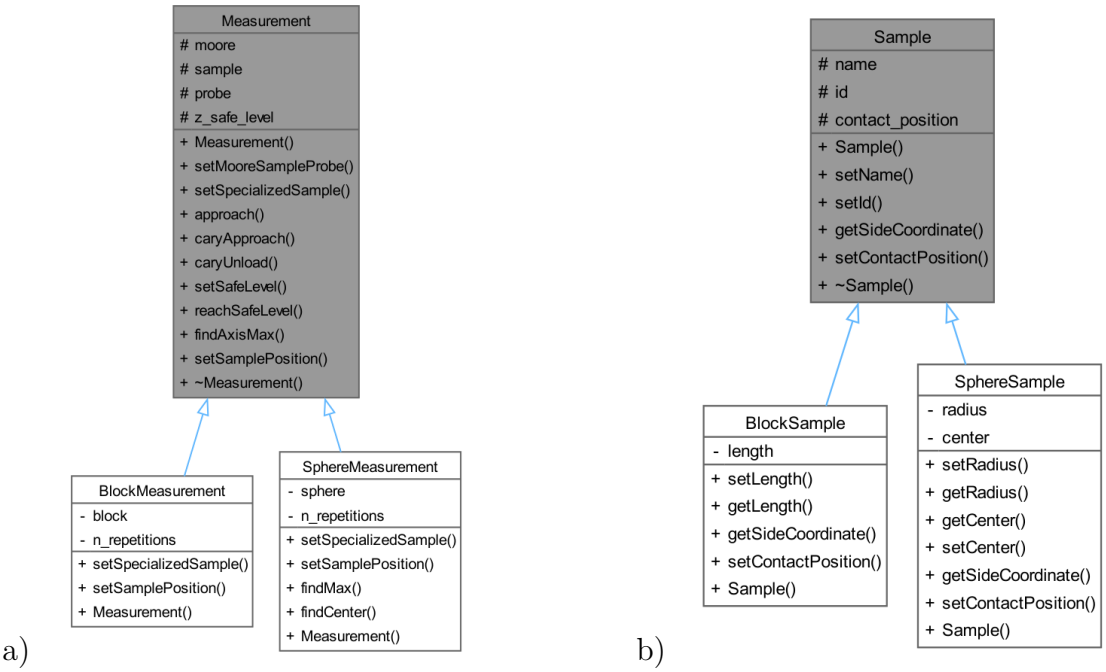


Figure 3.4: (a) The Measurement class structure, (b) The Sample class structure

This structure, illustrated in Figure 3.4, allows for the implementation of generalized methods (common to every type of measurement) in the base class `Measurement`, such as `approach` and `findAxisMax`. Specialized methods, on the other hand, are implemented in the derived classes. Some of the methods in the base class are virtual methods that must be redefined in the derived classes; however, they can still be invoked by the base class's generalized methods. This design promotes efficient code reuse and enhances maintainability.

Furthermore, the flexibility of this class hierarchy allows for the creation of additional derived classes that can specialize the measurement procedures and sample types beyond the block measurement and sphere measurement already implemented and shown in the figure. For instance, one could derive classes for cylindrical measurements, optical scales or even irregularly shaped samples. Each of these new classes can implement their own specific methods while still leveraging the common functionality provided by the base classes. This extensibility ensures that the measurement framework can adapt to various requirements and applications, facilitating future enhancements and integrations with new measurement sample types.

Theoretical note 16 *Virtual methods are a fundamental concept in OOP, particularly in languages like C++ and Java. They allow a base class to define a method that can be overridden by derived classes. When a method is declared as virtual in a base class, it enables polymorphism, which means that the method that gets executed is determined at runtime based on the type of the object being referenced, rather than the type of the reference itself. Virtual methods enable polymorphism, allowing a program to call derived class methods through base class pointers or references. This means that the correct method implementation is chosen based on the actual object type, not the type of the pointer/reference.*

3.6.1 Sphere diameter measurement

To test further the machine, the program to perform the measurement of the diameter of a sphere was created. The program is designed to measure the diameter of a sphere using the machine's sensors and encoders and it follows the steps reported in the following sections. The sphere is a 45 mm diameter marble ball, and it is used to calibrate the straightness of hte institute's stylus profilometer so the radius is well known and was calibrated many times over the years using the previous machine setup.

The setup used for the sphere measurement is shown in Figure 3.5. The setup consists of a sphere placed on a 3D printed holder and a gauge block.

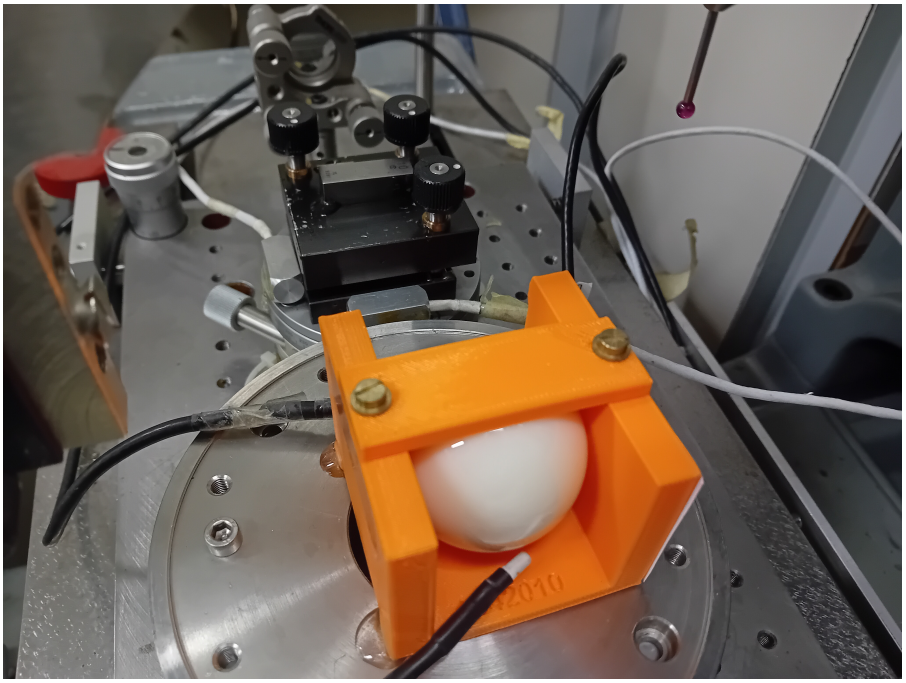


Figure 3.5: Setup for sphere measurement.

The procedure for measuring a sphere involves several critical steps to ensure accuracy and precision throughout the process. Initially, the user sets the Z safe value of the machine, which establishes a safe distance for the probe to avoid any potential collisions with the sample. Following this, the user manually adjusts the machine to find the point of contact with the sphere, whose nominal diameter is already known. This initial contact allows for the calculation of the sphere's center position, which serves as a reference for subsequent measurements.

Once the initial position is established, the machine is manually moved to a gauge block, where the contact position is set. This step is essential for performing the cary flex calibration and calculating the radius of the cary probe.

Next, the environmental parameters, such as temperature and humidity, are acquired and input into the Keysight laser reader. This information is crucial for correcting the refractive index of air, which can affect the accuracy of the measurements.

With the setup complete, the measurement procedure can commence. The machine systematically searches for the maximum points of the sphere's surface from both contact directions, across both axes, and in both scan directions. This approach results in a total of eight scans, ensuring that the measurement captures the sphere's geometry from multiple perspectives. The two maximum positions identified during this process are then averaged to refine the calculated position of the sphere's center. As a result, the contact points are updated and aligned with the newly calculated maximum positions.

To ensure consistency and reliability, the machine measures the sphere's diameter multiple times, with the number of repetitions (denoted as n) set by the user. This repeated measurement process helps to mitigate any potential errors and provides enough readings to estimate the measurement repetability. Additionally, the machine regularly updates the environmental parameters throughout the measurement process, ensuring that any fluctuations in conditions are accounted for in real-time.

If necessary, the radius of the cary probe can also be recalculated periodically during the measurement procedure. This flexibility allows for adjustments to be made in response to any changes in the system or environmental conditions.

3.6.2 Optical confocal probe measurement

This section of the testing involved the precise profilometric measurements on gear teeth utilized in the wind turbine industry (Figure 3.6).

The system is centered around a chromatic confocal point sensor (CCPS), designed to measure distances within a range of 100 microns [45]. This high-resolution CCP sensor is mounted on the Z-axis, allowing for vertical movement and accurate depth measurements (Figure 3.7 (a)).

