PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA

MINISTRY OF HIGHER EDUCATION AND SCIENTIFIC RESEARCH

AMAR TELIDJI UNIVERSITY- LAGHOUAT

FACULTY OF TECHNOLOGY

ELECTRONICS SECTOR

ELECTRONICS DEPARTEMENT

Dissertation Master Degree in Instrumentation

Presented by: 1- BENMOUSSA AHMED
2- KHELLOUFI HADJ AISSA

Numerical study and optimization of a digital temperature sensor using an Arduino assembly and data modeling with Python.

Board of Examiners:

Mr : M. Birane	MCA	Chairman	UATL
Mr : A. BELLAKHDAR	MCA	Examiner	UATL
Mr : A. MOUHOUB	MCA	Supervisor	UATL

2023/2024

Acknowledgments

We thank Almighty Allah for giving us the courage, will, health and patience to complete this work.

I would like to express my sincere gratitude to my supervisor, ***Dr. Abdelhafid Mouhoub****, for their invaluable guidance and support throughout my master's project. Their expertise and encouragement helped me to complete this research and write this project.*

I would also like to express my sincere appreciation to the members of the jury for their interest in my research and for agreeing to evaluate and enhance my work with their valuable suggestions. Their contribution is highly valued and appreciated.

Furthermore, I would also like to thank my friends and family for their love and support during this process. Without them, this journey would not have been possible.

Finally, I would like to thank all of the participants in my study for their time and willingness to share their experiences. This work would not have been possible without their contribution.

Contents

Chapter III: Realisation and discussions

List of symbolse

Symbol	Definition
PTC	Positive Temperature Coefficient
NTC	Negative Temperature Coefficients
C	Celsius
K	Kelvin
F	Fahrenheit
PWM	Pulse width modulation
RTD	Resistance Temperature Detector
USB	universal serial bus
PCB	Printed circuit board
DC	Direct Current
AC	Alternating Current
SPI	Serial Peripheral Interface
arduino ide	Arduino Integrated Development Environment
T	Temperature
ln	Natural logarithm
e	Exponential function
β	Beta
Tref	Temperature refence
Tmes	Temperature measured

List of Figures

General Introduction

GENERAL INTRODUCTION

We all use temperature sensors in our daily lives, be it in the form of thermometers, domestic water heaters, microwaves, or refrigerators. Usually, temperature sensors have a wide range of applications, the geotechnical monitoring field, being one of them.

Temperature sensors are designed to keep a regular check on concrete structures, bridges, railway tracks, soil, etc. Here we are going to tell you what is a temperature sensor, how it works, where is it used, and what are its different types [1].

A temperature sensor is a device that is designed to measure the degree of hotness or coolness in an object. The working of a temperature meter depends upon the voltage across the diode. The temperature change is directly proportional to the diode's resistance. The cooler the temperature, the lesser will be the resistance, and vice-versa [2].

The resistance across the diode is measured and converted into readable units of temperature (Fahrenheit, Celsius, Centigrade, etc.) and, displayed in numeric form over readout units. In the geotechnical monitoring field, these temperature sensors are used to measure the internal temperature of structures like bridges, dams, buildings, power plants, etc [3].

The microcontroller has the ability to read and process temperature information. We use the Firmata protocol, which facilitates communication between the microcontroller and the computer. We will link the microcontroller to the sensor.

This serial monitor is quite limited: it only allows real-time visualization of a series of values. For to go further, we propose the use of a Python script, which will have the role of retrieving the data sent by the Arduino from the serial port, and representing it graphically.

The main objective of this work is to:

- Study the response of a device modeled by a temperatures sensor.
- Determine the physical characteristics of a temperature sensor using a microcontroller and the Python programming language. This explains and justifies the research concerning the study of digital design, modeling and simulation programs.

To this objective, this work is approached according to the following chapters:

- The first is to define the temperature and the different units of measurement as well as the different types of sensors.
- The second is to gather sufficient information on a large category of interface maps (Arduino): its programming language, its construction, its principle of operation and how firmata protocol communicates with microcontroller from software on host computer.
- The third is to realize an electrical wiring capable of performing an action between a sensors and an interface card (Arduino) by explaining the different activities of its construction.
- Finally, we finish our work with a general conclusion.

Chapter I:

Temperature Sensors_ Types, Uses, Benefits and Design

I.1 Introduction

The temperature sensors are devices that detect and measure coldness and heat and convert it into an electrical signal. Temperature sensors are utilized in our daily lives be it in the form of domestic water heaters, thermometers, refrigerators, or microwaves. There is a wide range of applications of temperature sensors, including the geotechnical monitoring field.

A temperature sensor can also be defined as a simple instrument that measures the degree of coldness or hotness and then converts it into a readable unit. There are specialized temperature sensors used to measure the temperature of the boreholes, soil, huge concrete dams, or buildings.

This chapter will discuss what temperature sensors are, their uses, their components, and how they function, will discuss the different classes and types of temperature sensors as applied in control and compensation circuits and temperature sensor elements. It will also discuss the applications and benefits of temperature sensors and discuss temperature control using temperature sensors.

I.2 General information on temperature measurement

The temperature of an object is determined by the subjective perception of heat or cold when it is touched. Temperature is a quantitative measure of the average amount of energy possessed by the particles in a given sample of matter. It is often expressed in degrees on a standardized scale. Temperature can be measured using many methods, which range in terms of equipment cost and precision. A sensor can be defined as a device that enables us to measure a physical quantity by typically converting it into an electrical signal. Converting the measurement into an electrical signal is advantageous since it provides an output that is straightforward to handle. Sensors operate by detecting a physical occurrence that can provide information about changes in a physical property of interest [4].

The temperature is a significant physical quantity that has a high level of technological importance due to its changing nature. Understanding the temperature is crucial for ensuring the quality of various products, since it plays a significant role in determining many technical processes. Using a temperatureis essential to ensuring the quality of various items. Thermistances are a type of temperature sensor. These sensors rely on the characteristics of specific semiconductors to exhibit changes in electrical resistance in response to temperature variations. We are discussing the positive coefficient thermistor (CTP) and the negative coefficient thermistor (CTN). Additionally, we can discuss captures that occur as a result of

the expansion of solid or gaseous objects. As part of our project, we developed a temperature sensor utilizing the LM35 component.[5,6]

I.2.1 Temperature

The equation governing the transfer of heat between multiple systems also allows for its analysis. Additionally, we can understand it through the concept of entropy in thermodynamics, where we view it as a function that increases with the level of thermal agitation of particles, linked to the degree of disorder. Temperature is a fundamental concept that holds great significance in various domains, including metrology, climatology, medicine, and chemistry. Recognizing the limitations of traditional temperature quantification is crucial. This means that there is no standard that would be a comparison tool. This results in the erroneous statement that a body's temperature doubles, either by double or by double. We claim that temperature is an observable and non-measurable magnitude, but we use scales to assess temperatures. Saying that temperature is a vast and non-intensive magnitude is another way to put this [7].

I.2.2Temperature scales

I.2.2.1 Fahrenheit

The European physicist Daniel Gabriel Fahrenheit (1686–1736) first created the Fahrenheit scale, a temperature scale, in 1724 [8].The unit of measurement utilized is the degree Fahrenheit (symbol: °F). He initially developed his scale in multiple versions, but the original publication indicates that he determined the lowest reference point, 0 °F, as the temperature at which a brine solution, made from a combination of water, ice, and ammonium chloride (a type of salt), freezes. Two fixed values, 180 °F apart, determined the Fahrenheit scale in the 20th century: the freezing point of pure water at 32 °F and the boiling point of water at 212 °F, both at sea level and at standard atmospheric pressure. It is currently defined in a formal manner using the Kelvin scale [9].

I.2.2.3 Celsius

One of two temperature scales used in the International System of Units (SI), the other being the closely related Kelvin scale, is the Celsius temperature scale, which was previously known as the centigrade scale outside of Sweden. The degree Celsius is the unit of temperature on the Celsius temperature scale. You can describe a range of temperatures or a specific point on the Celsius temperature scale using the degree Celsius, represented by the symbol °C. It bears the name Anders Celsius in honor of the Swedish astronomer (1701–1744) [9].

Throughout the 19th century, scientists measured the freezing point of water at 0 °C and the boiling point of water at 1 atm pressure at 100 °C. (When Celsius first proposed the concept, the boiling and freezing points were, respectively, 0 and 100 degrees.).

Between 1954 and 2019, the unit degree Celsius and Celsius temperature scale were defined based on absolute zero and the triple point of water. Since 2007, the Celsius temperature scale has been defined in relation to the kelvin, which is the fundamental unit of thermodynamic temperature in the International System of Units (symbol: K). Absolute zero, the coldest possible temperature, is currently specified as precisely 0°K and −273.15 °C [10].

I.2.2.4 Kelvin

The kelvin, symbolized by the sign K, serves as the fundamental unit of measurement for temperature in the International System of Units (SI). The Kelvin scale is a thermodynamic temperature scale that begins at 0 K, which represents the lowest attainable temperature (absolute zero), and increases by precisely 1 K for every 1 °C. You can easily convert the Celsius scale (symbol °C) to the Kelvin scale. To convert to kelvin, add 273.15 to any temperature in degrees Celsius [10, 11].

I.3 Classes and Types of Temperature Sensors

Electrical sensors offer the benefit of increased application versatility (information, transmission, and recording) with enough accuracy for industrial and numerous laboratory applications. The electrical temperature sensors can be categorized into two categories:

- Active sensors, with thermoelectric torque
- Passive sensors, resistance or thermistor sensors.

I.3.1 Active sensors

An active sensor, which functions as a generator, is typically based on a physical phenomenon that guarantees the transformation of the energy temperature to be measured into electrical energy.

Based on the intensity and temperature recorded, an active sensor's output signal can be a voltage, current, or load amount. The thermoelectricity and physical effect used by active sensors to measure temperature.

I.3.2 Passive sensors

These are impedances in general when one of the deciding parameters is temperature-sensitive, and the following terms are found in the literal expression of this impedance:

- The dimensions or the shape of this sensor
- Electrical characteristics of the materials utilized, like permeability and resistivity

- Magnetic or other types of constants.

Passive sensors use the resistivity and, at very low temperatures, the dielectric constant as the physical effect to measure temperature.

Temperature sensors are found in different types, sizes, and shapes. There are two main temperature sensors classes: contact temperature sensors and non-contact temperature sensors.

I.3.3 Contact Temperature Sensors

A few temperature meters are capable of measuring the degree of hotness or coldness in an object by being in direct contact with the object. These types of temperature sensors fall under the class of contact-type sensors. They can be utilized in the detection of liquids, solids, or gasses over a broad range of temperatures.

I.3.4 Non-Contact Temperature Sensors

These types of temperature meters do not measure the temperature of an object while in direct contact; rather, they measure the degree of hotness or coldness from the radiation that is emitted by the heat source.

I.4 Principle of Temperature Sensors

The contact and non-contact temperature sensors are divided further into the following mentioned types of temperature sensors.

I.4.1Thermistor

A semiconductor material forms a thermistor, a type of resistor. Temperature significantly influences its resistance, more so than in regular resistors. The term "thermistor" is a combination of the words "thermal" and "resistor". Depending on materials used, thermistors are classified into two types [4]:

Figure I.1: Negative-temperature-coefficient (NTC).

I.4.1.1Thermistance types:

Thermistors are categorized into two types based on the materials they are made of :

- ✓ Negative Temperature Coefficient (NTC) Thermistor (NTC thermistors) exhibit a phenomenon where their resistance reduces as the temperature increases. Thermal agitation primarily causes this behavior by energetically elevating the number of conduction electrons from the valence band. When connected in series with a circuit, people frequently employ an NTC, also known as a negative temperature coefficient thermistor, as a temperature sensor or as an inrush current limiter.

  A thermistor is a temperature sensor that reacts to minute temperature changes. At very low temperatures, it provides high resistance. As the temperature increases, the resistance quickly drops. The glass coated NTC thermistor offers the highest accuracy. The smallest change in resistance per degrees of Celsius is immediately displayed. Negative temperature coefficient thermistors require linearization due to their use of the exponential working principle, where the temperature range is between -50°C to 250°C. NTC thermistors also require linearization because of the size and speed involved. The use of a NTC thermistor in a detection circuit requires the use of a small voltage to be passed across the thermistor. The temperature will be reflected by the thermistor's resistance, which rapidly drops as the temperature increases. For applications in the range of -40°C to 125°C, NTC thermistors provide an option that is less expensive than platinum RTDs, thermocouple sensors, and semiconductor based sensors. NTC's fast changing resistance base makes them capable of providing superior accuracy, power efficiency, stability, reliability and responsiveness. They can easily be integrated into any system [4,5].

- ✓ PTC thermistors exhibit a positive temperature coefficient, meaning that their resistance increases as the temperature increases. The heightened thermal lattice agitations, particularly those caused by impurities and defects, primarily cause this phenomenon. Frequently, circuits connect PTC thermistors in series, acting as resettable fuses to protect against overcurrent situations.

The materials used to make resistors and resistance temperature detectors (RTDs) are different. Typically, RTDs use pure metals, while thermistors typically use ceramics or polymers. RTDs and thermistors have differing temperature responses. RTDs are effective across wider temperature ranges, whereas thermistors offer more precision within a narrower temperature range, typically ranging from -90 °C to 130 °C [11].

I.4.1.2 Characteristics of a PTC and NTC thermistance:

The essential parameters of a thermistor are:

- The value of his resistance.
- The thermal sensitivity, often known as the temperature coefficient.
- a stability (provided by the manufacturer)

Benefits: rapid response time, cost-effective.

Drawbacks: The series exhibits a non-linear law of characteristics, making it less predictable. It is also susceptible to auto-heating and changes in connecting resistors.

I.1.4.3 Schematic representation

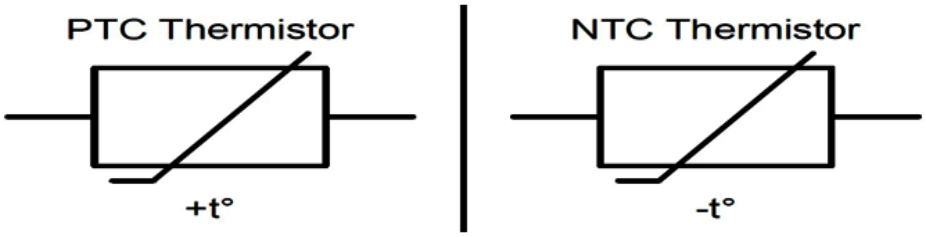

Figure I.2 : Représentation schématique [11].

This type of temperature sensor displays a change that is precise, predictable, and large in the alteration of different temperatures. With a change this large, it means that the reflection of the temperatures occurs rapidly and accurately [6].

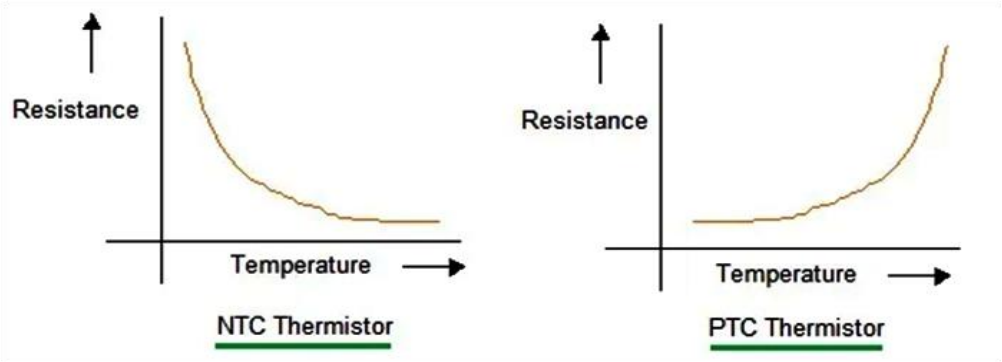

Figure I.3: Curves characterizing PTC and NTC [6].

I.4.2 The Resistance Temperature Detector (RTD)

This is known as the resistance thermometer and uses the resistance of the RTD element with temperature to measure the temperature. Different types of materials can be used to make the metal.

-

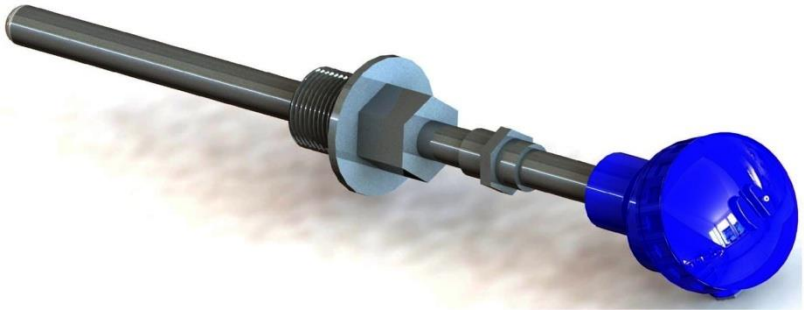

Figure I.3: Resistance Temperature Detector

The materials include nickel, platinum, and copper. However, platinum is the most accurate and therefore the most expensive [4].

I.4.3 Thermocouple sensors

Known as a "thermoelectrical thermometer," a thermocouple is an electrical instrument that consists of two distinct electrical conductors that form an electrical junction. The Seebeck effect causes a thermocouple to generate a voltage that varies with temperature, allowing us to determine the temperature. Thermocouples serve as commonly utilized temperature sensors [12]. Thermocouple sensors have two wires made of different metals connected at two points. The voltage between the two wires reflects the change in temperature. Although their accuracy may be slightly reduced to a degree that is lower than an RTD, they have a temperature range between -200 °C to 1750 °C and are generally more cost-effective.