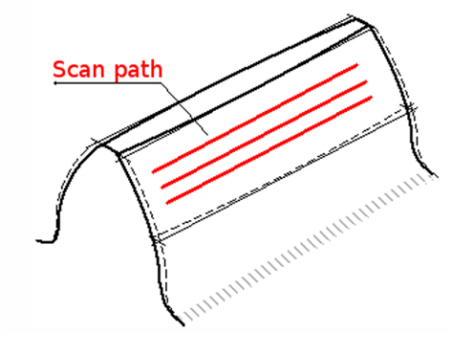
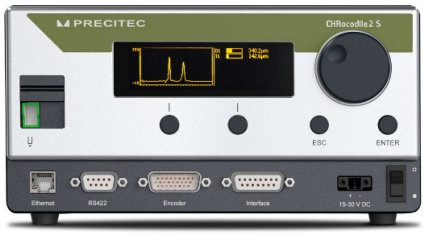
a)  b) 

Figure 3.6: Measured gear tooth: (a) the gear tooth, (b) the scan path in the x direction.

To ensure the accuracy and reliability of the measurements, the probe is characterized for noise by measuring a flat mirror and calculating the Power Spectral Density (PSD). This analysis provides insights into the sensor's performance and helps identify any potential sources of error. Additionally, the linearity of the probe is calibrated by rotating it along the X-axis and comparing the readings from the X interferometer. This calibration process is crucial for confirming that the sensor's output is consistent and reliable across its measurement range, ultimately enhancing the precision of the profilometric data collected from the gear teeth.

Theoretical note 17 *In CCPS (Chromatic confocal point sensor) technology white light is imaged through a chromatic lens to emit monochromatic light along the z-axis, when an object is present in this colour field, a single wavelength is fixed to its surface and then reflected back to the optical system. The backscattered beam passes through a filtering pinhole and is then acquired by a spectrometer. The beam's specific wavelength is calculated to precisely determine the position of the surface in the measurement field (Figure 3.7 (b)).*

a) 

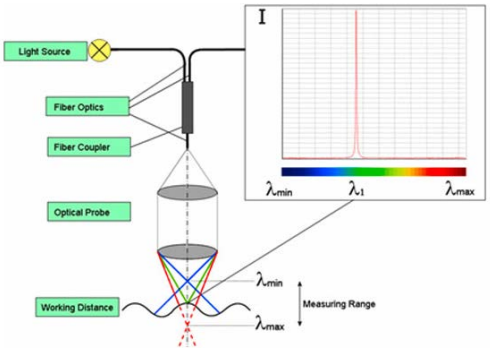
b) 

Figure 3.7: Chromatic confocal point sensor (a) the device used for the test measurement, (b) the technology concept behind its functionality.

To conduct the profilometric scans, the X-axis is programmed to move horizontally, enabling the sensor to record both the X and Z values of the confocal probe as it traverses the surface of the gear teeth. Given the limited measurement range of the sensor, a stitching technique is implemented to extend the effective measurement area. This technique involves repositioning the probe when it approaches 90% or 10% of its maximum range, ensuring that the sensor remains within its optimal operational limits while capturing comprehensive data across the entire surface (Figure 3.8).

The measurements were performed in both X directions to ensure comprehensive coverage of the gear tooth surfaces. Subsequently, the stitching method was refined by implementing a follow routine that maintained the probe at a consistent distance from the sample surface. This was achieved by continuously adjusting the Z-axis, allowing for real-time compensation of any variations in the surface profile. The follow routine was developed also using Python, incorporating a PID (Proportional-Integral-Derivative) controller to manage the movements of an additional X, Y stage. This setup enabled precise control over the probe's position relative to the sample, ensuring that it remained optimally aligned throughout the scanning process.

Theoretical note 18 *A PID, is a type of controller used to regulate variables such as temperature, speed, position, and pressure. The PID controller combines three control strategies to achieve optimal system response. The proportional component calculates the current error, which is the difference between the desired value (setpoint) and the actual measured value. The proportional response is directly related to this error; the larger the error, the greater the corrective action. The integral component takes into account the sum of past errors over time. By adding this component, the controller can correct persistent errors that are not resolved by the proportional action alone. The derivative component anticipates the future behavior of the error by calculating its rate of change. The derivative action helps to dampen oscillations and improve system stability by reducing the response to rapid changes in the error.*

$$u(t) = K_p \cdot e(t) + K_i \cdot \int e(t)\,dt + K_d \cdot \frac{de(t)}{dt}$$

where $u(t)$ is the controller output, $e(t)$ is the error at time t (the difference between the setpoint and the measured value), and K_p, K_i, and K_d are the proportional, integral, and derivative gains, respectively.

In this refined approach, the measurement results were derived from the combination of the sensor readings and the Z-axis optical encoder position readings. This integration of data sources enhanced the accuracy of the profilometric measurements, allowing for a more detailed and reliable representation of the gear tooth profiles, shown in Figure 3.8.

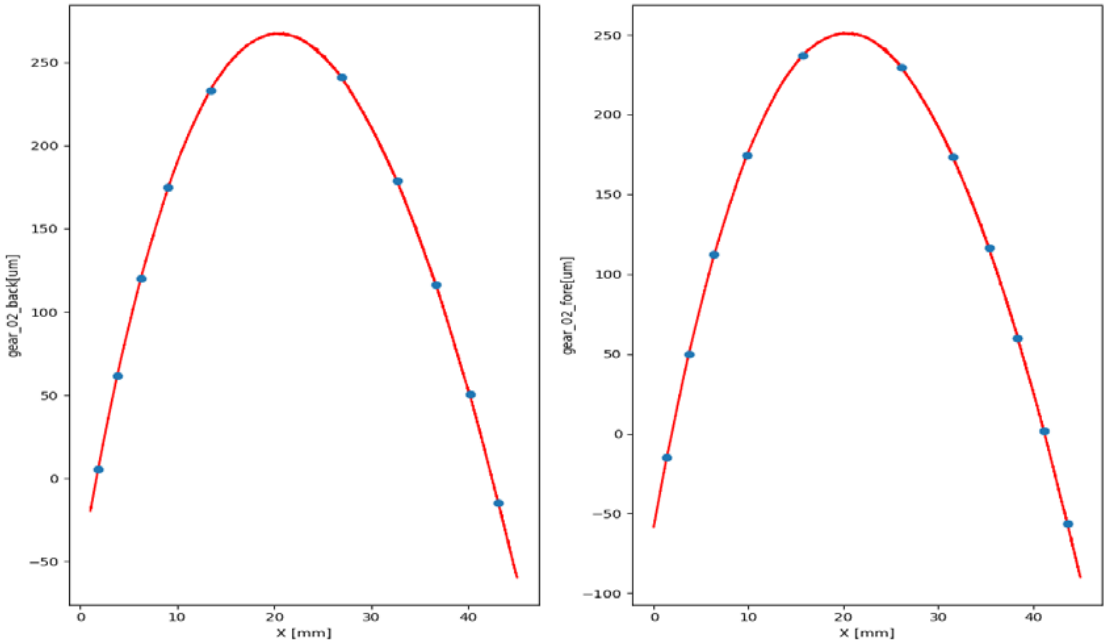


Figure 3.8: Measurement result in both directions with stitching points highlighted. Axes are not in scale.

Chapter 4

Uncertainty evaluation

4.1 Positioning error

To evaluate the positioning error of the machine, a program was developed to execute a series of movements within a restricted range of motion. The primary objective was to compare the actual arrival positions at each step with the predetermined target positions, thereby assessing the accuracy and reliability of the machine's positioning system.

In this study, a cubic volume measuring $3 \times 3 \times 3$ cm was divided into 5 discrete positions along each axis. This division resulted in a total of 125 test points, creating a three-dimensional grid of target positions. Each of these points served as a reference for the machine's movements, allowing for an evaluation of its performance across the defined volume and by estrapolation to all its volume.

To enhance the validity of the results, the positions were randomized without repetition. This randomization was crucial as it ensured that the machine's positioning was tested from a variety of angles and directions. By approaching each target position from different starting points, the evaluation captured a more comprehensive range of potential errors and variances in the machine's performance. This method not only mitigated the risk of bias that could arise from a fixed sequence of movements but also provided a more statistically robust dataset for analysis since during normal machine operation the direction of movement is unknown.

The program recorded the results in CSV format, which allowed the processing of data using a Python script. This script was designed to analyze the discrepancies between the target and actual positions, allowing for the calculation of positioning errors and the identification of patterns or trends in the machine's performance. The processing results are illustrated in Figure 4.1, which visually represents the distribution of errors across the test points. Green arrows represent the points displacement from ideal positioning and they were scaled up by a factor of 50 to make the discrepancy visible.

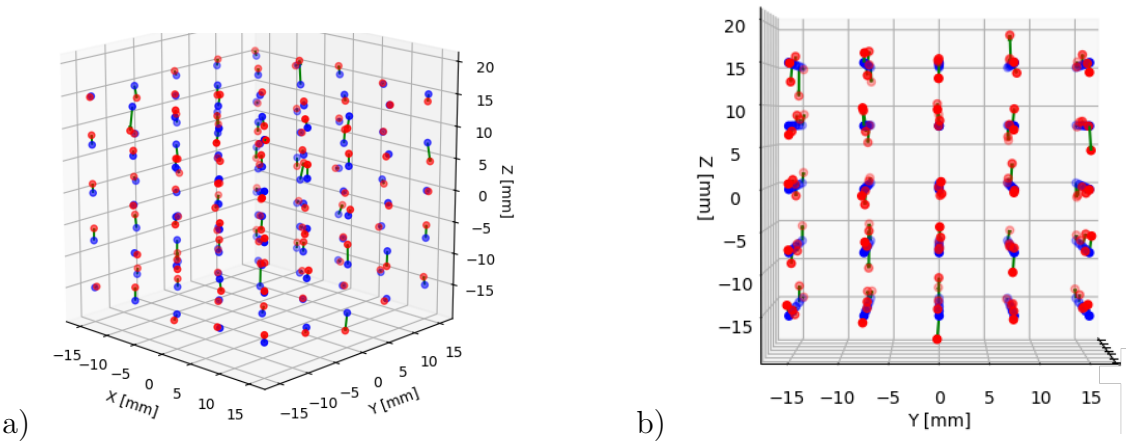


Figure 4.1: The target (blue) and actual (red) points set and reached by the machine: (a) An angled view of the points, (b) A side view to better show the point deviation. All deviations are in 50:1 scale.

4.1.1 Error parameters

The function `calculate_deviation_metrics` computes various metrics to quantify the deviation between a set of original points and their corresponding noisy points in a three-dimensional space. The metrics include overall displacement errors as well as axis-specific errors.

The function takes the following parameters:

- `original_points` P_o:
  - An array of shape $N \times 3$, where N is the number of points.
  - Each row represents a point in 3D space, with coordinates (x, y, z).
- `noisy_points` P_n:
  - An array of shape $N \times 3$, where N is the number of points.
  - Each row represents a noisy point in 3D space, with coordinates (x', y', z').

The following metrics are calculated in the function:

- `Average Displacement Error (ADE)`:
  - The average distance between each original point and its corresponding noisy point, calculated as:

$$\text{ADE} = \frac{1}{N} \sum_{i=1}^{N} \|P_{n_i} - P_{o_i}\| \tag{4.1}$$

- `Root Mean Square Error (RMSE)`:
  - The square root of the average of the squared distances, calculated as:

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\|P_{n_i} - P_{o_i}\|^2} \tag{4.2}$$

- `Maximum Displacement Error (MDE)`:
  - The maximum distance between any original point and its corresponding noisy point, calculated as:

$$\text{MDE} = \max_i \|P_{n_i} - P_{o_i}\| \tag{4.3}$$

The function returns the three results in Equations 4.1, 4.2 and 4.3 for the whole set of points and also calculates the same three metrics relative to each axis considering only the component relative to the axis under analysis.

The results obtained are reported in Tables 4.1 and 4.2.

Table 4.1: Overall deviation metrics between original and noisy points.

Metric	**Value** / µm
ADE	18.1
RMSE	18.6
MDE	30.3

Table 4.2: Deviation metrics for each axis (X, Y, Z).

Axis	**Metric**	**Value** / µm
X	ADE	1.0
	RMSE	1.4
	MDE	5.5
Y	ADE	0.5
	RMSE	0.6
	MDE	1.7
Z	ADE	18.0
	RMSE	18.5
	MDE	30.3

The results show that the machine's positioning system exhibits a maximum deviation of 30.3 micrometers, with an average deviation of 18.1 micrometers across all test points. The root mean square error (RMSE) is slightly higher at 18.6 micrometers, indicating that while most points are close to the target positions, there

are some outliers contributing to the overall error. This is mostly influenced by the Z axis, which has a maximum deviation of 30.3 micrometers and an average deviation of 18.0 micrometers. The X and Y axes show significantly lower deviations, with maximum deviations of 5.5 and 1.7 micrometers, respectively. This suggests that the machine's positioning system is more accurate in the X and Y directions compared to the Z direction probably due to the complex backlash and tension behavior of the belts that are used to move the Z axis (Section 2.1).

Since the main measurment direction is along the X axis, the Z axis error is not critical for the machine's operation but it could still be improved by implementind retensioning routines for the belts. Also it is important to discriminate between measurement and positioning uncertainty, in fact since the measurement is always done interferometrically and differentially, the positioning uncertainty is not critical for the measurement.

For each axis the bias metrics were also calculated, which are the average, minimum and maximum values of the displacement error in the direction of analysis. To do so the probability density function (PDF) of the displacement error was calculated for each axis and based on the measured values a kernel density estimation (KDE) was performed to obtain a smooth representation of the data [46]. The KDE was calculated using the `scipy` library in Python, which provides a convenient method for estimating the probability density function of a random variable. The resulting PDFs are shown in Figure 4.2.

Table 4.3: Bias metrics for each axis (X, Y, Z).

X	MEAN	0.09 µm
	VARIANCE	0.00 µm^2
	MIN	-2.5 µm
	MAX	5.5 µm
Y	MEAN	-0.01 µm
	VARIANCE	0.00 µm^2
	MIN	-0.88 µm
	MAX	1.7 µm
Z	MEAN	0.44 µm
	VARIANCE	0.36 µm^2
	MIN	-22.67 µm
	MAX	30.3 µm

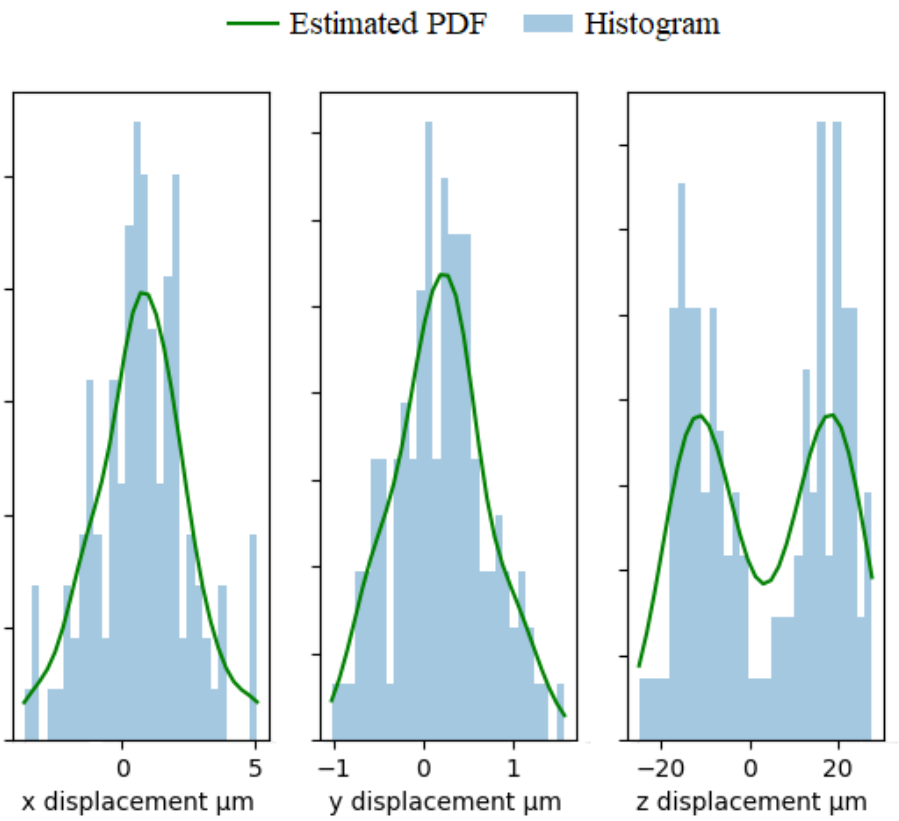


Figure 4.2: Probability density functions resulting from the single axis displacement analysis.