Commercial thermocouples are cost-effective, easily replaceable, come with standard connectors, and have the capability to measure a broad spectrum of temperatures. Unlike most other temperature measurement systems, thermocouples are self-powered and do not need any external source of stimulation. The precision of thermocouples is primarily limited, as obtaining system errors of less than one degree Celsius (°C) can be challenging [13].

Thermocouples have many applications in both scientific and industrial fields. Temperature measurement is used in a variety of industrial processes, including kilns, gas turbine exhaust, diesel engines, and more. Residential, commercial, and industrial settings utilize thermocouples as temperature sensors in thermostats and as flame sensors in safety mechanisms for gas-powered equipment.

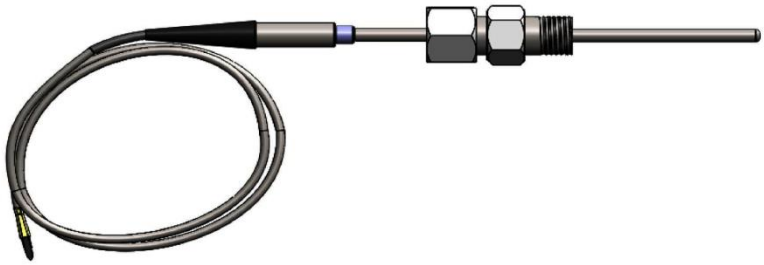

Figure I.4: Thermocouple sensor

I.4.3.1 Principle of operation:

The German physicist Thomas Johann Seebeck noted in 1821 that heat caused one of the metal connections in a circuit consisting of two distinct metals to shift in position. At the time, Seebeck referred to this outcome as thermomagnetism. He later determined that thermoelectric current was the cause of the magnetic field he observed. The voltage created at a single junction of two dissimilar wires is of practical interest, as it may be utilized to monitor temperature at both extremely high and low levels. The specific types of wire employed determine the magnitude of the voltage. Typically, the voltage is within the microvolt range, and it is important to provide a reliable measurement. Despite the minimal flow of electric current, a solitary thermocouple junction has the ability to generate power. Utilizing many thermocouples, such as a thermopile, is a widely used method for power generation [14].

The illustration depicts a typical arrangement for using a thermocouple. In summary, three inputs determine the required temperature T_{sense}: the thermocouple's characteristic function *E(T)*, the measured voltage *V*, and the temperature T_{ref} of the reference junctions. Solving the equation $E\ (T_{sense}) = V + E\ (T_{ref})$ yields the value of T_{sense}. These specificities, integrated into a single product that includes the reference junction block (with T_{ref} thermometer), voltmeter, and equation solver, often go unnoticed by the user [15].

I.4.3.2 Seebeck effect

In 1821, German physicist Thomas Seebeck made the discovery that when two metal wires made of different materials (Seebeck specifically used copper and bismuth) are connected at both ends to create a loop, an electric potential difference, or voltage, is generated in the circuit if the two junctions are maintained at different temperatures. A thermocouple is a pair of metals that make up a circuit. The phenomenon occurs when heat energy is transformed into electrical energy [16].

- The Seebeck effect is the occurrence of a voltage differential between two dissimilar electrical conductors or semiconductors when there is a temperature variation between them.
- Heat applies to either a conductor or a semiconductor, causing the heated electrons to move towards the cooler one. Direct current (DC) flows through the circuit when an electrical circuit links the pair.
- The Seebeck effect generates low voltages, typically only a few microvolts (millionths of a volt) per kelvin of temperature differential at the junction.
- If the temperature gradient is sufficiently significant, certain Seebeck-effect devices have the capability to generate a few millivolts (thousandths of a volt). Multiple devices can be linked in series to amplify the output voltage or in parallel to enhance the maximum deliverable current.
- When there is a significant temperature disparity across the junctions, significant arrays of Seebeck-effect devices can generate valuable, compact electrical power.

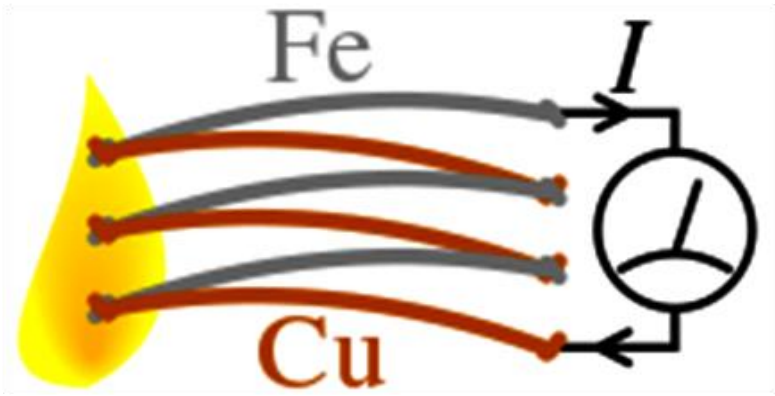

Figure I.5: Seebeck effect in a thermopile made from iron and copper wires [16].

I.4.3.3 Thermocouple types

Specific combinations of alloys have gained popularity and are now widely accepted as the norm in several industries. Factors such as cost, availability, convenience, melting point, chemical characteristics, stability, and output influence the selection of the combination. Different varieties are best suited for different applications. They typically choose them based on the required temperature range and desired sensitivity.

Thermocouples that have low sensitivities, such as the B, R, and S types, also have poorer resolutions. Additional factors to consider when selecting a thermocouple material include its chemical inertness and magnetic properties [12]. The following are the standard

types of thermocouples, given in order of the positive electrode (assuming the temperature sensed is higher than the reference temperature), followed by the negative electrode.

Thermocouples are separated into types, with each type being suitable for specific temperature conditions. The various classes of thermocouples are constructed to meet the needs of a specific application [16,17,18].

I.4.3.3.1 Type E Thermocouples - Type E thermocouples have chromel, a nickel and chromium alloy, and constantan with a temperature range of 0°C to 870°C and excellent EMF versus temperature values. They can be used in sub-zero temperatures and are used in inert environments but must be protected in sulfurous environments [12,17].

I.4.3.3.2Type J Thermocouples - Type J thermocouples, like type K thermocouples, are a general-purpose thermocouple made of iron and constantan, with the iron leg being positive and the constantan leg being negative. They can be used exposed or unexposed with a protective tube being recommended. Type J thermocouples are used in vacuum, inert, and reducing environments. As with type K thermocouples, type J thermocouples have to be carefully calibrated and do not react well to noise .Type J has a more restricted range (−40 °C to +1200 °C) than type K but higher sensitivity of about 50 μV/°C .

I.4.3.3.3 Type K Thermocouples - Type K thermocouples are made of Chromel–Alumel with small percentages of manganese and silicone. They are a general purpose thermocouple with a temperature range of -200°C up to 1350°C. Type K thermocouples need to be carefully calibrated and have small output signals. They are used in an assortment of environments including water, mild chemicals, gasses, and dry conditions. Common industries that use type K thermocouples are hospitals and food preparation. Regardless of their wide temperature range, type K thermocouples are mostly used for temperatures over (540°C). The Type K thermocouple, is widely used for general purposes and has a sensitivity of around 41 μV/°C [12].

I.4.3.3.4 Type N Thermocouples - Type N thermocouples have nicrosil, a nickel chromium alloy, and nisil, a nickel, silicon, and magnesium alloy. Their temperature range is from 650°C to 1260°C. Type N thermocouples are resistant to green rot and hysteresis and are used in refineries and the petrochemical industry.

I.4.3.3.5 Type R Thermocouples - Type R thermocouples are made of platinum and rhodium and are usable for temperatures up to 1480°C. They have to be protected by a gas-tight ceramic tube and a secondary outer tube. Type R thermocouples have improved stability, an increased temperature range over Type S thermocouples, and are often used in place of Type

S thermocouples. Applications for Type R are heat treating, control sensors, semiconductor industry, glass manufacturing, and ferrous and non-ferrous metals.

I.4.3.3.6 Type S Thermocouple - Type S thermocouples are used in high temperature applications in the BioTech and Pharmaceutical industries. They are also used for low temperature applications due to their accuracy and stability. Type S thermocouples have a temperature range of 980°C to 1450°C.

I.4.3.3.7 Type T Thermocouples - Type T thermocouples are made of copper and constantan with a temperature range of -200°C to 370°C. They are used in inert atmospheres and are resistant to decomposition even if there is moisture present. Type T thermocouples are used in food production and cryogenics.

I.4.3.3.8 Type B Thermocouple - Type B thermocouples are used for high temperature applications and have the highest temperature limit of the thermocouples with exceptional accuracy and stability. Their alloy combinations are platinum (6% Rhodium) and platinum (30% rhodium) with a temperature range of 1370°C to 1700°C.

I.4.3.3.9 Type C Thermocouples - Type C thermocouples have tungsten and rhenium legs and are used with applications that require temperatures up to 2315°C. They are used in hydrogen, inert, or vacuum atmospheres to prevent failure from oxidation. Type C thermocouples have protective sheaths made of molybdenum, tantalum, and inconel with insulators of alumina, hafina, and magnesium oxide.

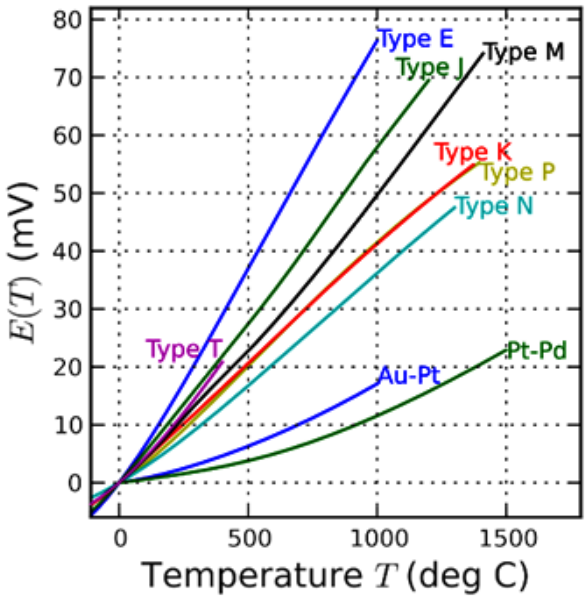

Figure I.6: Characteristic functions for thermocouples that reach intermediate temperatures [12].

I.4.3.4 Uses for Thermocouple Types:

- E: suitable for vacuum, inert, mildly oxidizing, or reducing conditions.
- J: used where there is limited oxygen.
- K: requires metal or ceramic protection.
- N: resists oxidation from sulfur.
- T: used in oxidizing or reducing environments.
- S, R, and B: must be protected with a form of tubing and used for high temperature applications.
- C (tungsten/rhenium): very common; requires protective sheathing and used for high temperature applications.
- A: a variant of type C and has limited use

I.4.4 Bi-Metal Thermometer

This type of thermometer consists of a connected gauge and stem. There is a spring attached to a rod on the tip of the sensor, leading up to the gauge's needle. Inside the stems, the spring sits sensing the end [3].

Figure I.7: Bi-Metal Thermometer.

When heat is applied on the sensing coil, there is a creation of movement in the coil, which causes the movement of the needle in the gauge – thus the temperature is displayed.

I.4.5 Digital Thermometer

This type of thermometer utilizes a probe such as a thermocouple or an RTD. The temperature is measured by the probe (sensing end) and displayed as a digital reading [12].

Figure I.8: Digital Thermometer.

I.4.6 Infrared Sensors

These types of temperature sensors detect temperatures from a distance by measuring the amount of thermal radiation that is being emitted by a heat source or object.

These temperature sensors find their application in high temperatures or hazardous environments, where a safe distance must be maintained away from a particular body [3].

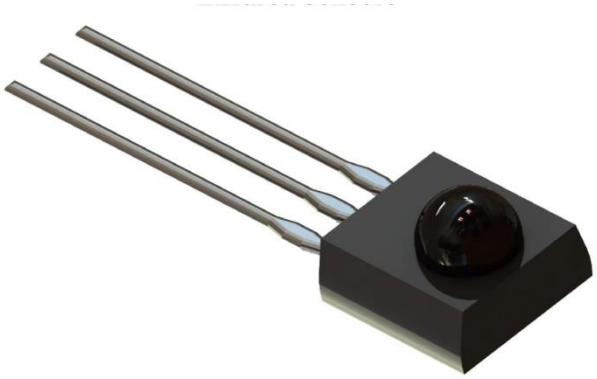

Figure I.9: Infrared Sensors.

I.4.7 Lm 35 sensors

I.4.7.1 Definition of lm 35

Texas Instruments manufactures the LM35, an analog temperature sensor. The electronics industry highly favors it due to its accuracy, low cost, user-friendliness, and dependability. In its most accurate version, the LM35 temperature sensor can measure temperatures between -55 °C and +150 °C. With the right installation, it can measure any temperature [14].

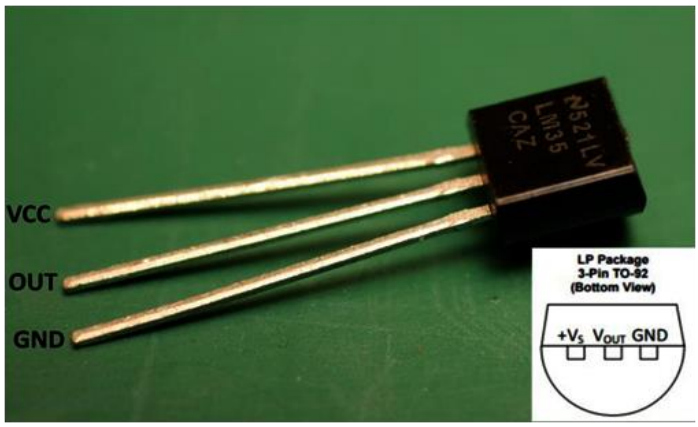

Figure I.10: LM35 Temperature Sensor

The sensor's analog output is proportional to temperature. Measure the tension at the sensor's output to determine the temperature. Each degree Celsius corresponds to a stress of +10 mV. Although Arduino cards cannot tolerate temperatures as high as -55 °C or +150 °C, the LM35 sensor can.

Exposing a "classic" Arduino card to severe temperatures will result in either complete failure or rapid deterioration. This statement applies to Arduino boards and other electronic circuits that adhere to "public" standards instead of "industrial" ones.

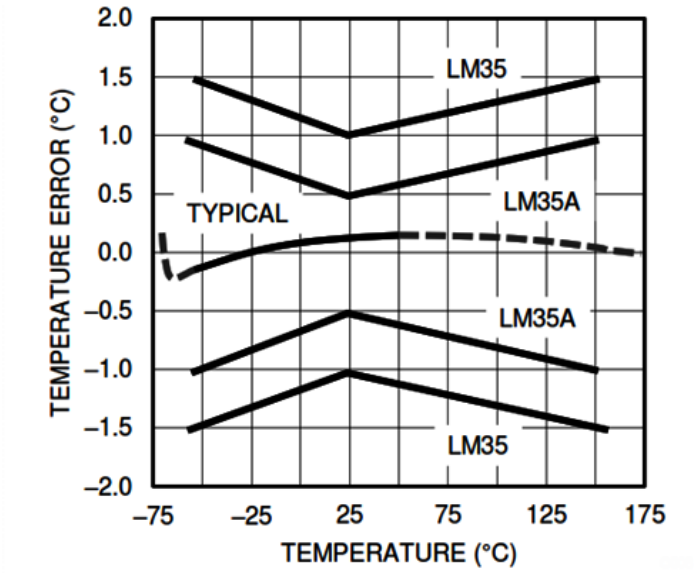

Figure I.11: Precision curves of different versions of LM35 [14].

For information, the classical temperature ranges in electronics are as follows:

- grand public : 0°C ~ 70°C
- Industry: -40°C ~ 85°
- Military: -55°C ~ 125°C

The LM35 sensor, which contributes to its widespread popularity, is its pre-calibrated output directly from the factory. During the manufacturing process, LM35 sensors undergo calibration in degrees Celsius. This suggests that removing the sensor from its packaging already calibrates it, eliminating the need for additional steps to adjust its settings.

An outstanding advantage of the LM35 sensor, which contributes to its widespread popularity, is its pre-factory calibration. During the manufacturing process, LM35 sensors undergo calibration in degrees Celsius. This implies that removing the sensor from its packaging already calibrates it, eliminating the need for additional steps to adjust it.

An outstanding advantage of the LM35 sensor, which contributes to its widespread popularity, is its pre-factory calibration. During the manufacturing process, LM35 sensors undergo calibration in degrees Celsius. This implies that removing the sensor from its packaging already calibrates it, eliminating the need for additional steps to adjust it.

For example, the LM35 sensor has exceptional linearity, with an error rate of less than 1 degree Celsius across the whole temperature range of -55 degrees Celsius to +150 degrees Celsius. The process of translating a measurement to temperature is a cross-produced beast because each degree Celsius equates to 10 millivolts, which is equivalent to 0.01 volts, and the output of the sensor is (nearly) perfectly linear.

In conclusion, you can use the LM35 sensor in practically any digital or analogue mounting configuration because it is compatible with any power voltage between 4 and 30 volts.