From the figure it is clear that the z displacement, which is the most affected by the machine's positioning error, even with a zero mean value, has a binormal distribution with a peak at -12 micrometers and another one at +19 micrometers. This is due to the fact that the machine is not able to reach the exact target position in z and the belts give way in both directions when moving the axis. This is a clear indication of the backlash present in the system, which is a common issue in mechanical systems with belts and pulleys. This confirms the hypothesis of the previous study.

The numerical results of the analysis are shown in Table 4.3.

4.2 Measurement uncertainty

The measurement uncertainty of the machine is a critical aspect that directly impacts the accuracy and reliability of the measurements obtained and it can be determined in two ways:

- **GUM analysis**: The GUM (Guide to the Expression of Uncertainty in Measurement) is a widely accepted framework for evaluating and expressing measurement uncertainty.
- **Monte Carlo analysis**: The Monte Carlo method is a statistical approach that uses random sampling to estimate the uncertainty associated with a measurement. This method is particularly useful for complex systems where analytical solutions may be difficult to obtain or where multiple sources of uncertainty are present.

The next sections will discuss the two methods in detail.

4.2.1 GUM analysis

In this section, will be discussed the various components contributing to the overall measurement uncertainty accoding to the GUM [47], including the uncertainties associated with the interferometer, the cary probe, and the environmental conditions. The analysis presented here is an adaptation of the uncertainty budget previously established for the machine before its redesign, focusing specifically on diameter measurements, which represent the machine's most frequent application.

The model equation for the diameter measurement is given by Equation 4.4:

$$D_{x,20} = \Delta L_x \pm D_{p,20} \pm \Delta P_x - D_x^* \cdot \frac{\vartheta_x^2}{2} + 2 \cdot \frac{a_x^2}{D_x^*} \mp \varepsilon_x + (b \cdot \varphi)_x + \rho_x - D_x^* \cdot \alpha_x \cdot \left(\overline{T_x} - 20\right) \quad (4.4)$$

where the $\pm$ sign indicates respectively whether the measurement is performed for internal diameters (upper sign) or external diameters (lower sign). and $D_{p,20}$ represents the diameter of the probe at 20 °C, which is measured with the gauge block (Equation 4.5):

$$D_{p,20} = \Delta L_g \mp L_{g,20} \pm \Delta P_g \mp L_{g,20} \cdot \frac{\vartheta_g^2}{2} \pm \varepsilon_g \pm (b \cdot \varphi)_g \pm \rho_g \mp L_{g,20} \cdot \alpha_g \cdot \left(\overline{T_g} - 20\right) \quad (4.5)$$

The parameters are defined as follows:

- $D_{x,20}$: Diameter at 20 °C.
- $\Delta L_{x,g}$: Interferometer reading difference in the two contanct positions (x for the sample contact, g for the gauge block contanct).

- $\Delta P_{x,g}$: Cary probe reading difference in the two contact positions (x for the sample contact, g for the gauge block contact).

- $L_{g,20}$: Gauge block length at 20 °C.

- D_x^*: Nominal diameter of the sample.

- $\vartheta_{x,g}$: Angle between the segment that unites the probed points and the actual sample diameter / block normal.

- a_x: Distance between the direction of measurement and the sample diameter.

- $\varepsilon_{x,g}$: Error in the cary probe reading due to the probe deformation.

- $(b \cdot \varphi)_{x,g}$: Abbe error (offset between the measurement direction and the laser beam) [48].

- $\rho_{x,g}$: Local form deformation (roundness / parallelism).

- $\alpha_{x,g}$: Coefficient of linear expansion of the sample / gauge block.

- $\overline{T_{x,g}}$: Average temperature of the sample / gauge block.

Once defined the measurement model and the parameters, the next step is to estimate the uncertainty associated with each component of the measurement. This is done by analyzing each parameter in detail and determining its contribution to the overall uncertainty based on the type of uncertainty (Type A or Type B) and the distribution of the data which will be measured in the first case and estimated from previous knowledge in the second case. The uncertainty components are summarized in Table 4.4, the distributions are marked with N if normal, and R if rectangular in the Dist column.

Table 4.4: Measurement uncertainty estimation for diameter measurement components.

Quantity	Estimate	Std. Uncertainty		Type	Dist.
ΔL_x	$\overline{\Delta L_x}$	$u(\Delta L_x)$	Equation 4.6		
ΔP_x	$\overline{\Delta P_x}$	$u(\Delta P_x)$	0.006 μm	B	R
ΔL_g	$\overline{\Delta L_g}$	$u(\Delta L_g)$	Equation 4.6		
ΔP_g	$\overline{\Delta P_g}$	$u(\Delta P_g)$	0.006 μm	B	R
$L_{g,20}$	$\overline{L_{g,20}}$	$u(L_{g,20})$		A	N
ϑ_x	0	$u(\vartheta_x)$	0,00006 rad	B	R
a_x	0	$u(a_x)$	4μm	B	R
ϑ_g	0	$u(\vartheta_g)$	0.00006 rad	B	R
ε_x	$\overline{\varepsilon_x}$	$u(\varepsilon_x)$	0.002 μm	B	R
ε_g	$\overline{\varepsilon_g}$	$u(\varepsilon_g)$	0.002 μm	B	R
$(b\cdot\varphi)_x$	0	$u((b\cdot\varphi)_x)$	0		
$(b\cdot\varphi)_g$	0	$u((b\cdot\varphi)_x)$	0		
ρ_x	0	$u(\rho_x)$	$0.002\cdot\Delta\rho_x\mu$m	B	R
ρ_g	0	$u(\rho_g)$	0		
α_x	$\overline{\alpha_x}$	$u(\alpha_x)$	$0.6\cdot10^{-6}{}^{\circ}\mathrm{C}^{-1}$	B	R
T_x	$\overline{T_x}$	$u(T_x)$	$0.6\cdot\overline{\Delta T_g}{}^{\circ}\mathrm{C}^{-1}$	B	R
α_g	$\overline{\alpha_g}$	$u(\alpha_g)$	$0.6\cdot10^{-6}{}^{\circ}\mathrm{C}^{-1}$	A	N
T_g	$\overline{T_g}$	$u(T_g)$	$0.6\cdot\overline{\Delta T_g}{}^{\circ}\mathrm{C}^{-1}$	B	R

$$
\begin{aligned}
u(\Delta L_{x,g}) &= \sqrt{\left(\frac{1}{n}\cdot\frac{\lambda_0}{2}\right)^2 u^2(\Delta N_{x,g}) + \left(-\Delta L_{x,g}\cdot\frac{1}{n}\right)^2 u^2(n) + \left(\frac{\Delta N_{x,g}}{2}\cdot\frac{1}{n}\right)^2 u^2(\lambda_o)} \\
&\cong \sqrt{\left(\frac{1}{n}\cdot\frac{\lambda_0}{2}\right)^2 u^2(\Delta N_{x,g}) + (\Delta L_{x,g})^2\, u^2(n) + (\Delta L_{x,g})^2\cdot\left(\frac{u(\lambda_o)}{\lambda_o}\right)^2}
\end{aligned}
\tag{4.6}
$$

Equation 4.6 can be split in three terms:

- The first term is the repeatability of the interferometer which can be derived from the instrument resolution $\frac{0.005}{\sqrt{3}}$, a typical value of $8nm$ is used.
- The second term is proportional to the length of the measurand and it is the uncertainty associated with the laser wavelength:

$$
u(n) = \sqrt{\left(\frac{\Delta n}{\Delta T_a}\right)^2 u^2(T_a) + \left(\frac{\Delta n}{\Delta P_a}\right)^2 u^2(P_a) + \left(\frac{\Delta n}{\Delta PH_a}\right)^2 u^2(PH_a)} \tag{4.7}
$$

  - For ambient temperature: $\dfrac{\Delta n}{\Delta T_a} = -9.3\cdot10^{-7}{}^{\circ}\mathrm{C}^{-1}$,
  - For ambient pressure: $\dfrac{\Delta n}{\Delta P_a} = 2.7\cdot10^{-9}\ \mathrm{Pa}^{-1}$,
  - For ambient humidity: $\dfrac{\Delta n}{\Delta H_a} = -4.3\cdot10^{-10}\ \mathrm{Pa}^{-1}$.

- The third term is the uncertainty associated with the laser wavelength which is given by the calibration certificate of the laser and it is: $\frac{u(\lambda_0)}{\lambda_0} = 10^{-8}$.

The next step is to calculate the combined uncertainty of the measurement using the law of propagation of uncertainty. The sensibility coefficients are calculated as the partial derivative of the model equation with respect to each parameter and are shown in Table 4.5.

Table 4.5: Sensitivity coefficients for the diameter measurement model.

$\mathbf{x_i}$	$\bar{\mathbf{x}}_\mathbf{i}$	$\mathbf{c_i} = \frac{\partial \mathbf{y}}{\partial \mathbf{x_i}}$	$\mathbf{u_i(y)} = \mathbf{c_i} \cdot \mathbf{u}\left(\mathbf{x_i}\right)_{[\mu\mathrm{m}]}$
ΔL_x	ΔL_x	1	0,008 μm
ΔP_x	ΔP_x	1	0,006 μm
ΔL_g	ΔL_g	± 1	0,008 μm
ΔP_g	ΔP_g	1	0,006 μm
$\overline{T_a}$	$\overline{T_a}$	$-0{,}93 \cdot 10^{-6} \cdot \Delta L_{x,g}\,[^\circ\mathrm{C}^{-1}\ \mathrm{m}]$	$-0{,}6 \cdot \Delta T_a \cdot \Delta L_{x,g}$
$\overline{P_a}$	$\overline{P_a}$	$0{,}27 \cdot 10^{-8} \cdot \Delta L_{x,g}\left[\mathrm{Pa}^{-1}\ \mathrm{m}\right]$	$0{,}0016 \cdot \Delta P_a \cdot \Delta L_{x,g}$
$\overline{H_a}$	$\overline{H_a}$	$-0{,}43 \cdot 10^{-9} \cdot \Delta L_{x,g}\left[\mathrm{Pa}^{-1}\mathrm{m}\right]$	$-0{,}00026 \cdot \Delta H_a \cdot \Delta L_{x,g}$
λ_0	λ_0	$\lambda_0^{-1} \cdot \Delta L_{x,g}$ [m]	$0{,}01 \cdot \Delta L_{x,g}$
$L_{g,20}$	$L_{g,20}$	± 1	0,010 μm
D_x^*	0	$-0{,}5 \cdot D_x^*$ [m]	$-0{,}0018 \cdot D_x^*$
a_x	0	$\frac{2}{D_x^*}$[m]	$\frac{0{,}00003}{D_x^*}$
ϑ_x	0	$0{,}5 \cdot L_g$[m]	$0{,}00185 \cdot L_g$
ε_x	ε_x	1	0,002 μm
ε_g	ε_g	1	0,002 μm
ρ_x	0	1	$0{,}2 \cdot \Delta\rho_x \mu$ m
α_x	α_x	$-D_x^* \cdot \left(\overline{T_x} - 20\right)$ [m°C]	$-0{,}6\left(\overline{T_x} - 20\right) \cdot D_x^*$
T_x	T_x	$-D_x^* \cdot \alpha_x$	$-0{,}6 \cdot \overline{\Delta T_x} \cdot \alpha_x \cdot D_x^*$
α_g	α_g	$\mp L_g \cdot \left(\overline{T_g} - 20\right)$ [m°C]	$+0{,}6 \cdot \left(\overline{T_g} - 20\right) \cdot L_g$
T_g	T_g	$\mp L_g \cdot \alpha_g$	$+0{,}6 \cdot 10^6 \cdot \overline{\Delta T_g} \cdot \alpha_g \cdot L_g$

Finally we can distinguish all the uncertainty contributions in two groups: the ones proportional to the measurand and the ones not proportional to the measurand. We can consider the length of the gauge block to be always equal to 10 mm, so the uncertainty associated with the gauge block is not proportional to the measurand.

The resulting CMC for the diameter measurement is given by the following Equation 4.8.

$$CMC(U/nm) = Q[0.1; 0.5E{-}03L] = \sqrt{(0.1)^2 + (0.5 \times 10^{-3} \times L)^2}, \text{L in mm} \quad (4.8)$$

4.2.2 Monte Carlo analysis

The Monte Carlo method [49] is a powerful statistical technique used to estimate the uncertainty of a measurement by simulating the measurement process multiple times with random variations in the input parameters. In this case, the Monte Carlo method was applied to the diameter measurement model to obtain an accurate estimate of the overall measurement uncertainty and validate the results obtained from the analytical approach.

The Monte Carlo simulation was performed using `monaco` [50], a Python library designed for uncertainty analysis. The simulation involved the following steps:

- Define the input parameters and their associated uncertainties based on the measurement model.
- Generate random samples for each input parameter using their respective probability distributions.
- Calculate the output (diameter) for each set of random samples using the measurement model explained earlier.
- Analyze the distribution of the output values to estimate the overall measurement uncertainty.

The input parameters were defined based on the measurement model, and their uncertainties were characterized using the distributions in Table 4.4 for most parameters. The simulation was run for 10,000 of iterations to ensure a robust estimate of the output distribution (Figure 4.3) and a seed [51] was chosen to be able to reproduce the results of the simulation over time. The results are reported in tables 4.6 and 4.7.

Table 4.6: Results for $D_{p,20}$.

Statistic	**Value**
Observations	10001
Min	6.0119 mm
Max	6.0121 mm
Mean	6.0120 mm
Sigma	0.034 μm
Variance	$1.18 \cdot 10^{-15}\ m^2$
Skewness	-0.0011
Kurtosis	0.0105

Table 4.7: Results for $D_{x,20}$.

Statistic	**Value**
Observations	10001
Min	20.0238 mm
Max	20.0242 mm
Mean	20.0240 mm
Sigma	0.049 μm
Variance	$2.37 \cdot 10^{-15}\ m^2$
Skewness	-0.0029
Kurtosis	0.0012

A plot of the simulation results is shown in Figures 4.4, 4.5. In the images the distribution of the results is shown on the y axis plots compared with the input

distributions on the x axis plots. The scatter plots in the middle indicate the sensitivity (partial derivative of the model function with respect to the input parameter) of the model to the input parameters. The points are the single iterations of the monte carlo simulation.

The results of the Monte Carlo simulation provide a comprehensive view of the measurement uncertainty associated with the diameter measurement process. The mean values obtained from the simulation closely match the expected values, confirming the accuracy of the model and the input parameters.

The simulation was carried out for a sample with a 20 mm diameter and a probe with a 6 mm diameter, which are the most common values used in the machine. Using the CMC calculation method we obtain the uncertainty in Equation 4.9.

$$CMC(U/nm) = Q[0.1; 0.5E-03L] = \sqrt{0.1^2 + (0.5E-03*20)^2} = 100\ nm \quad (4.9)$$

The Monte Carlo simulation results show a mean value of 20.02 mm with a sigma of 49 nm, resulting in an expanded uncertainty of 97 nm, which is consistent with the CMC value obtained from the analytical approach. The Monte Carlo method provides a more detailed understanding of the uncertainty distribution and allows for a more accurate estimation of the overall measurement uncertainty.

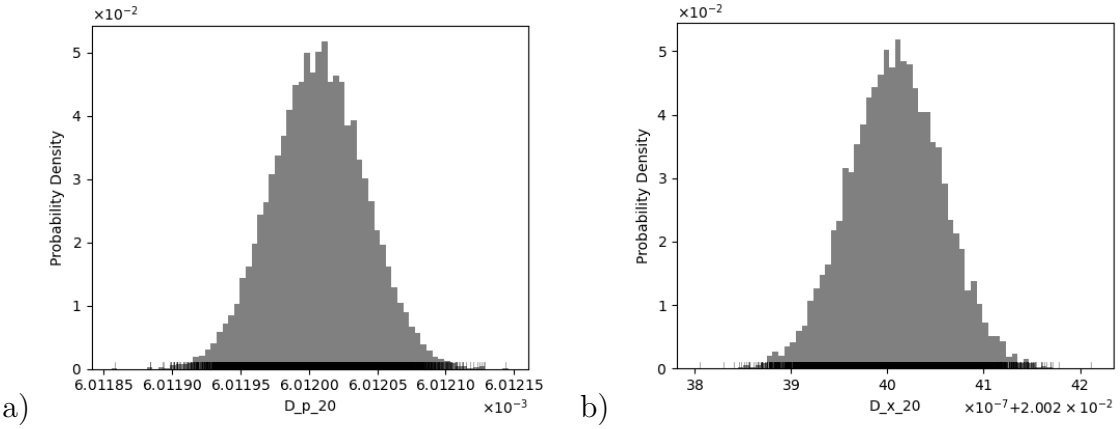


Figure 4.3: Result distributions: (a) probe diameter, (b) sample diameter.