I.4.7.2 Brochure of the lm 35

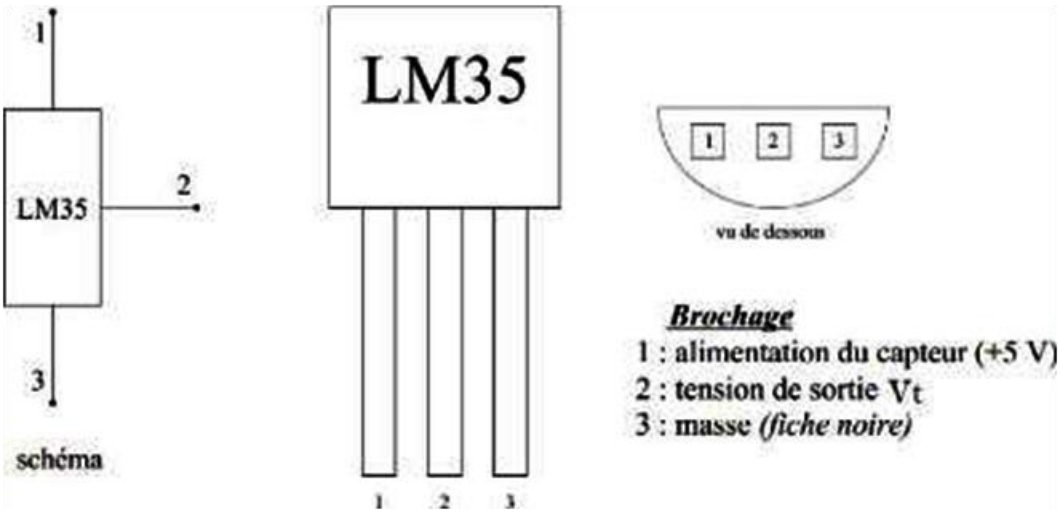

Figure I.12: Brochure of the lm35 [14].

I.4.7.3 Calibration of lm35

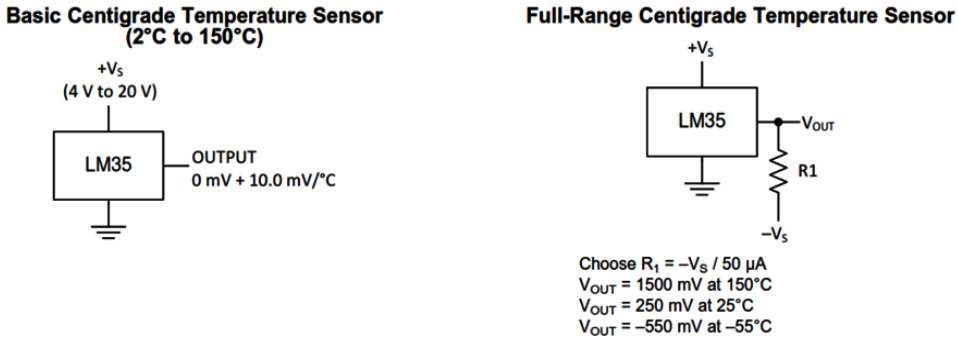

Figure I.13: Calibration of lm35.

I.4.8 Semiconductor-Based Sensors

Semiconductor-based temperature sensors work with dual integrated circuits. They consist of two diodes that are similar, with voltage and temperature-sensitive current characteristics for measuring the changes in temperature effectively.

Semiconductor-based sensors give a linear output; however, they are less accurate at 1 to 5°C. These types of temperature sensors have the slowest responsiveness (5 to 6 seconds) across the narrowest temperature range (-94°F to 302°F or -70°C to 150°C) [4].

Figure I.13: Semiconductor-Based Sensors.

I.5 Applications of Temperature Sensors

The function of temperature sensors is to measure temperature in many various applications and industries. Temperature sensors are all around us, present in both industrial settings and everyday life. The following are examples of the applications of temperature sensors [14]:

I.5.1 Industrial Applications

Temperature sensors are utilized to monitor various environments and machinery, power plants, and manufacturing. Temperature sensors are used to measure water temperatures in reservoirs and boreholes. They can also be used to interpret temperature-related stress and changes in volume in dams. Temperature sensors are also utilized in the study of the temperature effect on other installed instruments.

I.5.2 Scientific and Laboratory applications

Temperature sensors are utilized in science and biotech monitoring.

I.5.3 Medical Applications

Temperature sensors are utilized in the monitoring of patients, in medical devices, in thermodilution, in humidifiers, gas analysis, cardiac catheters, ventilator flow tubes, and dialysis fluid temperature.

I.5.3 Uses in Motorsports

Temperature sensors are used for measuring inlet air temperature, exhaust gas, engine temperature, and oil temperature.

I.5.4 Domestic Appliances

Temperature sensors are used in kitchen appliances (ovens, kettles, etc.) and also in white goods.

I.6 Benefits of Temperature Sensors

The benefits of temperature sensors include:

- Temperature sensors are precise, extremely reliable, and have a low cost.
- Temperature sensors are suitable for both embedded and surface applications.
- They provide low thermal mass resulting in a fast response time.
- The vibrating wire temperature sensor is completely interchangeable; all sensors can be read by one indicator.
- Temperature sensors are available with indicators for direct display of temperature.
- Temperature probes exhibit excellent hysteresis and linearity.
- The technology of the vibrating wire ensures long term stability, easy and quick readout.
- Temperature sensors perfectly suit remote scanning, reading, and data logging.

I.7 Conclusion

A temperature sensor can also be defined as a simple instrument that measures the degree of coldness or hotness and then converts it into a readable unit. There are different

types of temperature sensors, including thermocouples, thermistors, RTDs, etc. Each temperature sensor has its own unique characteristics, which makes it suitable in specific types of applications. However it is important to consider the temperature range, accuracy, size, and stability of a temperature sensor for optimal performance within a specific application.

Thermistors and the LM35 temperature sensor are notable for their outstanding durability, accuracy, and ability to seamlessly integrate with other systems. Thermocouples and the LM35 are essential for accurate temperature monitoring and control since they demonstrate substantial changes in resistance in response to temperature fluctuations. The LM35 is particularly valuable due to its precise and linear output.

Chapter II

Properties of The programable device Arduino and the Firmata communication protocol

Chapter II: The programable device Arduino and the Firmata communication protocol

II.1 Introduction

Arduino is a platform for electronics that is open-source, meaning its design and code are freely available. It integrates both hardware and software components, making it easy for users to develop and control electrical devices. The system comprises a microcontroller board and an integrated development environment (IDE) used for authoring and transferring code to the board. Arduino boards have the ability to communicate with a variety of sensors, actuators, and other devices, which makes them well-suited for a wide array of uses. These uses can range from basic educational tools to intricate prototypes and industrial solutions. The platform's user-friendly interface, adaptability, and robust community backing have established it as the preferred option for enthusiasts, educators, and experts.

Firmata is a communication protocol that facilitates serial communication between a microcontroller and a host computer. It enables the computer's software to manipulate the microcontroller's I/O pins and other functionalities, facilitating interaction with hardware using programming languages like Python, JavaScript, and Processing. Firmata encompasses a wide range of capabilities, including digital and analog input and output, pulse width modulation and servo control, and connection with I2C and SPI devices. The abstraction layer streamlines hardware control by managing the intricate technical aspects, enabling developers to concentrate on the application's logic. Firmata Express is a streamlined iteration of the Firmata protocol, specifically designed to enhance usability while preserving the essential functionalities required for the majority of applications. It simplifies the system by restricting the number of instructions to the most frequently used ones and prioritizing fundamental tasks like digital and analog input/output activities. This more efficient technique enhances performance by reducing unnecessary tasks and making the protocol more user-friendly for novices and instructors. Firmata Express is especially suitable for educational settings and quick prototyping, where simplicity and efficiency are of utmost importance.

In this chapter we show thatArduino and the Firmata protocols provide effortless communication between software and hardware, permitting users to construct and manage a diverse array of electrical projects with simplicity and adaptability.

II.2 Arduino Module Definition

The Arduino module is an open-source hardware printed circuit board (PCB) that serves as a control platform. Under an open license, the PCB design plans are freely available. However, some components of the PCB, such as the microcontroller and complementary components, are not subject to the same open licensing. A programmable microcontroller has

the capability to evaluate and generate electrical signals in a manner that allows it to carry out a wide range of operations [19].

Applications for Arduino include industrial and on-board electrical engineering, modeling, home engineering, contemporary art, robot piloting, engine control, light game creation, computer communication, and mobile device control. A voltage regulator operating at +5 V and a quartz oscillator with a frequency of 16 MHz (or a ceramic resonator in some variants) are features of every Arduino module. We use the IDEArduino software to program this map [20]

II.2.1 Arduino card ranges

Presently, there are around 20 iterations of the Arduino module. To provide a clear assessment of this scientific and academic product, we will highlight a few examples:

- The Arduino NG is equipped with a USB interface for programming and using an ATmega8 microcontroller.
- The Arduino, equipped with a USB interface, facilitates programming and the use of an ATmega8 microcontroller.
- The Arduino Mini is a compact iteration of the Arduino that utilizes an ATmega168 microprocessor.
- The Arduino Nano is a compact programmable board that utilizes the USB interface. The ATmega168 (or ATmega328 for a more recent iteration) microcontroller powers this particular version.
- The LilyPad Arduino is a simplified design specifically created for wearable applications, using an ATmega168 microcontroller.
- For programming and using an ATmega168 microcontroller, the Arduino NG+ has a USB interface.
- A Bluetooth interface on the Arduino Bluetooth enables programming using an ATmega168 microcontroller.
- The Arduino Diecimila uses an ATmega168 microprocessor and has a USB port.
- The Arduino Duemilanove uses either USB or DC power to power its ATmega168 (ATmega328 for a later version) microcontroller.
- The Arduino Mega utilizes an ATmega1280 microprocessor to provide more input/output capabilities and memory.
- The Arduino UNO, uses microcontroller ATmega328.

- The Arduino Mega2560 boasts an ATmega2560 microprocessor and a total memory capacity of 256 kilobytes. Additionally, it includes the latest ATmega8U2 (or ATmega16U2 in the case of version 3) USB chipset.
- The Arduino Leonardo has an ATmega32U4 microcontroller that removes the need for a USB connection and may function as a keyboard.
- The Arduino Esplora is a device that resembles a visual gaming controller and has a handle as well as built-in sensors for detecting noise, light, temperature, and acceleration [21].

From these options, we selected an Arduino UNO board (basic card). This card aims to speed up the implementation of the previously mentioned directive, with more details to follow. In its development environment, Arduino provides a programming interface based on open-source technologies. The environment has already transformed the program into hexadecimal code, which we inject into the microcontroller's memory using a straightforward method over the USB connection. We also provide function libraries that facilitate input-output exploitation. This card utilizes an ATmega 328 microcontroller along with other components. The Arduino board has a non-volatile memory capacity of 1 kilobyte. The device boasts 14 digital inputs and outputs, with six of them capable of serving as PWM outputs. It also features six analog outputs, a 16 MHz crystal, a USB connection, a zero-drop button, and a power jack outlet [22]. The card is illustrated in the figure below.

Figure II.1: Arduino UNO Card.

II.2.2 Arduino UNO benefits

For programming electronic devices, a variety of electronic cards with microcontroller-based platforms are available. Each of these technologies facilitates intricate programming complexities and seamlessly integrates them into a user-friendly interface. Likewise, the Arduino system streamlines the process of working with microcontrollers while providing several benefits to those who are interested. These advantages include:

- The price: Arduino boards are quite affordable in comparison to other platforms. You can manually construct the most affordable iterations of the Arduino module, with pre-made Arduino boards priced below 2500 dinars.
- Multi Platform: Windows and Linux operating systems can run the Java-written Arduino software. The majority of microcontroller systems have restricted compatibility with the Windows operating system.
- The Arduino IDE programming environment is easy to use for novices and sufficiently versatile for more experienced users to get benefits from it as well.
- An open source license allows expert programmers to add to the Arduino software and language. The Arduino module programming software functions as a code editor and compiler and is a cross-platform Java application that can run on any operating system. It also has the ability to transmit programs via serial linking. Depending on the module, it can transmit programs via RS232, Bluetooth, or USB.
- The Arduino cards are based on the ATMEGA8, ATMEGA168, and ATMEGA 328 microcontrollers from Atmel. The module layouts are available for free under a Creative Commons license, and skilled circuit designers are welcome to modify and enhance the original Arducino cards. Even relatively unskilled users can make the trial board version of the Arduino card; its goal is to help them learn how it functions in order to save money [22,23].

II.2.3 The constitution of the Arduino UNO card

An ATMEL AVR microprocessor and other parts that make programming and circuit interface easier are usually the foundation of an Arduino module. Every module features a 16 MHz quartz oscillator (or, in some variants, a ceramic resonator) and at least one 5V linear regulator. A separate programmer is not required since the microcontroller comes preprogrammed with a bootloader.

II.2.3.1 Material Part

Generally speaking, one or more programmable circuits serve as the foundation for every electrical module having a programming interface.

II.2.3.2 The Microcontroller ATMega328

An integrated circuit known as an ATMega328 microcontroller combines several intricate components onto a compact chip.

These days, a lot of bulky parts, including transistors, resistors, and capacitors, may be welded together to create a little black plastic casing with many pins that can be programmed in the C programming language. An ATmega 328 microcontroller from the Arduino map is seen in Figure II.2 [23].

Figure II.2: Microcontroller ATMega328

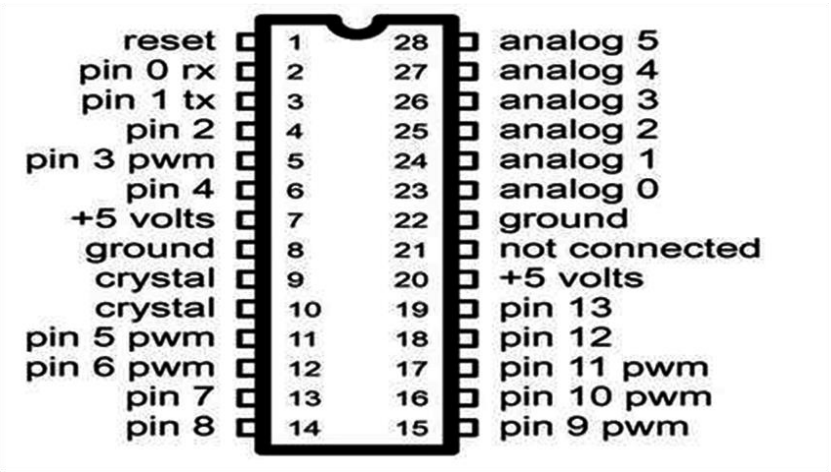

Figure II.3: Atmega328 brochure.

Each of the components that make up the ATMega328 microcontroller has a specific purpose. In actuality, it is composed of the same components as a computer's motherboard. All things considered, this programmable circuit's internal design is mostly composed of:

- Flash memory: This is the type of memory that will hold the executable software. This memory program, which has a 32-bit Ko bootloader included, is rewriteable and deleteable.

- RAM: also known as "live" memory, is the type of memory that holds the program's variables. The reason it is referred to as "volatile" is that it is destroyed if the microcontroller's power source is shut off. It has a 2 KB capacity.
- EEPROM: This is the microcontroller's hard drive. Even if the map needs to be stopped, it captures data that must endure over time. When the microcontroller is reprogrammed or turned off, this memory is not removed [24].

II.2.3.3 The power sources of the card

Two categories of power sources (Input Output) can be distinguished in the manner described below:

- V_{in} is the positive input voltage (as opposed to the 5V from the USB connection or any other regulated 5V source) that the Arduino board receives when it is connected to an external voltage source. This pin can be used to power the card or to access the power supply voltage if the power jack is the source of power..
- 5V is the regulated voltage that powers the Arduino card's microcontroller and other parts. To put this into context, digital electronic circuits need a perfectly stable power voltage known as "regulated tension," which is provided by an Arduino card-integrated regulator component. Therefore, the regulated 5V provided by this pin may originate from any regulated power source, the USB connection (which supplies the regulated 5V), or the V_{IN} power voltage through the card regulator.
- 3V3. The card's FTDI integrated circuit, which converts the signal between the USB port on your computer and the serial port on the ATmega, provides a 3.3V power supply. This is interesting since some external devices need this voltage instead of 5V. This pin has a maximum intensity of 50 mA [22,25].

II.2.3.4 Inputs & Outputs

Using the Arduino programming languages pinMode(), digitalWrite(), and digitalRead(), this card's 14 digital pins (numbered 0 to 13) can be utilized as either digital inputs or outputs. These pins are 5V compatible. A maximum of 40 mA of intensity can be supplied or received by each pin, and each pin has an inbuilt 20–50 kOhm resistance set as the default unconnected state. The digital Write instruction (broche, HIGH) is used to turn on this intrinsic resistance on an input pin. In addition, some pins have specialized functions:

- External interruptions: Pins 2 and 3. These pins have the option of being set up to interrupt on a low value, rising or falling front, or value change. Modulated pulse

width, or PWM, is present in pins 3, 5, 6, 9, 10, and 11. Utilizing an analog Write instruction, it generates a 1-bit PWM impulse ().

- SPI stands for Peripheral Series Interface, and its pins are SS, MOSI, MISO, and SCK. The SPI communications library is required in order to use these pins for SPI (Series Peripheral Interface) communication. The ICSP connector, which is mechanically compatible with Mega cards, is also connected to SPI pins.
- I2C: SDA and SCL pins 4 and 5. Support TWI (Two-Wire Interface—Interface "2 Wire"), which is provided through the Wire/I2C library, or I2C protocol communications (ou TWI—Two-Wire Interface—Interface "2 fils").
- LED: Pine 13. The card has an LED that is attached to pin 13. The LED turns on when the pin is at the UP level and turns off when the pin is at the base level.

The Arduino language's very helpful analogRead() function can measure each of the six analog inputs on the UNO card at a 10-bit resolution (1024 levels, or 0 to 1023). The numbers 0 through 5 represent the inputs. These pins measure in the range of 0V (0 value) to 5V (1023 value) by default. However, by utilizing the AREF pins and the Arduino language's analogReference() instruction, it is possible to modify the upper reference of the measurement range.

A fuse built inside the Arduino UNO card guards against an intensity overload on the computer's USB port (which is typically limited to 500 mA). The card fuse adds an extra degree of security, even though the majority of PCs already have internal protection. The card fuse will immediately disconnect until the short circuit or overload is terminated if more than 500 mA is applied to the USB port [22].

II.2.3.5 Communications ports

There are many ways to communicate with the outside world using the Arduino UNO card.

The digital pins 0 (RX) and 1 (TX) of the Atmega328 enable series UART TTL (5V) communication [26].

TTL-level serial data is transmitted using (TX) and received using (RX). These pins are linked to the matching pins of the ATmega328 integrated circuit, which is configured to function as a USB-to-series card converter. This component serves as an interface between the computer's USB port and the TTL levels.

The Arduino serial connection is a very practical way for a PC and Arduino software to communicate as a virtual communication port. However, using the Arduino serial

connection requires a USB cable, which can be avoided by using one of the following methods to utilize the Arduino wirelessly:

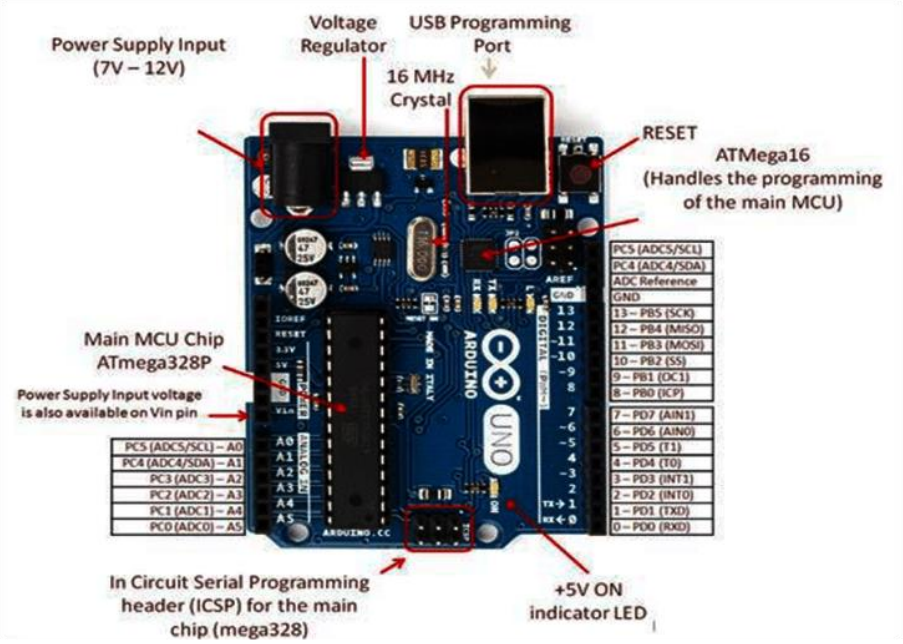

Figure II.4: Arduino UNO Pin Connections.

II.3 Program Part

An acquisition card of this type, whose design revolves around a microcontroller, needs to have a programming interface, which our card does. You can download the free open-source Arduino programming environment for Windows, Linux, and Mac OS X.

II.3.1 The programming environment

A code editor (in a language similar to C) is provided by the Arduino card's programming software. After the software is typed or edited on the keyboard, the USB connection will transfer and store it on the card. In addition to providing power to the card, the USB cable transmits data for the Arduino IDE application [27].

II.3.1.1 General structure of the programme (Arduino IDE)

Any Arduino operating system that supports C programming can run an easy, simple interface, much like any other programming language.

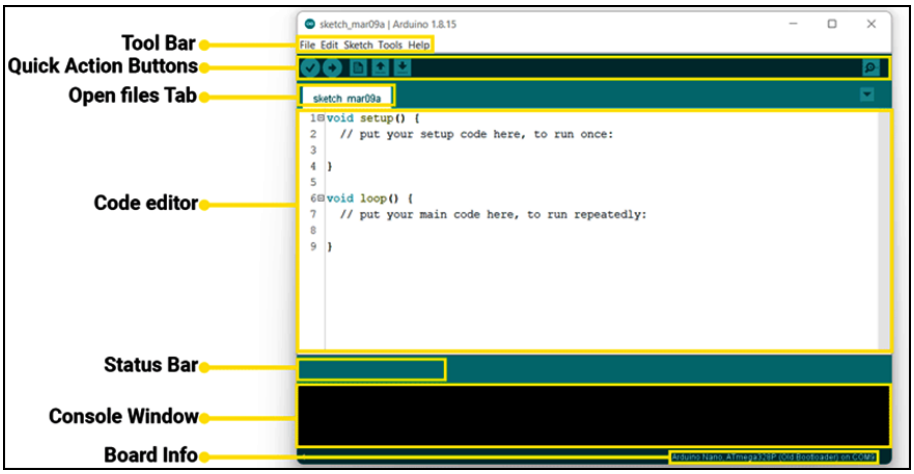

Figure II.5: Arduino IDE Interface.

II.3.1.2 Program Injection

For example of this figure, it needs to choose the Arduino UNO card type and USB port number (COM 3) before sending a program to the card.

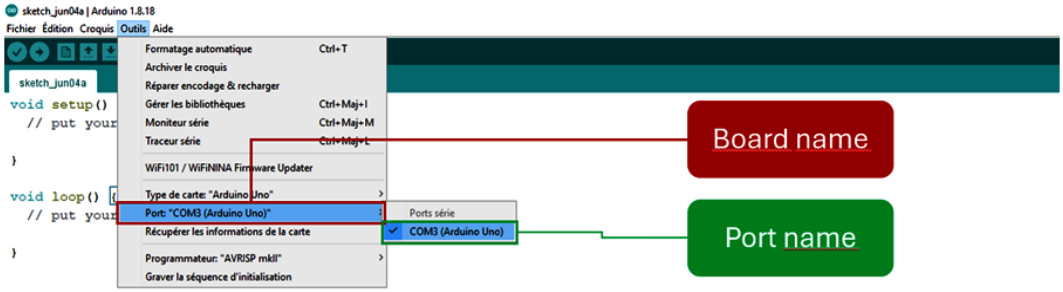

Figure II.6: Select board and port in Arduino IDE.

II.3.1.3 Program Description

An arduino program is a set of basic instructions in text form (line by line). The card reads and then performs the instructions one after the other in the order provided by the code lines.

II.3.1.4 Comments

Comments are, in computer programming, elements of the source code ignored by the compiler or interpreter, as they are not meant to impact the execution of the program.

//temperature control program with lm35

II.3.1.5 Definition of variables

The analog input of the card, designated as, for instance, A0, will be used for our assembly; so this variable needs to be defined and called sensor A0.

configuration of void inputs and outputs setup ()

When a sketch launches, the setup() function is run. Use it to begin using libraries, set pin modes, initialize variables, etc. After the Arduino board is powered on or reset, the setup() code will only execute once [22].

II.3.1.6 Programming void loop interactions

The setup() function sets the starting values and initializes your program. The loop() function then does exactly what its name implies, looping back and forth until your program responds and changes. To actively control the Arduino board, use it.

II.3.2 Upload a program in Arduino IDE

The Arduino card can be programmed with code by performing a straightforward chain manipulation using the USB interface.

1. Using the Arduino IDE software, create a new program or open an existing one.
2. We are using the Arduino software (compilation) to verify this program.
3. The program is changed if errors are reported.
4. The software is loaded into the card.
5. The program runs automatically after a few seconds.
6. The card is powered either by the USB port or by a standalone power supply (pile 9 volts par exemple).

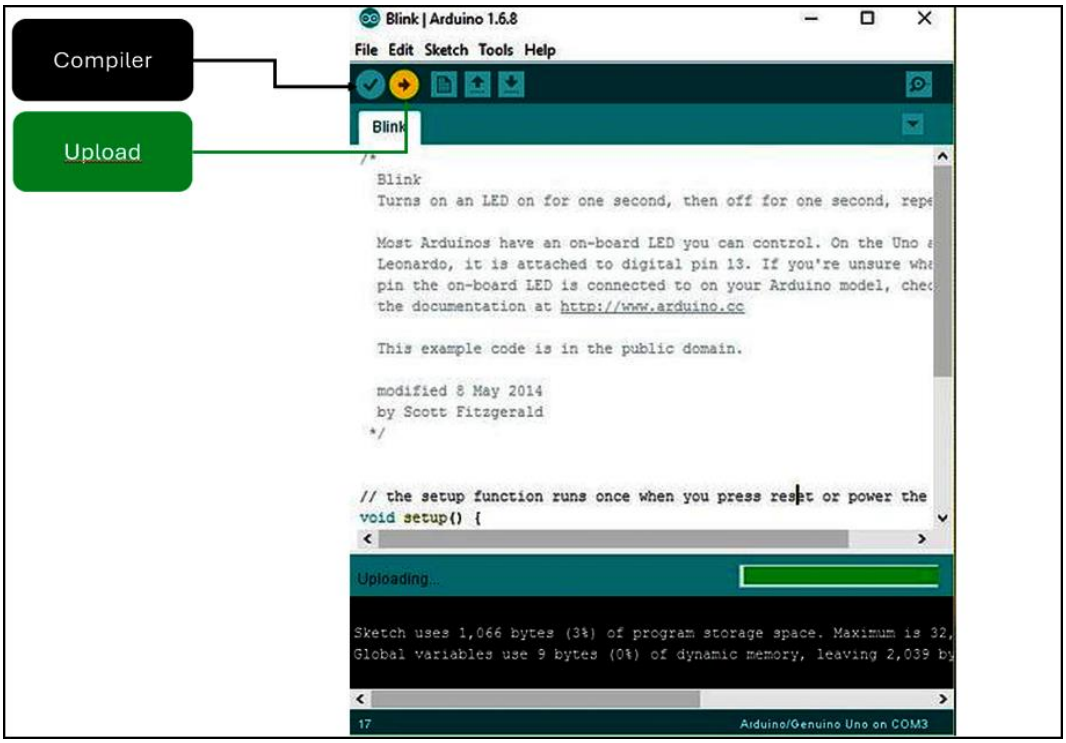

Figure II.7: Upload a sketch in Arduino IDE.

II.3.3 Arduino Card Accessories

The Arduino card is usually combined with accessories that simplify achievements.

II.3.4 Communication

The manufacturer has suggested that such a card should be equipped with several communication ports; some types can be clarified at the moment.

II.3.5 The Arduino Bluetooth module

Popular Arduino platform, the Arduino Bluetooth Microcontroller Module, has a low power consumption, modest range (approximately ten meters), low flow rate, low cost, and a non-overpowering Bluetooth serial connection instead of a USB connection.

Figure II.8: Module Type Bluetooth.

II.3.6 Arduino Wifi Shield Module

An Arduino card can be linked to a wireless WiFi Internet network using the Arduino Shield WiFi module.

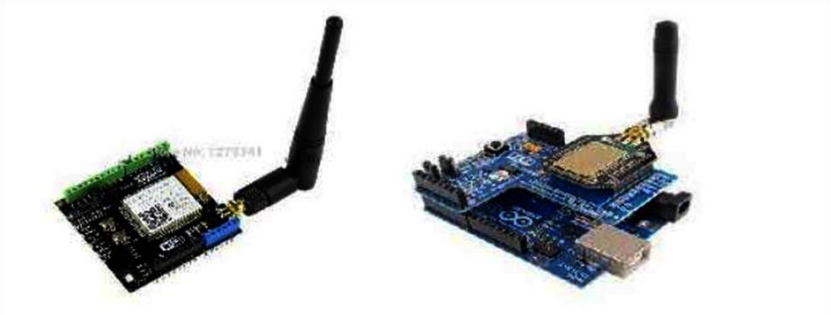

Figure II.9: Arduino Wifi Shield Module.

II.4 Python scripts

Python is an object oriented scripting language that was released publicly in 1991. It was developed by Guido van Rossum of the National Research Institute for Mathematics and Computer Science in Amsterdam. Python has rapidly become one of the world's most popular programming languages. It's now particularly popular for educational and scientific

computing, and it recently surpassed the programming language R as the most popular datascience programming language [28].

Here are some reasons why Python is popular and everyone should consider learning it:

- It’s open source, free and widely available with a massive opensource community.
- It’s easier to learn than languages like C, C++, C# and Java, enabling novices and professional developers to get up to speed quickly.
- It’s easier to read than many other popular programming languages.
- It’s popular in artificial intelligence, which is enjoying explosive growth, in part because of its special relationship with data science.
- It enhances developer productivity with extensive standard libraries and thirdparty opensource libraries, so programmers can write code faster and perform complex tasks with minimal code.

In interactive mode, the lines of instructions are no longer accessible once executed. But it is of course possible to write and keep a program (a script), using an editor, to be able to execute it at will or to modify it later. There are many Python script editors that also include an interpreter to run the programs. This is called an IDE or Development Environment. It is a complete programming environment that comes in the form of an application. It generally consists of a code editor, an interpreter, a debugger... We can cite Pycharm, Spider... But for an introduction to programming in Python, the use of the IDLE interpreter which allows also script editor is quite adaptable.

II.5 Arduino-Python communication via the "Firmata" communication protocol

We learned that Python software may communicate with an Arduino through serial linking. But depending on the electronic circuits, the sensors you use, and the actions you want to take, you'll need to download an appropriate program into the Arduino's memory and frequently alter the related Python program to send or receive data. In order to solve this issue, "Firmata," a communication protocol, is built around two programs:

- a computer software called "commandator" written in Python that can communicate with the Arduino and receive data over the USB port.
- a microcontroller "driver" program that, as the name implies, will regulate the linked equipment in accordance with commands received.

The "driver" code needs to be loaded into the Arduino's memory using the "IDE ARDUINO" software before the Python "commandator" program can be launched [29.

But only the Python program will be changed in accordance with your desired actions once the "driver" program has been downloaded. In fact, reading and writing on Arduino inputs and outputs is possible straight from the Python application when using the "Firmata" communication protocol, which facilitates the exploitation of sensor data [29].

Two distinct communication protocols, "Firmata Standard" and "Firmata Express," will be utilized with the Arduino, depending on the circuits studied or the sensors used.

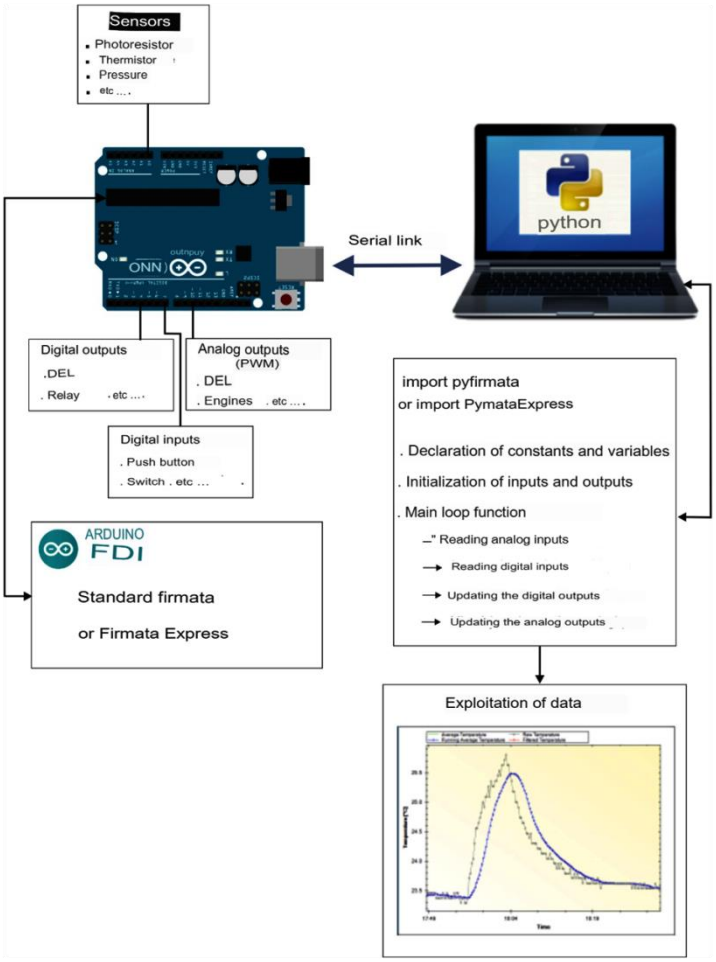

Figure II.10: Firmata communication protocol.

II.5.1 The "Firmata Standard" communication protocol

II.5.1.1 Uploading the "Firmata Standard" code into the Arduino memory

- Connect the Arduino via a USB port,
- To load the "Firmata standard" bookshop on the Arduino, first run the "IDE Arduino" program and choose:

File > Examples > Firmata > Standard Firmata,

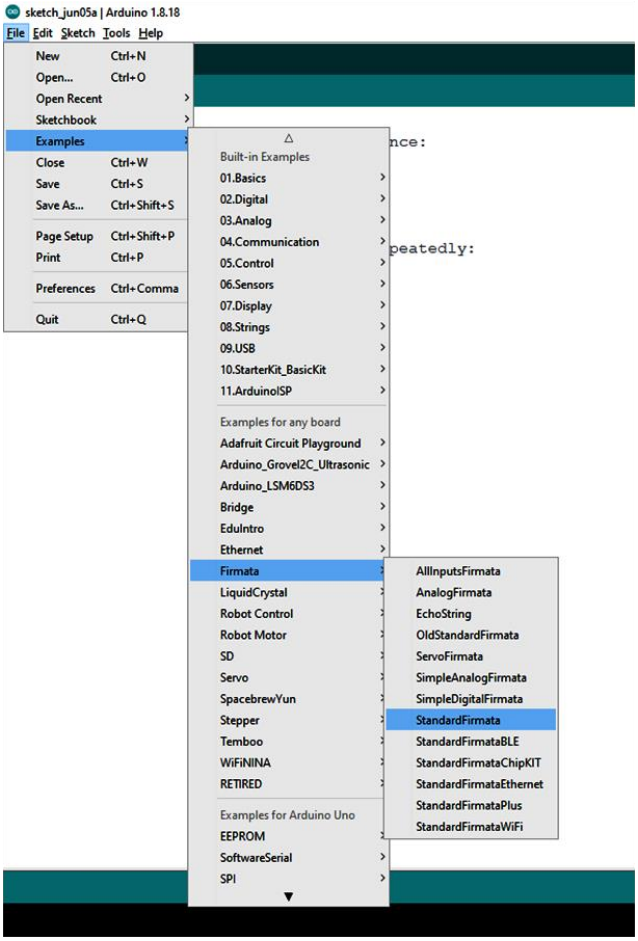

Figure II.11: Uploading the Firmata Standard code.