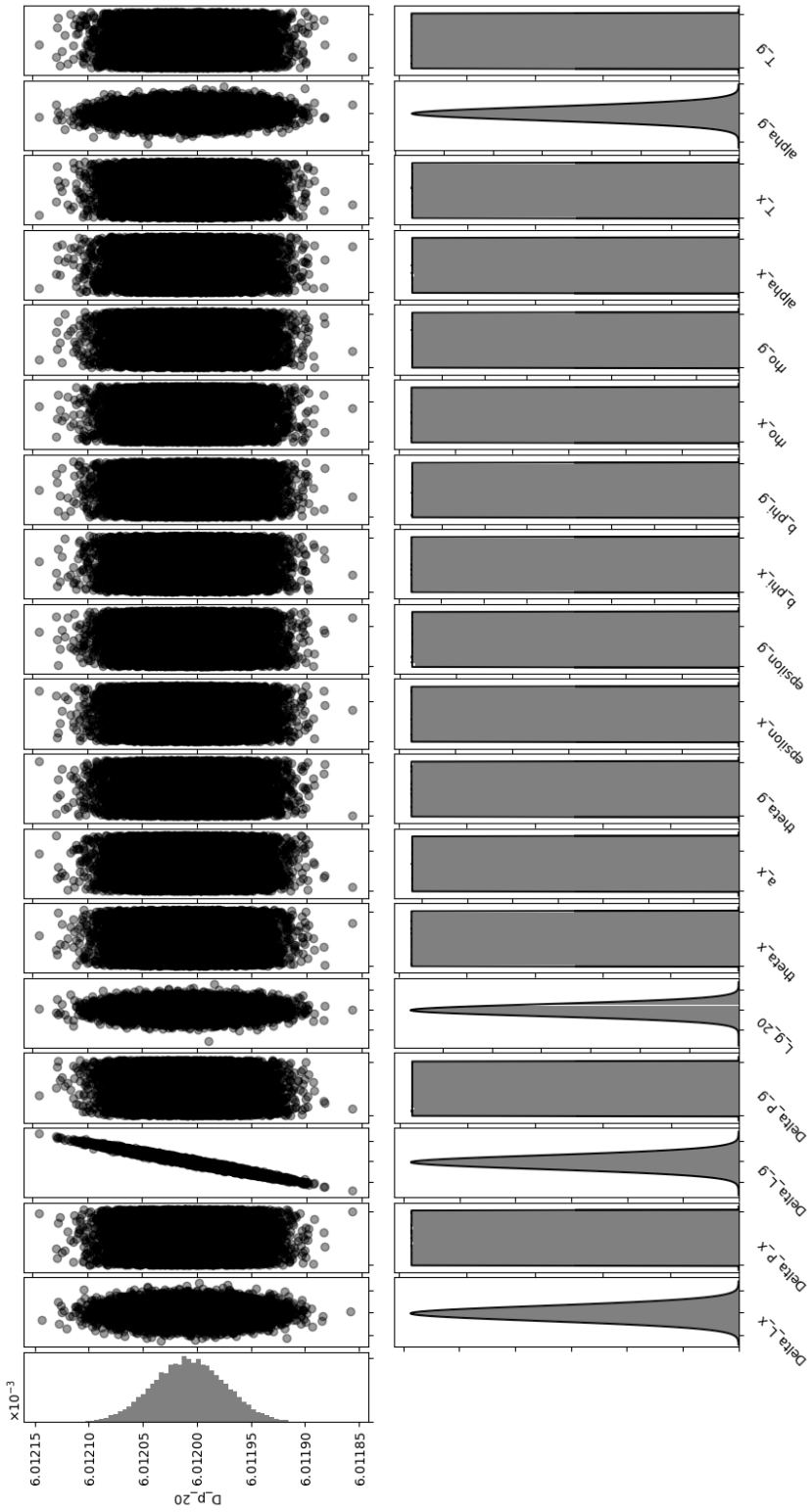


Figure 4.4: Monte Carlo simulation results with all test points for probe diameter uncertainty

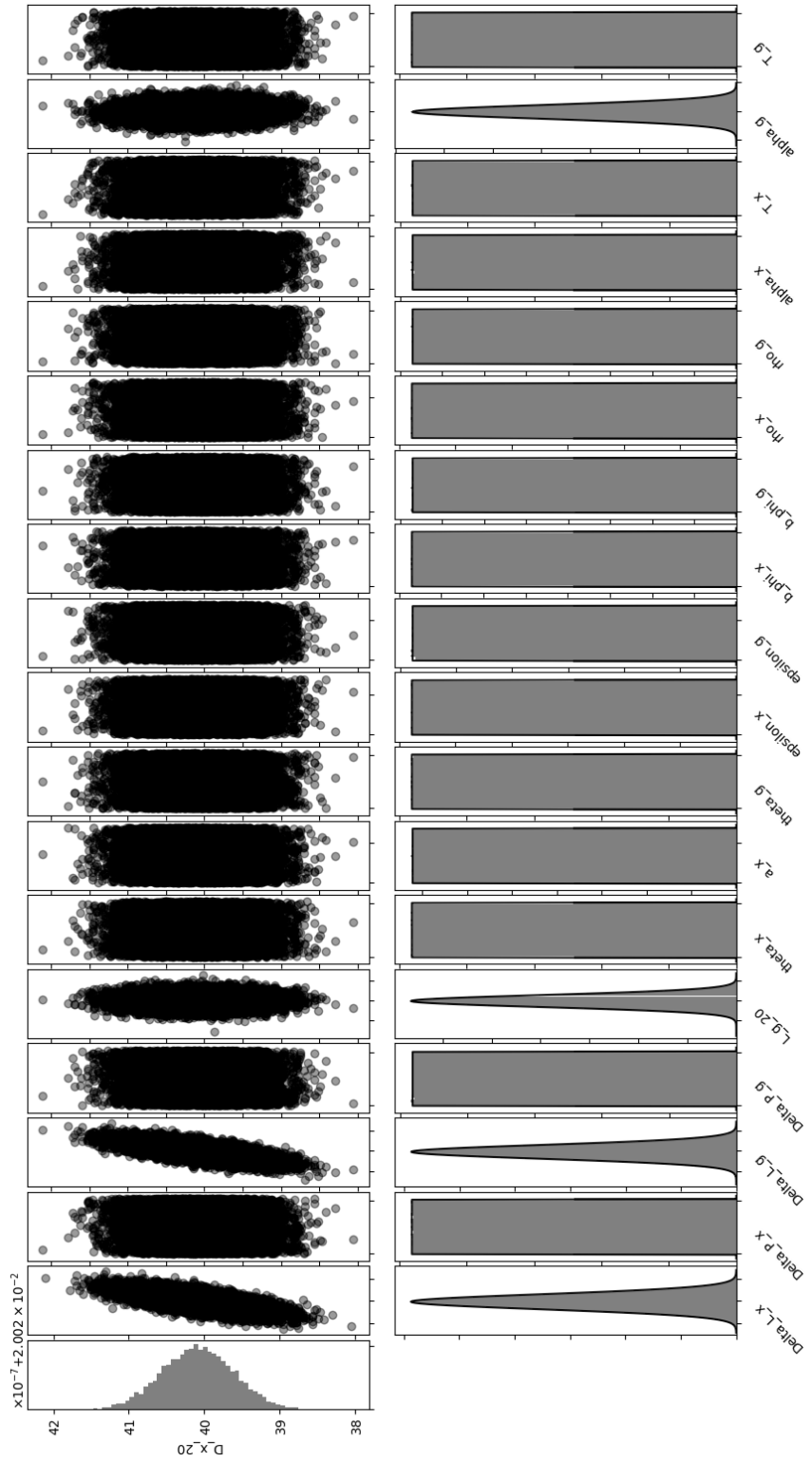


Figure 4.5: Monte Carlo simulation results with all test points for sample diameter uncertainty

Chapter 5

Conclusions

The primary objective of this thesis was to modernize the existing three-axis interferometric measuring machine, enhancing its capabilities for measuring diameters and optical scales through a comprehensive redesign and implementation of a new control system. This project has resulted in significant advancements in measurement reliability, and precision, achieved through the integration of advanced hardware and robust software solutions.

The necessity for this redesign arised from several critical flaws in the previous setup. The machine lacked a modern interface, which limited overall usability and made it difficult for operators to interact with the system efficiently. Additionally, the existing control program did not feature an intuitive user interface, rendering operation less accessible and potentially confusing for users. The absence of automation capabilities meant that many measurement processes required manual intervention, leading to a time-consuming and inefficient workflow. This reliance on manual operations increased the possibility of human errors, particularly during steps such as positioning the machine and aligning it with the sample, which were susceptible to inaccuracies. Furthermore, the manual positioning process required careful adjustments, introducing inconsistencies that could compromise measurement accuracy.