- then click on "Upload".

II.5.1.2 Installing the PyFirmata library in Python:

Python requires the "Py Firmata" module in order to execute a Python program that interfaces with the Arduino using the "Firmata standard" communication protocol.

Use the following command line to install this using "pip": “pip install pyfirmata”.

Using the following instructions, import the module in order to use the "pyfirmata" library in a Python program: Import Pyfirmata.

By indicating the COM port to which the Arduino is attached, the "Arduino" method of the "pyfirmata" module can be used to connect the microcontroller via the serial port:

board = pyfirmata.Arduino(Port COM)

Analog or digital Arduino inputs and outputs can be queried or modified after the connection has been made.

II.5.1.3 Digital Input Management

A digital pin (such as pin No. 5) must first be declared in advance using the following command in order to be read in its logical state: pin5 = board.get_pin('d:5:i')

The object formed when executing the "Arduino" method of the "pyfirmata" module is called "board," and the syntax is "d" for digital, "5" for pin number, "i" for input, and so on.

Then, we can use the following command to read the pin's logical state:

valeur = pin5.read()

At 5 V input, it returns "1"; at 0 V, it returns "0.".

Naturally, the read data travels via the serial link. However, in order to prevent an excessive number of measurements from overloading the serial connection between the Arduino and the computer, an iterator from pyFirmata must also be used:

it = pyfirmata.util.Iterator(board)

it.start()

The iterator must be launched after connecting to the Arduino.

II.5.1.4 Management of analog outputs

An Arduino pin must first be declared by creating an object called "pinPWM" with the following command in order to be used in analog output (PWM mode):

pinPWM = board.get_pin('d:numéro_de_pin:p')

The syntax is as follows: "d" stands for digital, "numéro_de_pin" for pin number (remember, pins 3, 5, 6, 9, 10, and 11 on the Arduino Uno allow PWM), "p" for PWM, and "board" for an object produced when the "Arduino" method of the "pyfirmata" module is called.

The PWM signal cyclic ratio between 0 (0%) and 1 (100%) can be adjusted by adjusting the voltage of the piston N°11 declared as an analog output:

board.digital[11].write(rapport cyclique)

- if cyclic ratio = 0, then the voltage is 0 V
- if cyclic ratio is = 1, then the tension is 5 V
- if cycle ratio was = 0.5, then it is 2.5 V

I.1.1.1 Management of analogue inputs

In order to obtain the voltage value of an analog input, such as pin A0, one must first declare it using the following instruction in the analog entry: pinA0 = board.get_pin('a:0:i')

The syntax is "a" for analog, "0" is the number of the pin A0, "i" for input and "board" is an object created when calling the "Arduino" method of the "pyfirmata" module.

Next, we read the pin voltage value using the "read()" function: pinA0.read()

While "analogRead()" in Arduino language returns an integer between 0 and 1023, the "read()" method for an analog input piston returns a decimal number between 0 and 1.

In order to prevent overloading the serial connection between the Arduino and the host computer, an iterator must also be used [29].

II.5.2 The communication protocol "Firmata Express"

The Firmata standard 2.5.8 library for Arduino has been enhanced to create the "Firmata Express" communications protocol. It was intended to be used in Python programs in conjunction with the Firmata client "pymata-express."

II.5.2.1 Downloading "Firmata Express" Library

- Connect the Arduino via a USB port,
- Open the software "IDE ARDUINO",
- Select "Sketch/Include a Library/Manage Libraries",

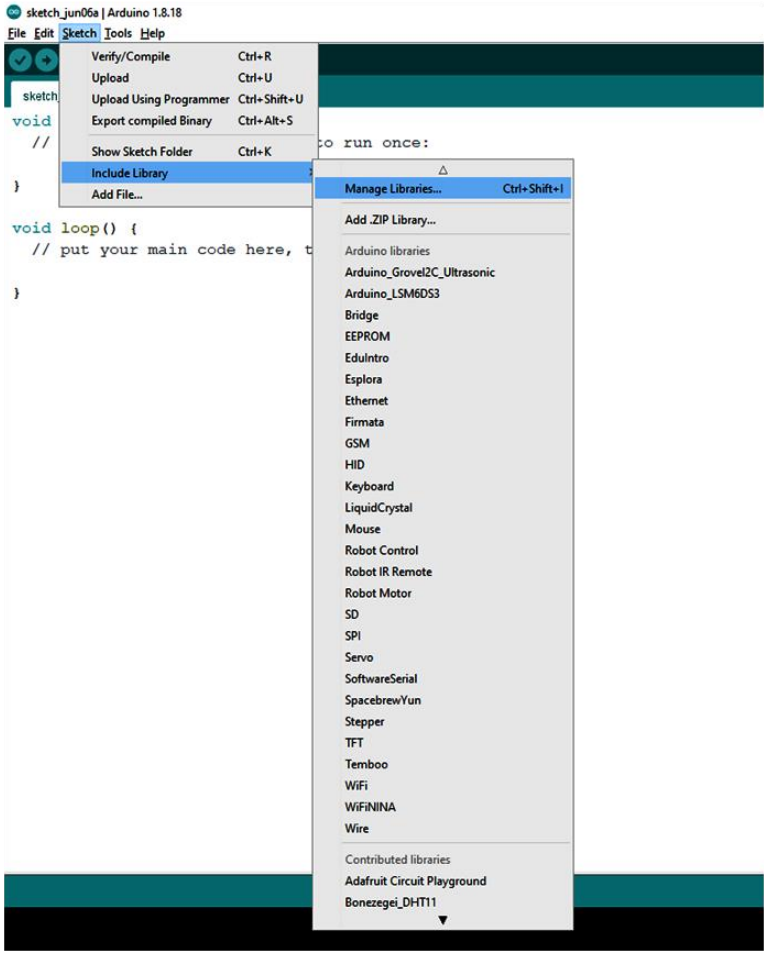

Figure II.12: Selecting Manage Libraries.

- Enter "FirmataExpress" in the search field:

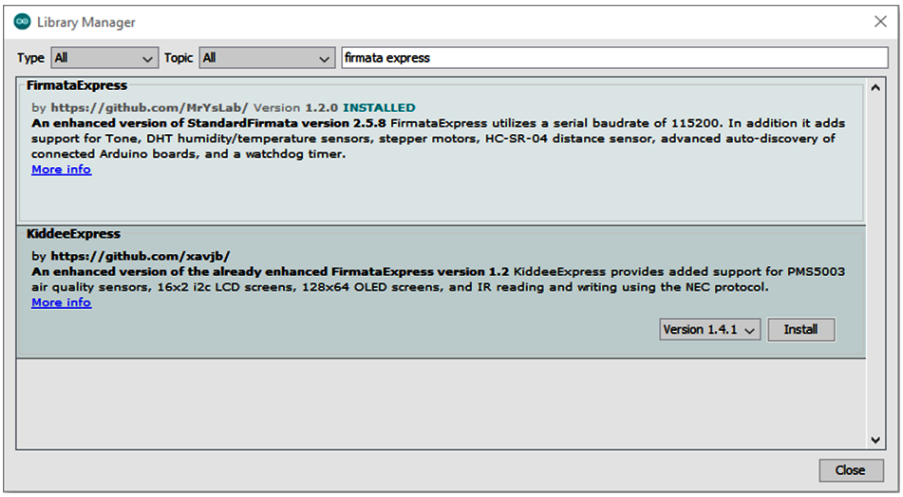

Figure II.13 : Downloading "Firmata Express" Library.

- and click on "install".

Firmata Express also requires the "Ultrasonic by Erick Simões" bookstore to be installed.

II.5.2.2 Uploading code ”Firmata Express” into Arduino memory:

- connect the Arduino via a USB port,
- To load the "Firmata Express" bookshop on the Arduino, first run the "IDE Arduino" program and choose:

File > Examples > FirmataExpress >

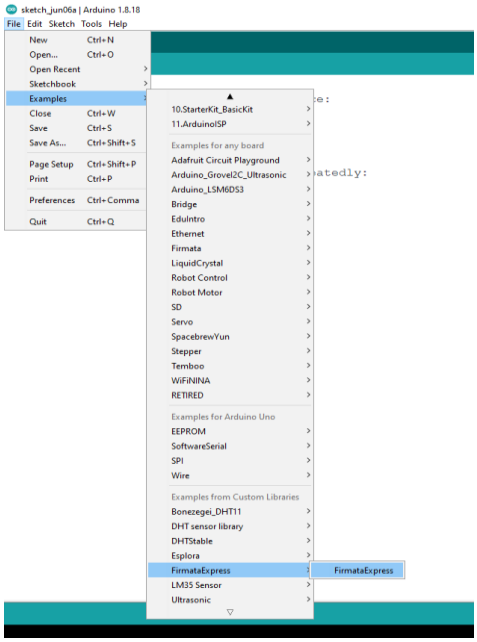

Figure II.14: Uploading code Firmata Express into Arduino memory.

- then click on ”Upload”.

II.5.2.3 Installing the "pymata-express" library in Python:

Python version 3.7 or later is the only one that works with the "pymata-express" package. Python has to have the "pymataexpress" package in order to run a Python application that communicates with Arduino using the "Firmata Express" protocol. Python must have the "pymataexpress" package.

The command-line tool "pip" can be used to install this: pip install pymata-express==1.4

The version of the library to install can be specified using the "pymata-express" installation command line.

Installing "pymata-express" version 1.4 is crucial to ensuring that the programs using the ultrasonic sensors work as intended [29].

In a Python program, you must import the "PymataExpress" module by following these instructions in order to use the "pymata-express" library:

from pymata_express.pymata_express import PymataExpress

Using this module, one can establish a serial port connection to the microcontroller by designating the COM port that the Arduino is attached to:

board = PymataExpress(com_port = PortComArduino)

Analog or digital Arduino inputs and outputs can be queried or modified after the connection has been made.

By utilizing asyncio tasks, also known as coroutines, "pymata-express" makes use of the "asyncio Python 3.7" library to enable asynchronous programming, which is a program that runs numerous codes concurrently without blocking.

Asyncio tasks that need to be defined will be used to transmit all commands sent to the Arduino.

II.5.2.4 Digital Output Management

You must first declare the digital outputs using the following instruction in order to modify their state (a digital output is the Arduino language equivalent of a "digital write").

Loop.run_until_complete(board.set_pin_mode_digital_output(pin))

Or:

board is the object created when calling the PymataExpress method of the pymata-express module,

"pin" is the microcontroller pin number that needs to be designated as a digital output.

"loop" is the asyncio task loop stated as follows: *loop = asyncio.get_event_loop()*

II.5.2.5 Digital Input Management

To read the logical state of a digital pin (e.g., pin No. 12), it is necessary to declare it in advance with the following command:

Loop.run_until_complete(board.set_pin_mode_analog_input(12))

Or:

The object created while utilizing the "PymataExpress" function of the "pymata-express" module is called "board."

"12" is the microcontroller pin number to be declared as digital input.

"loop" is the asyncio task loop stated as follows: *loop = asyncio.get_event_loop()*

Then we can read the logical state of the pin using this instruction:

value = loop.run_until_complete(board.digital_read(12))

which, at 5 V input, returns "1"; at 0 V, it returns "0"; thus produces a list with value[0] as its first element.

II.5.2.6 Management of analog outputs

You must first declare an Arduino pin using the following instruction in order to use it in analog output (PWM mode):

loop.run_until_complete(board.set_pin_mode_pwm(pin))

Or:

The object created while utilizing the "PymataExpress" function of the "pymata-express" module is called "board."

"pin" is the pin number of the microcontroller to be declared in analog output,

"loop" is the asyncio task loop stated as follows: *loop = asyncio.get_event_loop()*

Thus, the instruction for declaring the piston "11", of the object "board", in analogue output is: *Set_AnalogOutput_Pin(board, 11)*

The cyclic ratio of the PWM signal can be adjusted to vary the voltage of the piston N°11 declared in the analog output between val = 0 (0%) and val = 255 (100%):

loop.run_until_complete(board.analog_write(pin, val))

- If val = 0, then the voltage is 0 V
- if val = 255, then the voltage is 5 V
- if val= 128, then the tension is 2.5 V

II.5.2.7 Management of analog inputs

In order to obtain the voltage value of an analog input, such as pin A0, one must first declare it using the following instruction in the analog entry:

Loop.run_until_complete(board.set_pin_mode_analog_input(0))

Or:

The object created while utilizing the "PymataExpress" function of the "pymata-express" module is called "board."

The microcontroller's A0 pin number designated for analog input is "0."

"loop" is the asyncio task loop stated as follows: *loop = asyncio.get_event_loop()*

The syntax for declaring the pin A0 as a digital input is then simpler: *Set_AnalogInput_Pin(board,0)*

Then we can read the value of the pin voltage using this instruction:

value = loop.run_until_complete(board.analog_read(0))

Which returns a list whose first element (value[0]) is a decimal integer between 0 and 1023 representing a voltage between 0 to 5V.

II.6 Conclusion

This chapter presents a comprehensive overview of the Arduino Uno, which is an often-utilized acquisition card. We have provided a clear explanation for the decision-making process and thoroughly examined the available options.

Following that, we clarified the essential elements of Arduino, specifically the hardware component and the programming component, with greater accuracy. Furthermore, we have clarified the Arduino board's operational principle while ensuring the incorporation of its unique characteristics.

In addition, we investigated the Firmata protocol, which improves the Arduino environment by enabling connections between microcontrollers and host computers, enabling advanced control of hardware through other programming languages. In addition, we analyzed Firmata Express, a modified version of Firmata that reduces the command set to improve ease of use, making it especially well-suited for educational applications and quick prototyping.

Chapter III:
Realisation and discussions

III.1 Introduction

Given that the propagation speed of sound waves in the air is contingent upon temperature, we will employ our Arduino device to estimate temperature levels. An Arduino frequently uses two different types of temperature sensors:

- The sensors commonly used for this purpose are TMP 36 or LM 35, which have an output voltage that varies in a linear manner with temperature.
- The NTC thermistors are temperature sensors that show a negative temperature coefficient, which means their resistance reduces non-linearly as the temperature increases.
- To control the arduino using the Firmata communication protocol, the Python program utilizes the "pyfirmata" library. Firstly, upload the "Firmatastandard" sketch to the Arduino's memory.

The arduino utilizes sensors to control the ambient temperature of a room, oversee the functioning temperature of equipment, and monitor weather stations.

In this chapter, we will comprehensive the temperature and operation measurements using the specified circuit:

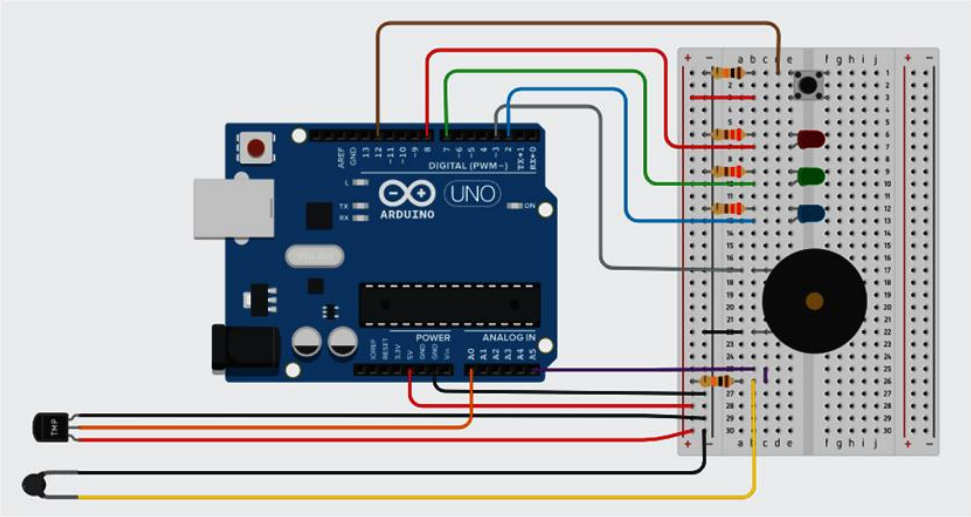

Figure III.1: Temperature and operating measurement circuit

Component list:

- 1 temperature sensor (LM 35)
- 1 thermistance
- 1 red LED
- 1 bleu LED

- 1 green LED
- 3 resistors of 220 Ω (protection resistance of LEDs)
- 2 resistors of 10 kΩ (push button and NTC resistance)
- 1 push button
- 1 speaker (ou piezo)
- 1 breadboard
- Connection Wire

III.2 Temperature measurement with LM 35 sensor

In this activity, we will measure a temperature using LM 35 sensor. This sensor operates based on the temperature dependence of the current-voltage characteristic of the silicon junction diodes, which are a key component of the sensor.

The voltage across a silicon diode changes with temperature when the current is held constant. By amplifying the voltage change, these sensors produce an analog signal that is directly proportional to the temperature.

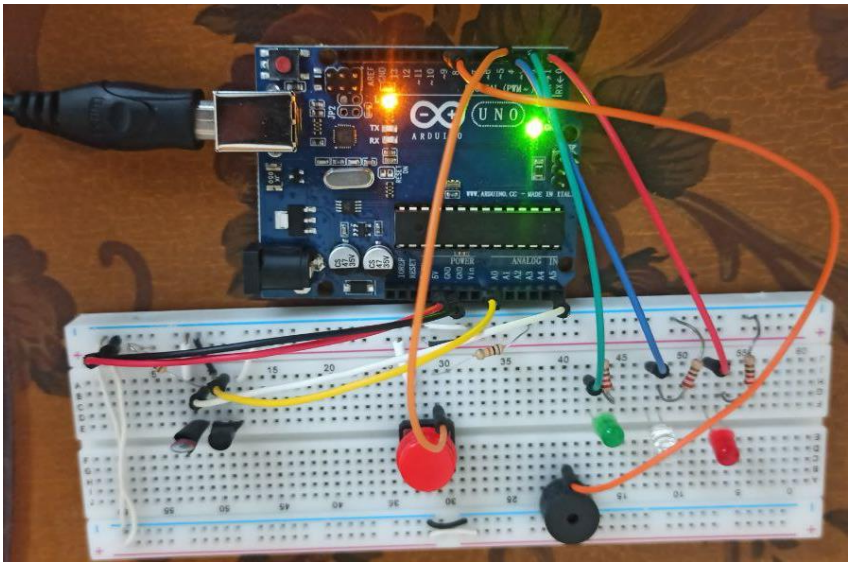

Figure III.2: The circuit on an Arduino Uno

III.2.1 Temperature sensor LM 35

The LM 35 sensor, manufactured by Texas Instrument, also has 3 pins as shown in figure III.3:

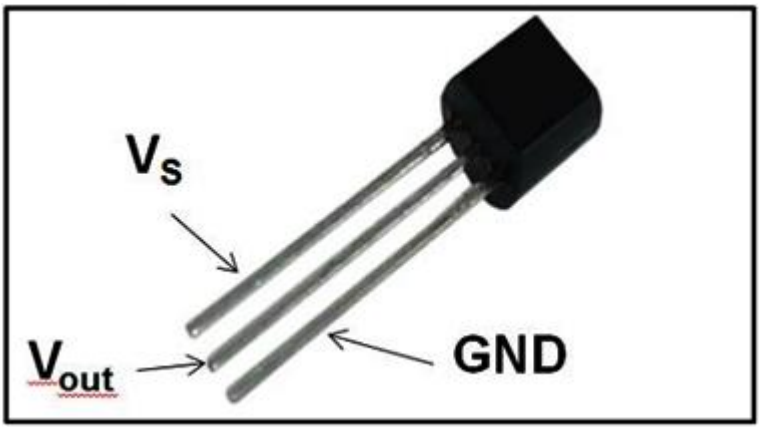

Figure III.3 : LM 35sensor

III.2.2 Characteristic

- Operating voltage: + 4 V to + 30 V.
- Calibrated output in °C (10 mV / °C).
- Measurement range: -55 °C to +150 °C.
- Accuracy: +/-0.5°C at 25°C and +/-1°C to -55°C or +150°C.
- Output voltage: -0.55V (-55°C) to 1.5V (150°C).
- Charge current: 0.06 mA

When using the LM35 sensor, if the temperature falls below 0°C, the output voltage becomes negative (-10°C corresponds to -0.1 V). Indeed, utilizing an Arduino allows for the measurement of solely positive temperatures because of the requirement that the voltage supplied to an analog input of the microcontroller must strictly stay in the range of 0 to 5 volts [4].

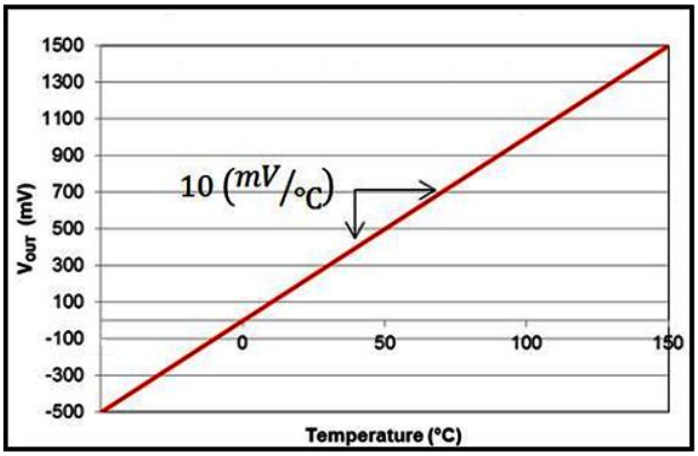

Figure III.4: The progression of converting the voltage output of the sensor into temperature in degrees Celsius

The process of converting the voltage output of the sensor into temperature in degrees Celsius is as follows:

T (°C) = Output voltage (mV) / 10

Which is: T (°C) = Output voltage (V) * 100

III.2.3 Programme

Written in either Python or Arduino language, the activity code retrieves the decimal value of pin A0, which ranges from 0 to 1023. It then transforms this value into voltage V, a decimal number ranging from 0 to 5 V. Finally, it displays the temperature corresponding to this voltage in the Python Shell window or serial monitor.

III.2.3.1 Python Program

```
import pyfirmata
import time
# Replace 'COM3' with the port our Arduino is connected to
board = pyfirmata.Arduino('COM3')
# Start an iterator thread so the serial buffer doesn't overflow
it = pyfirmata.util.Iterator(board)
it.start()
# Define the analog pin for the sensor
analog_pin = board.get_pin('a:0:i')
# Define the digital pin for the button
button_pin = board.get_pin('d:5:i')
voltage=[]
# Flag to indicate whether the program should be running or not
running = False
time.sleep(1)  # Give some time for the connection to settle
print("press the button to start ")
def read_temperature(sensor_value):
 # Convert the analog value to voltage (assuming a 5V system)
voltage = (sensor_value) * 5.0
# Convert the voltage to temperature in Celsius (for LM35)
temperature_c = voltage * 100.0
return temperature_c
try:
while True:
 # Read the state of the button
button_state = button_pin.read()
# If button is pressed and the program is not already running, start the program
        if button_state == 1 and not running:
         running = True
         print("Program started")
         print("sensorValue : Voltage : temperature ")
    # If button is pressed and the program is running, stop the program
elif button_state == 1 and running:
        running = False
```

```
            print("Program stopped")
            print("press the button to start ")
        # If the program is running, read the sensor value and print the temperature
    if running:
            sensor_value = analog_pin.read()
             if sensor_value is not None:
            temperature = read_temperature(sensor_value)
            print(f"{sensor_value*1023:.2f}: {sensor_value*5:.2f}:  {temperature:.2f}°C")
    time.sleep(1)
    except KeyboardInterrupt:
    print("Program stopped")
    finally:
    board.exit()
```

III.2.3.2 Progress of the programme

- ✓ **Importing library:**

– import pyfirmata: This imports the pyfirmata library, which is used to communicate with an Arduino board.

– import time: This imports the time library, which provides various time-related functions.

- ✓ **Connecting to Arduino:**

– This initializes a connection to the Arduino board on the specified port (COM3).board = pyfirmata.Arduino('COM3').

– This starts an iterator thread that continuously reads data from the Arduino, preventing the serial buffer from overflowing.

```
it = pyfirmata.util.Iterator(board)
it.start()
```

- ✓ **Declaration of constants and variables:**

– We defines the analog pin A0 on the Arduino as an input pin for reading sensor values.

```
analog_pin = board.get_pin('a:0:i')
```

– This defines the digital pin D5 on the Arduino as an input pin for reading the button state.

```
button_pin = board.get_pin('d:5:i')
```

– We define the function read_temperature to accept a sensor value as an argument, convert it to voltage, and then convert it back to temperature in Celsius. The system relies on an LM35 temperature sensor, which produces an output voltage that is directly proportional to the temperature, with a gradient of 10 mV per degree Celsius.

```
def read_temperature(sensor_value):
 # Convert the analog value to voltage (assuming a 5V system)
voltage = (sensor_value) * 5.0
# Convert the voltage to temperature in Celsius (for LM35)
 temperature_c = voltage * 100.0
return temperature_c
```

– This starts an infinite loop and reads the state of the button connected to D5.

```
try:
while True:
# Read the state of the button
button_state = button_pin.read()
```

– If the button is touched (button_state == 1) and the program is not currently running (not running), it assigns the value True to the variable "running," indicating that the main part of the program should commence. In addition, it displays the message "Program started" as well as a heading for the sensor readings, voltage, and temperature.

```
if button_state == 1 and not running:
        running = True
        print("Program started")
        print("sensorValue : Voltage : temperature ")
```

– If the button is touched (button_state == 1) and the program is currently running (running), it changes the value of running to false, halting the main portion of the program. In addition, it displays the message "Program stopped" and requests the user to push the button in order to restart the program.

```
elif button_state == 1 and running:
        running = False
        print("Program stopped")
        print("press the button to start ")
```

– A0. If a sensor_value is proper (i.e., not None), it is converted to temperature using the read_temperature function. The sensor value, related voltage, and temperature are then printed.

```
if running:
        sensor_value = analog_pin.read()
        if sensor_value is not None:
```

```
temperature = read_temperature(sensor_value)
print(f"     {sensor_value:.2f}:  {sensor_value*5:.2f}:  {temperature:.2f}°C")
```

III.2.3.3 Results in the Python Shell window:

```
press the button to start
Program started
sensorValue : Voltage : temperature
     0.06:  0.31:  30.80°C
     0.06:  0.31:  30.80°C
     0.06:  0.30:  30.30°C
     0.06:  0.31:  31.30°C
     0.06:  0.31:  30.80°C
     0.06:  0.31:  30.80°C
     0.06:  0.30:  30.30°C
     0.06:  0.31:  30.80°C
     0.06:  0.31:  30.80°C
     0.06:  0.30:  30.30°C
     0.06:  0.31:  30.80°C
     0.06:  0.31:  30.80°C
     0.06:  0.31:  30.80°C
     0.06:  0.30:  30.30°C
     0.06:  0.31:  30.80°C
Program stopped
```

Figure III.5: Results in the Python Shell window:

III.2.3.3 Programme in langage Arduino

```
// Define the analog pin for the sensor
const int analogPin = A0;
// Define the digital pin for the button
const int buttonPin = 5;
// Variable to store the state of the button
int buttonState = 0;
int lastButtonState = 0;  // Variable to store the previous button state
// Flag to indicate whether the program should be running or not
bool running = false;
void setup() {
// Initialize serial communication at 9600 bits per second
Serial.begin(9600);
// Initialize the button pin as an input
pinMode(buttonPin, INPUT);
// Print initial message
Serial.println("Press the button to start");
}
```

```
void loop() {
// Read the state of the button
buttonState = digitalRead(buttonPin);
 // If the button is pressed
if (buttonState == HIGH && lastButtonState == LOW) {
// Toggle the running state
 running = !running;
 if (running) {
  Serial.println("Program started");
  Serial.println("sensorValue : Voltage : temperature");
   } else {
    Serial.println("Program stopped");
    Serial.println("Press the button to start");
   }
   // Debounce delay
   delay(50);
  }
  // Update the last button state
  lastButtonState = buttonState;
  // If the program is running, read the sensor value and print the temperature
  if (running) {
   // Read the analog input
   int sensorValue = analogRead(analogPin);
   // Convert the analog value to voltage (assuming a 5V system)
   float voltage = sensorValue * (5.0 / 1023.0);
   // Convert the voltage to temperature in Celsius (for LM35)
   float temperatureC = voltage * 100.0;
   // Print the sensor value, voltage, and temperature
   Serial.print("     ");
   Serial.print(sensorValue);
   Serial.print(":  ");
   Serial.print(voltage);
   Serial.print(":  ");
   Serial.print(temperatureC);
   Serial.println("°C");
   // Delay for a second
   delay(1000);
  }
}
```

III.2.3.4 Progress of the programme:

- ✓ **Pin Definitions**:
- analogPin is set to A0 for the temperature sensor.
- buttonPin is set to 5 for the button.
- ✓ **Variable Declarations**:
- buttonState stores the current state of the button.

- lastButtonState stores the previous state of the button to detect changes.

✓ **Setup:**

- Serial communication is initialized at 9600 bps.
- The button pin is set as an input.
- An initial message is printed to the serial monitor.

Loop:

- The button state is read.
- If the button state changes from LOW to HIGH (indicating a press), the running flag is toggled.
- If the program starts, a message is printed.
- If the program stops, a different message is printed.
- If the program is running, the sensor value is read from the analog pin, converted to voltage and temperature, and printed to the serial monitor.
- A 1-second delay controls the rate of reading and printing.

III.2.3.5 Results in serial monitor

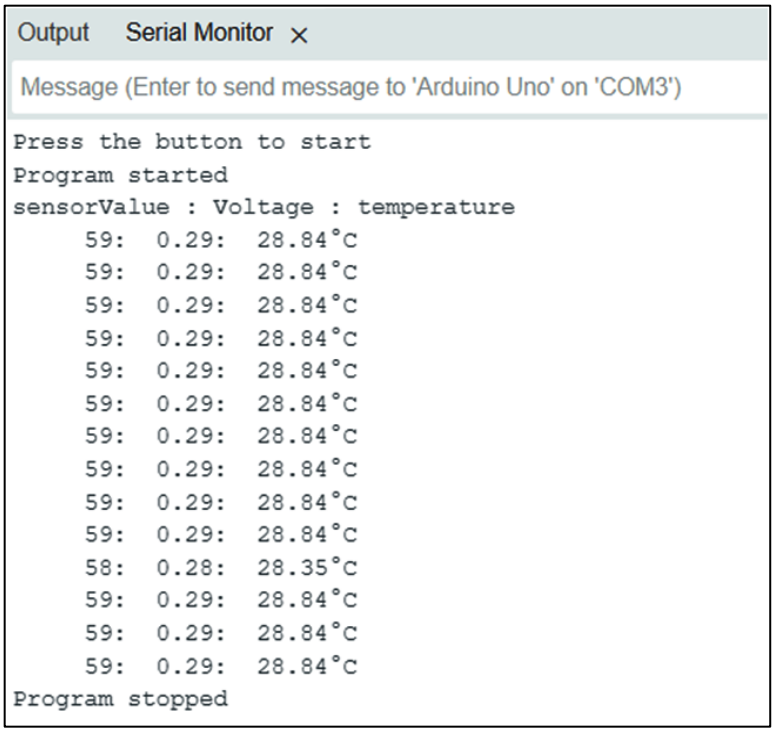

Figure III.6: Results in serial monitor

III.3 Sound alarm by exceeding temperature

The purpose of this activity is to create a tendency and visual alert that will activate when the temperature measured by the LM 35 sensor in our testing circuit exceeds a predetermined threshold value. This will help prevent users from exceeding the recommended temperature limit for a particular piece of equipment.

To do this, it is only necessary to summarize the program from the activity before and add the code for the sound and visual alarm that has previously been covered. The temperature measurements begin on pushing a button and stop when pressing it again.

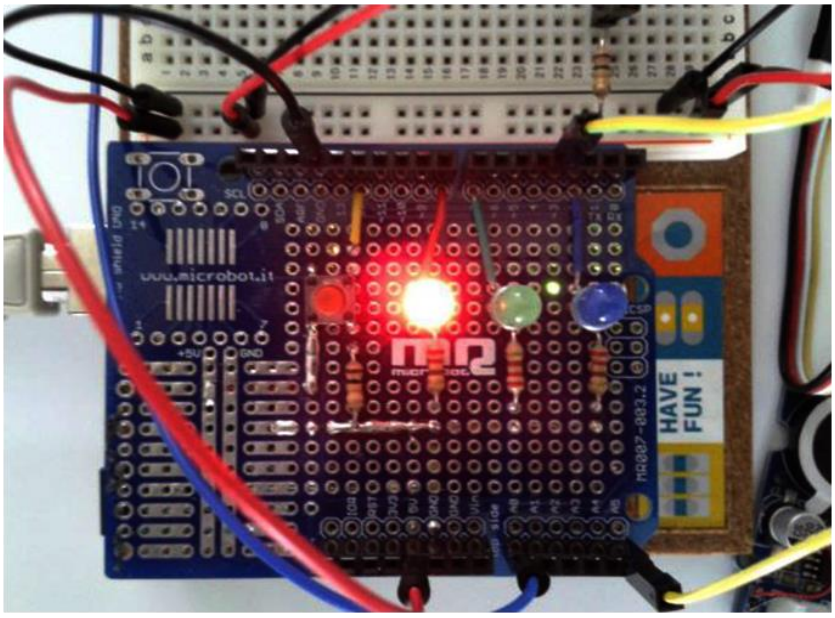

Figure III.7: The circuit on an Arduino Uno using sound alarm

III.3.1 Programme

The Python or Arduino code for the activity can be changed to see the effect of variables such as temperature threshold, sound wave frequency, broadcast time, and silence period.