Key Contributions and Achievements

The modernization project has successfully delivered several key contributions. Notably, the complete redesign of the control system has facilitated the integration of new motors and drivers, which have improved the machine's operational capabilities. The development of a sophisticated C++ software program has enabled effective management of the instrumentation, incorporating features such as acceleration ramps and coordinate system definitions. Furthermore, the implementation of multithreading has allowed for simultaneous axis movement and environmental parameter monitoring, enhancing the overall efficiency of the measurement process.

The incorporation of OpenCV for coordinate system definition and ArUco marker detection has provided a novel approach to workpiece localization, while the development of specific measurement routines, including those for sphere diameter measurement, has demonstrated the machine's enhanced functionality. Additionally, the calibration routines established for probe radius and deformation have significantly improved measurement accuracy, and the application of the GUM method for uncertainty analysis has provided a quantitative framework for assessing the system's performance.

To validate the performance of the modernized measuring machine, a comprehensive suite of tests was conducted, focusing on fundamental operations such as sample approach, calibration of the Cary probe parameters, and the implementation of a maximum determination algorithm for locating extremes in measured samples. Specific test programs were developed to assess the functionality of individual components, including motors, optical encoders, and the Keysight interferometer. The integrated system was further assessed through profilometric measurements on gear teeth using an optical confocal probe.

The positioning uncertainty of the machine was characterized by testing movements within a defined volume, which revealed variations in accuracy across different axes. Additionally, the measurement uncertainty for diameter measurements was analyzed using the GUM method and Monte Carlo simulations, considering contributions from the interferometer, Cary probe, and environmental conditions. This thorough testing and uncertainty characterization provide a comprehensive evaluation of the modernized measuring machine's capabilities and limitations.

Challenges and Limitations

Despite these successes, the project faced several challenges and limitations that warrant acknowledgment. The analysis of positioning uncertainty revealed a maximum deviation along the Z-axis, attributed to potential backlash in the belts,while the X and Y axes demonstrated significantly lower positioning errors.While this issue is less critical for interferometric measurements, it highlights an area for future improvement.

A key finding from the uncertainty analysis is that the dominant contributions arise from the measurement of the probe radius and the sample radius, which are determined using the interferometer and the LVDT, respectively. Notably, the LVDT exhibits significantly lower precision compared to the interferometer, and this limitation directly impacts the overall measurement uncertainty. As a result, the precision of the LVDT represents a constraint in achieving lower uncertainty in the system's measurements.

The marker detection for workpiece localization, while functional, remains an approximation that requires refinement for more precise initial positioning. Furthermore, the parameters associated with the residuals of the linear regression used

for probe calibration have yet to be integrated into the program, representing an opportunity for future enhancement in assessing probe linearity correction.

Additionally, the current method of generating graphs for the GUI using Python outside the main interface suggests a need for further integration to enhance user experience.

Significance and Potential Impact

The modernized measuring machine holds significant potential for improving the reliability, traceability, and precision of measurements in various applications. By enhancing the quality and timing of calibration services, the upgraded system can facilitate precise measurements across research and industrial sectors. The advancements made in this project contribute to the broader field of metrology.

In conclusion, this thesis has successfully modernized the Moore measuring machine, resulting in a sophisticated system that significantly enhances measurement capabilities. The work presented herein not only addresses current limitations but also lays the groundwork for future advancements and applications in precision measurement. The potential for continued development and refinement underscores the importance of this project in the ongoing evolution of metrology.

5.1 Future developments

The control program of the machine is thought to be expanded for new functionality, future developements will include:

- the development of the routines for cylinder measurements
- the integration of an optical microscope to use instead of the cary for linear scale calibration
- the expansion of the GUI capabilities with graphs (now drawn with Python outside the interface)
- the redesign of the PCB to change the timer of the STM32 from a 16 bit counter to a 32 bit one
- the refinement of the movement and approach operation with the CS (microstepping) of the stepper motor
- the test of the entire setup and the time needed to perform a measurement for comparison with the previous setup

Data availability

All the source code and the data used in this work are available at `https://github.com/andeledea/Moore` [15]. No license was applied given the specific implementation and setup needed to utilize the code.

Bibliography

[1] Christian Schrader and Rainer Tutsch. «Calibration of a microprobe array». In: *Measurement Science and Technology* 23.5 (May 2012), p. 054001. DOI: 10.1088/0957-0233/23/5/054001. URL: https://iopscience.iop.org/article/10.1088/0957-0233/23/5/054001 (visited on 04/03/2025).

[2] Gianfranco Di Martino, Luca Dusini, Alberto Nicoletta, Fabio Lo Savio, Giuseppe Tuccari, and Michele Calì. «Easy Experimental Cylindricity Tolerance Verifications in Close-Range Bushings of Automotive Hinges». en. In: *Design Tools and Methods in Industrial Engineering IV*. Ed. by Paolo Di Stefano, Francesco Gherardini, Vincenzo Nigrelli, Caterina Rizzi, Gaetano Sequenzia, and Davide Tumino. Series Title: Lecture Notes in Mechanical Engineering. Cham: Springer Nature Switzerland, 2025, pp. 463–475. DOI: 10.1007/978-3-031-76597-1_49. URL: https://link.springer.com/10.1007/978-3-031-76597-1_49 (visited on 04/03/2025).

[3] *Metforwind*. en. Feb. 2020. URL: https://www.ptb.de/empir2020/met4wind/home/ (visited on 04/03/2025).

[4] *Laboratorio di misura di Campioni Lineari e Diametrali | INRIM*. URL: https://www.inrim.it/it/ricerca/settori-scientifici/metrologia-della-lunghezza/laboratori/laboratorio-di-misura-di-campioni (visited on 04/28/2025).

[5] *ISO 1:2022 - Geometrical product specifications (GPS) — Standard reference temperature for the specification of geometrical and dimensional properties*. en. URL: https://www.iso.org/standard/80702.html (visited on 04/29/2025).

[6] Martin Melichar, Dana Kubatova, and Jan Kutlwaser. «CMM Measuring Cycle and Human Factor». In: *DAAAM Proceedings*. Ed. by Branko Katalinic. 1st ed. Vol. 1. DAAAM International Vienna, 2016, pp. 0371–0376. DOI: 10.2507/27th.daaam.proceedings.055. URL: http://www.daaam.info/Downloads/Pdfs/proceedings/proceedings_2016/055.pdf (visited on 04/03/2025).

[7] *HP Computer Museum*. URL: https://hpmuseum.net/display_item.php?hw=33 (visited on 04/03/2025).

[8] *Turbo Pascal*. en. Page Version ID: 1282075903. Mar. 2025. URL: https://en.wikipedia.org/w/index.php?title=Turbo_Pascal&oldid=1282075903 (visited on 04/03/2025).

[9] *Olivetti computers*. en. Page Version ID: 1278632254. Mar. 2025. URL: https://en.wikipedia.org/w/index.php?title=Olivetti_computers&oldid=1278632254 (visited on 04/03/2025).

[10] Gian Bartolo Picotto, Roberto Belotti, Marco Pometto, and Marco Santiano. *The INRiM 1D comparator with a new interferometric set-up for measurement of diameter gauges and linear artefacts*. en. Tech. rep. Artwork Size: 193035 bytes, 4 pages Medium: application/pdf. Physikalisch-Technische Bundesanstalt (PTB), July 2013, 193035 bytes, 4 pages. DOI: 10.7795/810.20130620O. URL: https://oar.ptb.de/resources/show/10.7795/810.20130620O (visited on 02/06/2025).

[11] S. Carmignato, L. De Chiffre, H. Bosse, R.K. Leach, A. Balsamo, and W.T. Estler. «Dimensional artefacts to achieve metrological traceability in advanced manufacturing». en. In: *CIRP Annals* 69.2 (2020), pp. 693–716. DOI: 10.1016/j.cirp.2020.05.009. URL: https://linkinghub.elsevier.com/retrieve/pii/S0007850620301426 (visited on 04/03/2025).

[12] G B Picotto, M Aksulu, F Alqahtani, N Alqahtani, A A Arce, M Äremann, M Astrua, G Bajić, C Bandis, G Baršić, D Bellelli, R Bellotti, T Bozovic, D De Borst, V Duchoň, B Gastaldi, R Hanrahan, O Jusko, M Karanfilovic, M Katić, R Koops, G Kotte, A Lassila, M Matus, F Meli, I Meral, R Milne, R Muñoz, M Pometto, E Prieto, J Salgado, V Šimunović, J Spiller, G Szikszai, R Szumski, A Stelmaszczuk, D Teodorescu, R Thalmann, J B Toftegaard, and M Trösch. «Calibration of diameter standards (EURAMET.L-K4.2015)». In: *Metrologia* 58.1A (Jan. 2021), p. 04004. DOI: 10.1088/0026-1394/58/1A/04004. URL: https://iopscience.iop.org/article/10.1088/0026-1394/58/1A/04004 (visited on 04/04/2025).