III.3.1.2 Python Program

```
import pyfirmata
import time
# Configuration of pins
PinSensor = 0
# Initialization of variables
ValSensor = 0
tension = 0.0
```

```
Temp = 0.0
OldTemp = 0.0
TempAlarme = 30
ValButton = 0
OldValButton = 0
State = 0
OldState = 0
# Initialization of the Arduino board
board = pyfirmata.Arduino('COM3')  # Replace with the appropriate serial port for your system
# Configuration of inputs and outputs
it = pyfirmata.util.Iterator(board)
it.start()
# Define the analog pin for the sensor
analog_pin = board.get_pin('a:0:i')
button = board.get_pin('d:5:i')  # Pin 12 as digital input
led = board.get_pin('d:2:o')  # Pin 8 as digital output
PIEZO_PIN = board.get_pin('d:9:p')
print("Press the push button to start measurements.")
try:
    while True:
        ValButton = button.read()
        time.sleep(0.01)
        if ValButton and not OldValButton:
            State = 1 - State
        OldValButton = ValButton
        if State == 1:
            if OldState == 0:
                print("Temperature measurement in progress.")
                print("")
                print("Temperature in degrees Celsius:")
                OldState = 1
            ValSensor = board.analog[PinSensor].read()
            if ValSensor is not None:
                tension = ValSensor * 5.0
                # LM 35 sensor
                Temp = tension * 100
                if OldTemp != Temp:
                    OldTemp = Temp
                if Temp > TempAlarme:
                    print(f"{Temp:.1f}:alarme")
                    led.write(1)
                    PIEZO_PIN.write(0.2)
                    board.pass_time(0.5)  # Allow the buzzer to produce sound
                else:
                    print(f"{Temp:.1f}")
                    led.write(0)
                    PIEZO_PIN.write(0)
                    board.pass_time(0.5)  # Deactivates the buzzer
            time.sleep(0.1)
```

```
        else:
            if OldState == 1:
                print("End of measurements.")
                OldState = 0
                led.write(0)
                PIEZO_PIN.write(0)
                board.pass_time(0.5)
except KeyboardInterrupt:
    print("Program interrupted")
    board.exit()
```

III.3.1.3 Results in the Python Shell window:

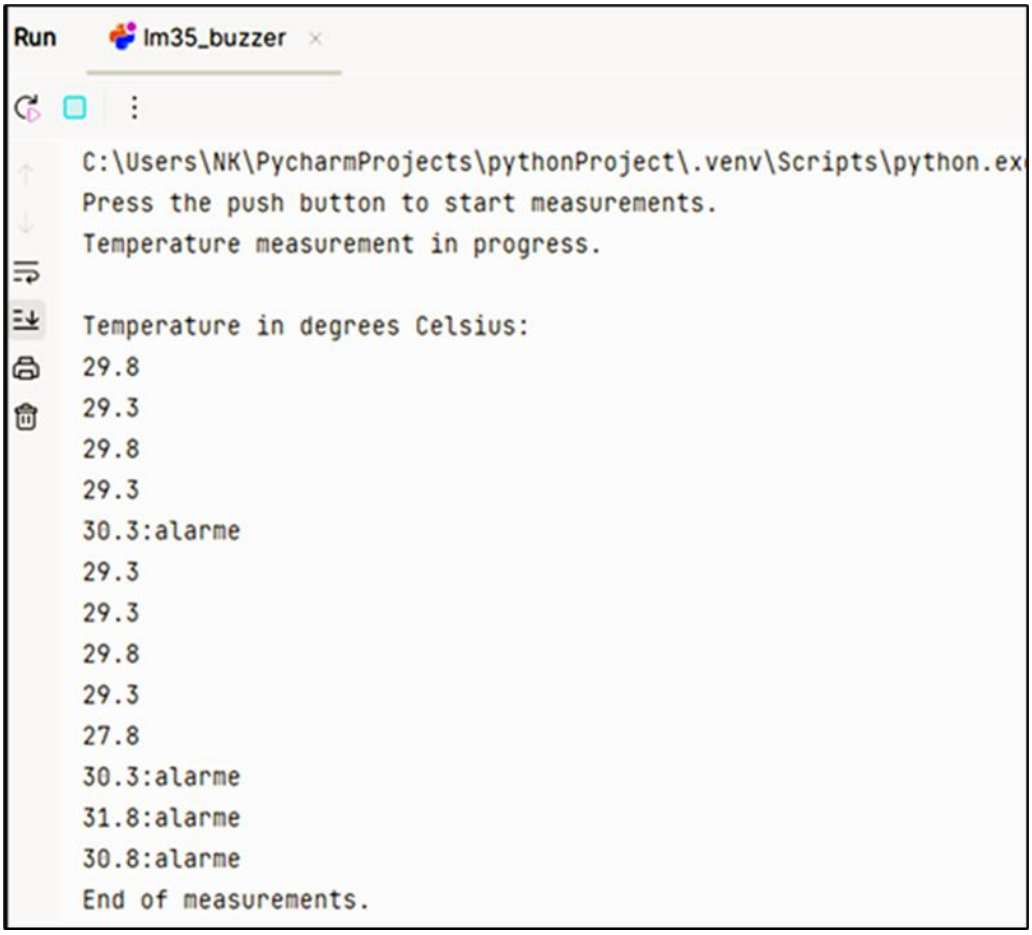

Figure III.8: Results in the Python Shell window using sound alarm.

III.3.1.4 Program in Arduino language:

```
// Configuration of pins
const int PinSensor = A0;     // Analog pin for the temperature sensor
const int buttonPin = 5;      // Digital pin for the push button
const int ledPin = 2;         // Digital pin for the LED
const int piezoPin = 9;       // Digital pin for the piezo buzzer
// Initialization of variables
```

```
float ValSensor = 0;
float tension = 0.0;
float Temp = 0.0;
float OldTemp = 0.0;
const float TempAlarme = 30.0;
int ValButton = 0;
int OldValButton = 0;
int State = 0;
int OldState = 0;
void setup() {
 // Initialize the serial communication
 Serial.begin(9600);
// Configure pin modes
 pinMode(buttonPin, INPUT);
 pinMode(ledPin, OUTPUT);
 pinMode(piezoPin, OUTPUT);
// Initial states
 digitalWrite(ledPin, LOW);
digitalWrite(piezoPin, LOW);
Serial.println("Press the push button to start measurements.");
}
void loop() {
ValButton = digitalRead(buttonPin);
delay(10);  // Debounce delay
// Toggle state on button press
 if (ValButton == HIGH && OldValButton == LOW) {
State = 1 - State;
 }
OldValButton = ValButton;
 if (State == 1) {
          if (OldState == 0) {
          Serial.println("Temperature measurement in progress.");
           Serial.println("");
           Serial.println("Temperature in degrees Celsius:");
            OldState = 1;
          }
ValSensor = analogRead(PinSensor);
tension = (ValSensor / 1024.0) * 5.0;  // Convert analog reading to voltage
Temp = tension * 100;  // Convert voltage to temperature (LM35)
          if (OldTemp != Temp) {
          OldTemp = Temp;
          }
          if (Temp > TempAlarme) {
          Serial.print(Temp, 1);
          Serial.println(":alarme");
          digitalWrite(ledPin, HIGH);
          analogWrite(piezoPin, 51);  // Activate piezo buzzer with low volume
          } else {
          Serial.println(Temp, 1);
          digitalWrite(ledPin, LOW);
```

```
            analogWrite(piezoPin, 0);  // Deactivate piezo buzzer
            }
            delay(1000);  // Delay for measurement frequency
            } else {
             if (OldState == 1) {
            Serial.println("End of measurements.");
            OldState = 0;
            digitalWrite(ledPin, LOW);
             analogWrite(piezoPin, 0);  // Ensure buzzer is off
   }
  }
}
```

III.3.1.5 Results in serial monitor:

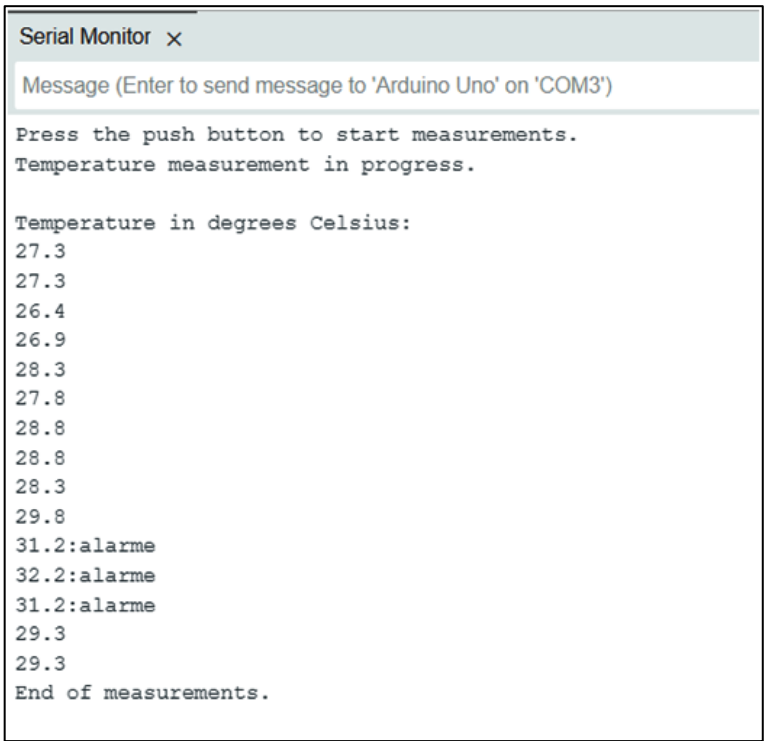

Figure III.9: Results in serial monitor using sound alarm.

III.4 Electroluminescent diode thermometer:

In this section, we will utilize the red, green, and blue LEDs of the study circuit to visually represent the region in which the temperature measured by a sensor LM 35 (measured) is in relation to a reference value (Tref) and a defined temperature difference (ΔT).

- if Tmed < Tref - ΔT: The blue LED is on,
- if TMed >Tref + ΔT: The red LED is on,

- Tref - ΔT < Tmeasured < Tref + ΔT: The green LED is on.

III.4.1 Programme

The code in Python or Arduino language can be modified to see the effect of the variables (reference temperature, temperature difference) on the activity.

III.4.1.1 Python Program

```
import pyfirmata
import time
# Replace 'COM3' with the port your Arduino is connected to
board = pyfirmata.Arduino('COM3')
# Start an iterator thread so the serial buffer doesn't overflow
it = pyfirmata.util.Iterator(board)
it.start()
# Define the analog pin for the sensor
analog_pin = board.get_pin('a:0:i')
# Define the digital pin for the button
button_pin = board.get_pin('d:5:i')
# Define the digital pins for the LEDs
led_red_pin = board.get_pin('d:2:o')
led_green_pin = board.get_pin('d:3:o')
led_blue_pin = board.get_pin('d:4:o')
# Flag to indicate whether the program should be running or not
running = False
time.sleep(1)  # Give some time for the connection to settle
DT=6
tempref=30
print("Press the push button to start measurements.")

def read_temperature(sensor_value):
    # Convert the analog value to voltage (assuming a 5V system)
    voltage = sensor_value * 5.0
    # Convert the voltage to temperature in Celsius (for LM35)
    temperature_c = voltage * 100.0
    return temperature_c
try:
    while True:
        # Read the state of the button
        button_state = button_pin.read()
        # If button is pressed and the program is not already running, start the program
        if button_state == 1 and not running:
            running = True
            print("Temperature measurement in progress.")
            print("")
            print("Temperature in degrees Celsius:")
        # If button is pressed and the program is running, stop the program
        elif button_state == 1 and running:
            running = False
```

```
            # Turn off all components
            led_red_pin.write(0)
            led_green_pin.write(0)
            led_blue_pin.write(0)
            print("End of measurements.")
        # If the program is running, read the sensor value and print the temperature
        if running:
            sensor_value = analog_pin.read()
            if sensor_value is not None:
                temperature = read_temperature(sensor_value)
                print(f"{temperature:.2f}°C")
                # Activate the appropriate LED based on temperature thresholds
                if temperature >= tempref-DT and temperature <= tempref+DT:
                    led_green_pin.write(1)
                    led_red_pin.write(0)
                    led_blue_pin.write(0)
                elif temperature > tempref+DT:
                    led_red_pin.write(1)
                    led_green_pin.write(0)
                    led_blue_pin.write(0)
                elif temperature < tempref-DT:
                    led_blue_pin.write(1)
                    led_green_pin.write(0)
                    led_red_pin.write(0)
        time.sleep(1)
except KeyboardInterrupt:
    print("Program stopped")
finally:
    board.exit()
```

III.4.1.2 Results in the Python Shell window:

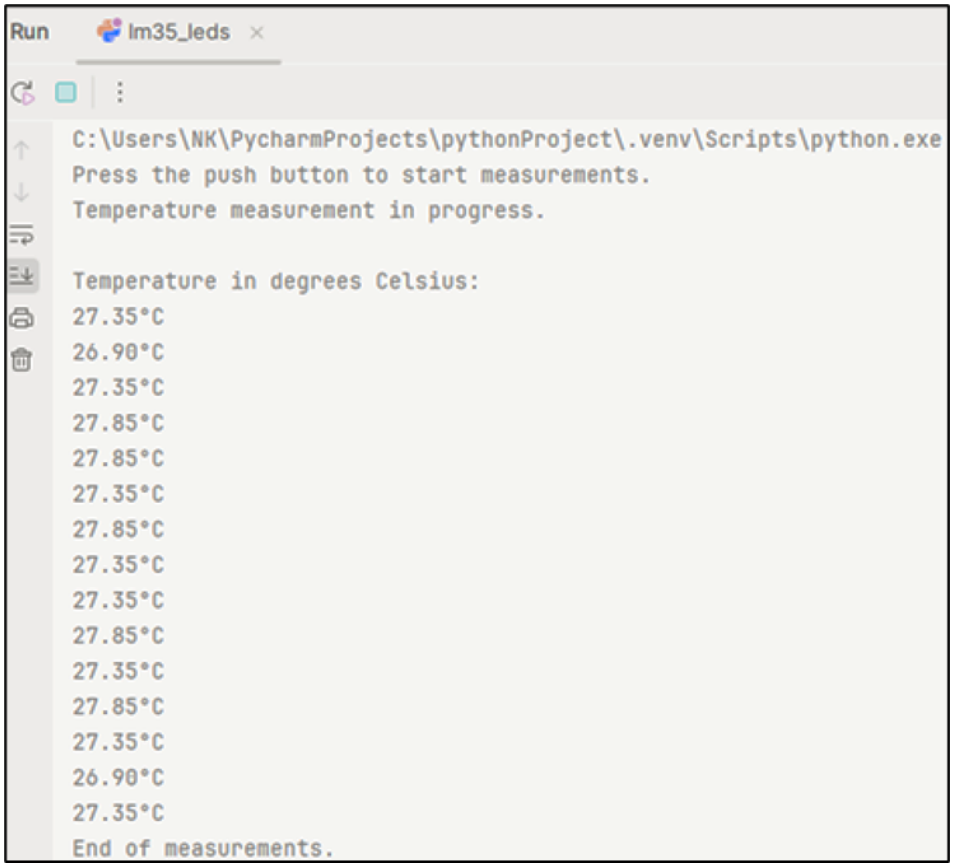

Figure III.10: Results in the Python Shell window using Electroluminescent diode thermometer.

III.4.1.3 Program in Arduino language:

```
const int analogPin = A0; // Analog pin for the temperature sensor
const int buttonPin = 5;  // Digital pin for the button
const int ledRedPin = 2;  // Digital pin for the red LED
const int ledGreenPin = 3; // Digital pin for the green LED
const int ledBluePin = 4;  // Digital pin for the blue LED
bool running = false;
const float tempRef = 30.0;
const float DT = 6.0;
void setup() {
  Serial.begin(9600);
  pinMode(buttonPin, INPUT);
  pinMode(ledRedPin, OUTPUT);
  pinMode(ledGreenPin, OUTPUT);
  pinMode(ledBluePin, OUTPUT);
  Serial.println("Press the push button to start measurements.");
}
float readTemperature(int sensorValue) {
  // Convert the analog value to voltage (assuming a 5V system)
  float voltage = sensorValue * (5.0 / 1023.0);
  // Convert the voltage to temperature in Celsius (for LM35)
```

```
  float temperatureC = voltage * 100.0;
  return temperatureC;
}
void loop() {
  int buttonState = digitalRead(buttonPin);
 // If button is pressed and the program is not already running, start the program
  if (buttonState == HIGH && !running) {
    running = true;
    Serial.println("Temperature measurement in progress.");
    Serial.println("");
    Serial.println("Temperature in degrees Celsius:");
  }
  // If button is pressed and the program is running, stop the program
  else if (buttonState == HIGH && running) {
    running = false;
    // Turn off all components
    digitalWrite(ledRedPin, LOW);
    digitalWrite(ledGreenPin, LOW);
    digitalWrite(ledBluePin, LOW);
    Serial.println("End of measurements.");
  }
  // If the program is running, read the sensor value and print the temperature
  if (running) {
    int sensorValue = analogRead(analogPin);
    float temperature = readTemperature(sensorValue);
    Serial.print(temperature);
    Serial.println("°C");
        // Activate the appropriate LED based on temperature thresholds
    if (temperature >= tempRef - DT && temperature <= tempRef + DT) {
      digitalWrite(ledGreenPin, HIGH);
      digitalWrite(ledRedPin, LOW);
      digitalWrite(ledBluePin, LOW);
    }
    else if (temperature > tempRef + DT) {
      digitalWrite(ledRedPin, HIGH);
      digitalWrite(ledGreenPin, LOW);
      digitalWrite(ledBluePin, LOW);
    }
    else if (temperature < tempRef - DT) {
      digitalWrite(ledBluePin, HIGH);
      digitalWrite(ledGreenPin, LOW);
      digitalWrite(ledRedPin, LOW);
    }
  }
  delay(1000); // Delay for 1 second
}
```

III.5 Calibration of a thermistance (NTC)

The objective of this activity is to calibrate a thermistor with unknown characteristic dimensions by utilizing an LM 35 temperature sensor.

In order to do this, we will adjust the temperature and instruct the Arduino to measure the thermistor's resistance, which correlates with the temperature the sensor is concurrently detecting; next to defining the relation between resistance and temperature, we can accurately determine a thermistor's temperature by measuring its resistance.