[13] Shih-Ming Wang and Kornel F. Ehmann. «Measurement methods for the position errors of a multi-axis machine. Part 1: principles and sensitivity analysis». en. In: *International Journal of Machine Tools and Manufacture* 39.6 (June 1999), pp. 951–964. DOI: 10.1016/S0890-6955(98)00069-8. URL: https://linkinghub.elsevier.com/retrieve/pii/S0890695598000698 (visited on 04/29/2025).

[14] F Stanciu, D Nitoi, O Chivu, and A Tapardea. «Dimensional measuring of parts as function of temperature variation and FEM study». In: *IOP Conference Series: Materials Science and Engineering* 1182.1 (Oct. 2021), p. 012075. DOI: 10.1088/1757-899X/1182/1/012075. URL: https://iopscience.iop.org/article/10.1088/1757-899X/1182/1/012075 (visited on 04/29/2025).

[15] andeledea. *andeledea/Moore*. original-date: 2023-07-04T12:49:04Z. Apr. 2025. URL: https://github.com/andeledea/Moore (visited on 04/20/2025).

[16] *Moore Tool Company - Precision Machining*. en-US. URL: https://mooretool.com/ (visited on 03/28/2025).

[17] *All | lathes.co.uk*. URL: https://store.lathes.co.uk/model/all?page=7 (visited on 04/04/2025).

[18] *Homepage | INRIM*. URL: https://www.inrim.it/it (visited on 04/28/2025).

[19] *PKP564N28A2-TS30, 2.36 in. (60 mm) PKP Series 5-Phase Spur Gear (Offset Shaft) Stepper Motor (Gear Ratio: 30:1)*. en-us. URL: https://catalog.orientalmotor.com/item/pkp-series-5-phase-stepper-motors/pk-series-60mm-5-phase-stepper-motors/pkp564n28a2-ts30 (visited on 04/04/2025).

[20] Danielle Collins. *Microstepping for Stepper Motors*. en-US. Nov. 2017. URL: https://www.linearmotiontips.com/microstepping-basics/ (visited on 04/29/2025).

[21] *CVD528BR-K, CVD 5-Phase Stepper Motor Driver (24 VDC)*. en-us. URL: https://catalog.orientalmotor.com/item/cvd-5-phase-stepper-drivers/cvd-5-phase-stepper-motor-drivers/cvd528br-k?srsltid=AfmBOoqAJXAPG2ykINuprESDTX_zzRYisjrckoeZvKypl0VXDRZx5UeL (visited on 04/29/2025).

[22] *STM32 Nucleo boards - STMicroelectronics*. URL: https://www.st.com/en/evaluation-tools/stm32-nucleo-boards.html (visited on 03/28/2025).

[23] *Arduino - Home*. en. URL: https://www.arduino.cc/ (visited on 04/04/2025).

[24] *Getting started with STM32 system peripherals - stm32mcu*. URL: https://wiki.st.com/stm32mcu/wiki/Getting_started_with_STM32_system_peripherals (visited on 03/28/2025).

[25] *RS-232*. en. Page Version ID: 1274736209. Feb. 2025. URL: https://en.wikipedia.org/w/index.php?title=RS-232&oldid=1274736209 (visited on 04/04/2025).

[26] *KiCad EDA*. en-us. URL: https://www.kicad.org/ (visited on 03/28/2025).

[27] Keysight. *E1735A USB Axis Module*. en-IT. Section: Article Section. URL: https://www.keysight.com/it/en/product/E1735A/usb-axis-module.html (visited on 03/28/2025).

[28] *Misurazione computerizzata*. it-IT. URL: https://www.heidenhain.it/prodotti/convertitori-di-segnale/misurazione-computerizzata (visited on 03/28/2025).

[29] *Engineering Metrology Toolbox*. URL: https://emtoolbox.nist.gov/wavelength/Documentation.asp (visited on 04/29/2025).

[30] *OpenCV: Camera Calibration*. URL: https://docs.opencv.org/4.x/dc/dbb/tutorial_py_calibration.html (visited on 03/28/2025).

[31] *OpenCV: Camera Calibration and 3D Reconstruction*. URL: https://docs.opencv.org/4.x/d9/d0c/group__calib3d.html (visited on 03/28/2025).

[32] *OpenCV: Camera Calibration*. URL: https://docs.opencv.org/4.x/dc/dbb/tutorial_py_calibration.html (visited on 02/06/2025).

[33] *OpenCV: Detection of ArUco Markers*. URL: https://docs.opencv.org/4.x/d5/dae/tutorial_aruco_detection.html (visited on 03/28/2025).

[34] *Doxygen homepage*. URL: https://www.doxygen.nl/index.html (visited on 03/28/2025).

[35] *LaTeX - A document preparation system*. URL: https://www.latex-project.org/ (visited on 04/23/2025).

[36] *Clay - UI Layout Library*. URL: https://www.nicbarker.com/clay (visited on 03/28/2025).

[37] Sean Barrett. *nothings/stb*. original-date: 2014-05-25T16:51:23Z. Mar. 2025. URL: https://github.com/nothings/stb (visited on 03/28/2025).

[38] *raylib*. en. URL: https://www.raylib.com (visited on 03/28/2025).

[39] *WebAssembly*. en. URL: https://webassembly.org/ (visited on 03/28/2025).

[40] *Immediate UI vs Retained UI*. en. URL: https://collquinn.gitlab.io/portfolio/my-article.html (visited on 04/23/2025).

[41] *wxWidgets: Cross-Platform GUI Library*. URL: https://www.wxwidgets.org/ (visited on 03/28/2025).

[42] *wxFormBuilder/wxFormBuilder*. original-date: 2016-09-01T17:41:41Z. Mar. 2025. URL: https://github.com/wxFormBuilder/wxFormBuilder (visited on 03/28/2025).

[43] George Maier. *georgemaier/simple-linear-regression*. original-date: 2020-04-22T08:13:05Z. Oct. 2024. URL: https://github.com/georgemaier/simple-linear-regression (visited on 04/19/2025).

[44] *Pearson Correlation and Linear Regression*. URL: https://sites.utexas.edu/sos/guided/inferential/numeric/bivariate/cor/ (visited on 04/19/2025).

[45] Precitec GmbH & Co KG. *Misura spessore strato | CHRocodile 2S|SE|HS | PRECITEC*. it. URL: https://www.precitec.com/it/visual-metrologia-3d/prodotti/sensori-a-punto/chrocodile-2s-2se-2s-hs/ (visited on 04/29/2025).

[46] *Kernel density estimation*. en. Page Version ID: 1285981995. Apr. 2025. URL: https://en.wikipedia.org/w/index.php?title=Kernel_density_estimation&oldid=1285981995 (visited on 04/22/2025).

[47] *Evaluation of measurement data — Guide to the expression of uncertainty in measurement*. 2008. DOI: 10.59161/JCGM100-2008E. URL: https://www.bipm.org/doi/10.59161/JCGM100-2008E (visited on 04/29/2025).

[48] Richard Leach. «Abbe Error/Offset». en. In: *CIRP Encyclopedia of Production Engineering*. Ed. by The International Academy For Produ, Luc Laperrière, and Gunther Reinhart. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 1–4. DOI: 10.1007/978-3-642-35950-7_16793-1. URL: https://link.springer.com/10.1007/978-3-642-35950-7_16793-1 (visited on 04/22/2025).

[49] *Evaluation of measurement data — Supplement 1 to the "Guide to the expression of uncertainty in measurement" — Propagation of distributions using a Monte Carlo method*. 2008. DOI: 10.59161/JCGM101-2008. URL: https://www.bipm.org/doi/10.59161/JCGM101-2008 (visited on 04/29/2025).

[50] *scottshambaugh/monaco: Quantify uncertainty and sensitivities in your computer models with an industry-grade Monte Carlo library.* URL: https://github.com/scottshambaugh/monaco/tree/main (visited on 04/23/2025).

[51] *Random seed*. en. Page Version ID: 1281892956. Mar. 2025. URL: https://en.wikipedia.org/w/index.php?title=Random_seed&oldid=1281892956 (visited on 04/29/2025).