The resistance of a thermistor doesn't change linearly with temperature. However, if the particular relation between resistance and temperature is known, especially for a negative temperature coefficient (NTC) thermistor, it is indeed possible to utilize it for highly accurate temperature measurements.

When the Joule effect, which refers to the heating caused by the flow of electric current, can be ignored, the relationship between the NTC's resistance (negative temperature coefficient) and its temperature can be expressed using the Steinhart-Hart equation [10]:

$$\frac{1}{T} = A + Bln(R_T) + C(\ln(R_T))^3 \quad \text{(III.1)}$$

(R_T): is the resistance (in ohms) of the sensor at temperature T (en kelvins).

A, B and C are Steinhart–Hart coefficients (given by the manufacturer or obtained experimentally with three reference measurements) that are component characteristic constants valid at any temperature.

This formula, valid for all temperatures, can be simplified over a limited range of temperatures. The formula becomes:

$$\frac{R_T}{R_0} = e^{(\beta \times \left(\frac{1}{T} - \frac{1}{T_0}\right))} \quad \text{(III.2)}$$

R_T: is the resistance (in ohms) of the sensor at temperature T (en kelvins).

R_0: is the announced resistance at a reference temperature (often 25 °C).

B: (in kelvins) is a coefficient considered constant by approximation, the use of which is limited to a temperature range [T1;T2]:

$$\beta = \frac{T_1 \times T_2}{T_2 - T_1} \times \ln\left(\frac{R_1}{R_2}\right) \quad \text{(III.3)}$$

The main characteristic of a NTC or PTC is the resistance value R_0 (in Ω) at 25°C. Typically, the component marking is used to represent the value of R_0. The number consists of three digits. The first two digits reflect the value, while the third digit represents the multiplier

coefficient in powers of 10. Or following table is used to know the value of color bands of resistors ;

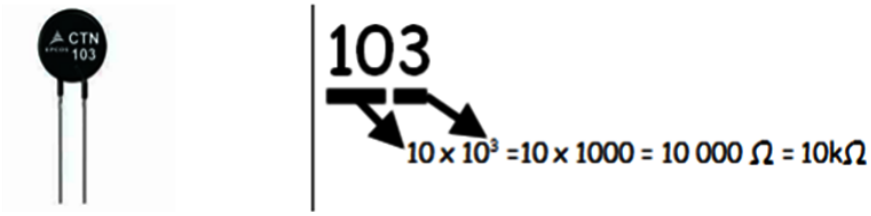

Figure III.11: The main characteristic of a NTC thermistor.

Color	First Band	Second Band	Third Band (Multiplier)	Fourth Band (Tolerance)	Fifth Band (Temperature Coefficient)
Black	0	0	1		250
Brown	1	1	10	± 1%	100
Red	2	2	100	± 2%	50
Orange	3	3	1000		15
Yellow	4	4	10000		25
Green	5	5	100000	± 0.5%	20
Blue	6	6	1000000	± 0.25%	10
Violet	7	7	10000000	± 0.1%	5
Grey	8	8		± 0.05%	1
White	9	9			
Gold				± 5%	
Silver				± 10%	
None				± 20%	

Figure III.12: Color Coding of Resistors [15].

Following method is used to calculate resistance value from the color bands of resistors using color coding table given above:

Table III.1: Method is used to calculate resistance value from the color bands of resistors using color coding [15].

First Band	Second Band	Third Band	Fourth Band	Fifth Band
		Multiplier Band	**Tolerance**	
Write down value of first band.	Write down value of second band.	Multiply two digit number obtained from first and second bands with the value of third band.	Calculate %age value of figure obtained from 1st three bands. Prefix ± before value of fourth band.	

With an Arduino, We use the thermistance in a mounting (see below), with a fixed resistance of 10 kΩ, which is called a voltage divider.

The thermistor is supplied with a voltage of 5V from the Arduino. The connection point between the two resistors is linked to an analog pin of the Arduino, and the voltage of this pin is measured using the "analogRead" function.

- Voltage Divider:

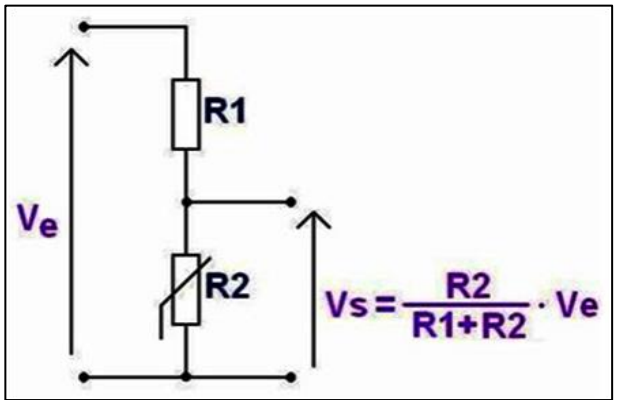

Figure III.13: Voltage Divider.

The thermistor R_2 exhibits variations in resistance depending on the temperature (T), resulting in changes in the current flowing through the circuit and consequently altering the voltage (V_s) between the terminals of R_2.

The resistance R_1, with a value of 10 kΩ, serves to restrict the flow of current in the circuit. The V_e voltage is supplied by the ARDUINO and has a value of 5V. V_s is the voltage measured by the microcontroller. We can then calculate R_2:

$$R_2 = \frac{V_{out}(R_1+R_2)}{V_{in}} \quad \text{(III.4)}$$

$$R_2 = \frac{V_{out}\times R_1}{(V_{in}-V_{out})} \quad \text{(III.5)}$$

III.5.1 Programme

At this point the code of the activity in Python and Arduino language is here.

III.5.1.1 Python Program

```
import pyfirmata
import time
import matplotlib.pyplot as plt
# Set up the Arduino board
board = pyfirmata.Arduino('COM3')
# Define pins
pinTMP = board.get_pin('a:0:i')  # Analog pin A0
```

```
pinCTN = board.get_pin('a:5:i')  # Analog pin A5
pinButton = board.get_pin('d:5:i')  # Digital pin 5
# Start iterator to read analog inputs
it = pyfirmata.util.Iterator(board)
it.start()
# Initial values
old_temp_tmp = 0.0
state = 0
old_state = 0
old_val_button = 0
print("Press the push button to start measurements.")
# Lists to store temperature and resistance values
temperatures = []
resistances = []
while True:
 val_button = pinButton.read()
 time.sleep(0.1)  # Debounce delay
 if val_button is None:
    continue
 if (val_button == 1) and (old_val_button == 0):
    state = 1 - state
 old_val_button = val_button
 if state == 1:
    if old_state == 0:
       print("Measuring temperature and thermistor resistance.")
       print("")
       print("Temperature in degrees Celsius ; Thermistor resistance in Ohms:")
       old_state = 1
    val_tmp = pinTMP.read()
    val_ctn = pinCTN.read()
    if val_tmp is not None and val_ctn is not None:
       # Convert sensor readings
       temp_tmp = val_tmp * 5.0 * 100  # LM35 calculation
       v_ctn = val_ctn * 5.0
       r_ctn = 10000 * v_ctn / (5 - v_ctn)
       if old_temp_tmp != temp_tmp:
          print(f"{temp_tmp:.1f} ; {r_ctn:.1f}")
          old_temp_tmp = temp_tmp
       # Append to lists
            temperatures.append(temp_tmp)
            resistances.append(r_ctn)
      time.sleep(0.1)  # Measurement delay
   else:
      if old_state == 1:
         print("End of measurements.")
         # Plot the graph
         plt.plot(temperatures, resistances, marker='o', linestyle='-')
         plt.xlabel('Temperature (°C)')
         plt.ylabel('Thermistor Resistance (Ohms)')
         plt.title('Thermistor Resistance vs Temperature')
```

```
plt.grid(True)
plt.show()
old_state = 0
```

III.5.1.2 Results in the Python Shell window:

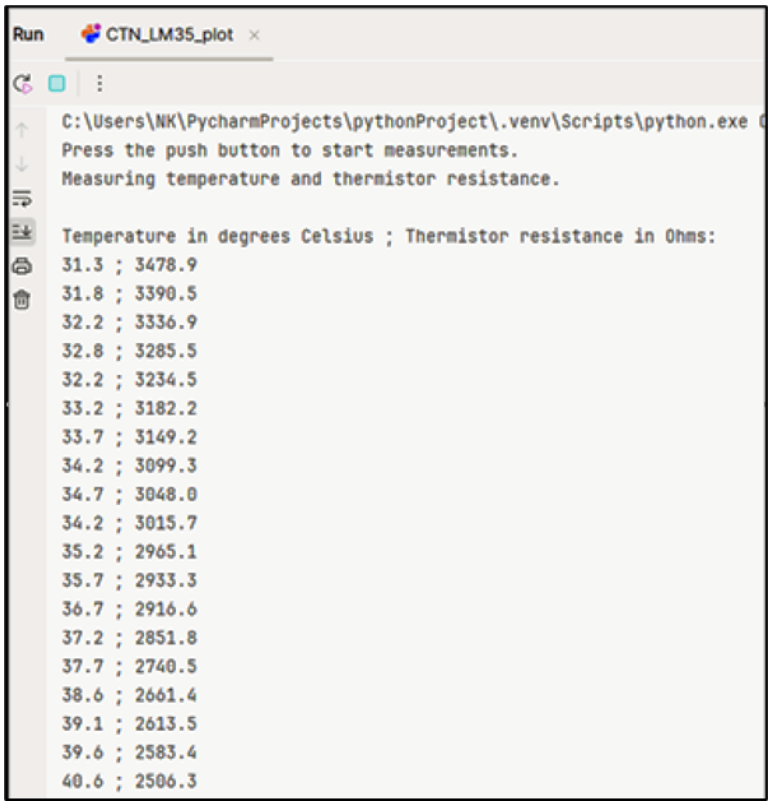

Figure III.14: Results in the Python Shell window using NTC thermistor and Lm 35 temperature sensor.

III.5.1.3 Program in Arduino language

```
// Declaration of constants and variables
const int PinTMP = 0;
const int PinCTN = 5;
const int PinButton = 5;
int ValTMP = 0;
int ValCTN = 0;
float TempTMP = 0.0;
float OldTempTMP = 0.0;
float Rctn = 0.0;
float Vctn = 0.0;
int ValButton = 0;
int OldValButton = 0;
int State = 0;
int OldState = 0;
```

```
// Initialization of inputs and outputs
void setup() {
  Serial.begin(9600);
  pinMode(PinButton, INPUT);
  Serial.println("Press the push button to start measurements.");
}
// Main loop function
void loop() {
  ValButton = digitalRead(PinButton);
  delay(1000);
  if ((ValButton == HIGH) && (OldValButton == LOW)) {
    State = 1 - State;
  }
  OldValButton = ValButton;
  if (State == 1) {
    if (OldState == 0) {
      Serial.println("Measuring temperature and NTC resistance.");
      Serial.println("");
      Serial.println("Temperature in Celsius; NTC resistance in Ohms:");
      OldState = 1;
    }
    ValTMP = analogRead(PinTMP);
    // TMP 36 Sensor
    //TempTMP = ((ValTMP/1023.0)*5.0 - 0.5) * 100;
    // LM 35 Sensor
    TempTMP = (ValTMP / 1023.0) * 5.0 * 100;
    ValCTN = analogRead(PinCTN);
    Vctn = (ValCTN / 1023.0) * 5.0;
    Rctn = 10000 * Vctn / (5 - Vctn);
    if (OldTempTMP != TempTMP) {
      Serial.print(TempTMP, 1);
      Serial.print(" ; ");
      Serial.println(Rctn, 1);
      OldTempTMP = TempTMP;
    }
    delay(100);
  } else {
    if (OldState == 1) {
      Serial.println("End of measurements.");
      OldState = 0;
    }
  }
}
```

III.5.1.4 Results in serial monitor:

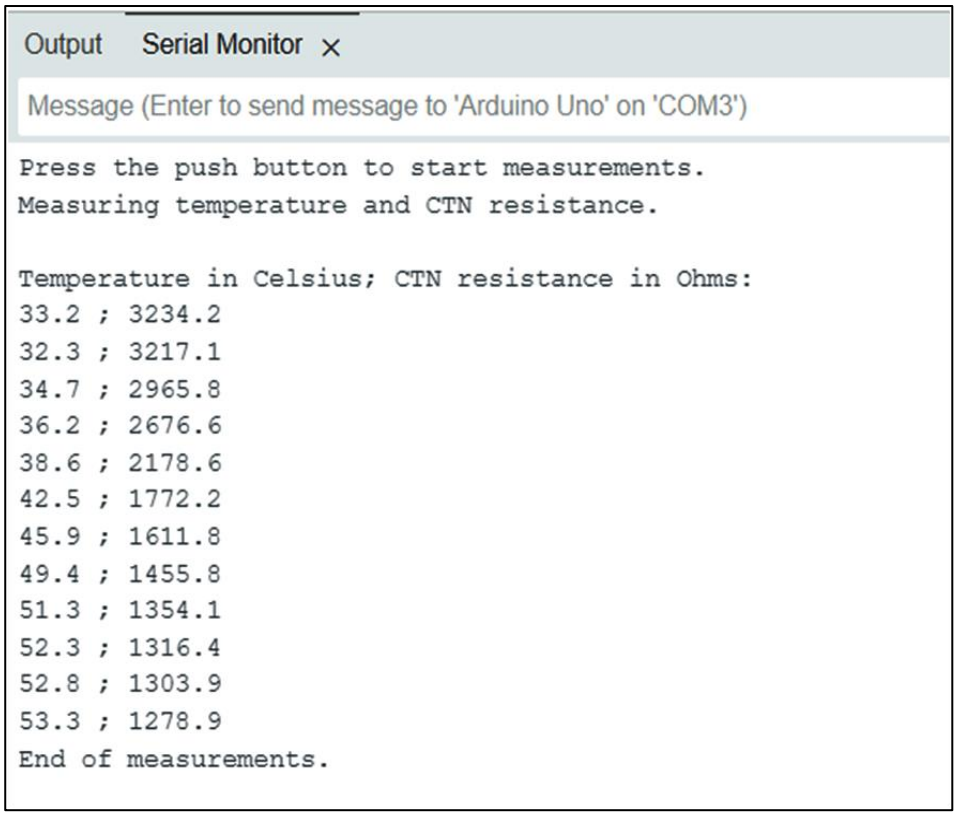

Figure III.15: Results in serial monitor using NTC thermistor and Lm 35 temperature sensor.

III.5.1.5 Exploitation of measures

By incrementally raising the temperature of hot air, both the temperature (*T*) measured by the LM 35 sensor and the resistance (R_t) of the NTC rise. Next, we proceed to map the relationship between the characteristic *Rt* and T. Using Arduino, we can determine the temperature for any value within the experiment's temperature range by measuring the resistance of the NTC (negative temperature coefficient).

$$\frac{R_T}{R_0} = e^{(\beta \times \left(\frac{1}{T} - \frac{1}{T_0}\right))} \quad \text{(III.6)}$$

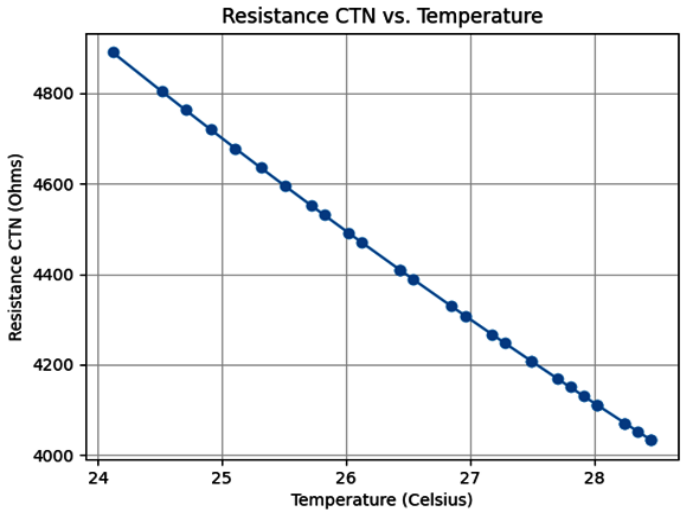

Figure III.16: The temperature (T) given by the LM35 sensor and the resistance (Rt) of the CTN.

From the graph, one of the two characteristic sizes of the CTN is determined:

at T_0 = 298.15 K , R_0= 4.7 kΩ and β= 3977 K

In the temperature range of the experiment, the simplified Steinhart-Hart relationship is verified. We then deduce the expression of the temperature (°K) based on the resistance of the CTN (Ohms):

$$R_T = R_0 \times e^{\beta(\frac{1}{T}-\frac{1}{T_0})} \quad \text{(III.7)}$$

$$\ln(R_T) = \ln(R_0 \times e^{\beta(\frac{1}{T}-\frac{1}{T_0})}) \quad \text{(III.8)}$$

$$\ln(R_T) = \ln(R_0) + \beta(\frac{1}{T}-\frac{1}{T_0}) \quad \text{(III.9)}$$

$$\beta\left(\frac{1}{T}-\frac{1}{T_0}\right) = \ln(R_T) - \ln(R_0) = \ln\left(\frac{R_T}{R_0}\right) \quad \text{(III.10)}$$

$$\frac{1}{T} = \frac{1}{\beta} \times \ln\left(\frac{R_T}{R_0}\right) + \frac{1}{T_0} \quad \text{(III.11)}$$

III.6 Temperature measurement with a NTC thermistance

To measure a temperature with a thermistance NTC, it is necessary to know its characteristic magnitude. Most often, the manufacturer provided the following values:

- The value of its resistance R_0 (nominal resistance in Ω) at the reference temperature T_0 = 25 °C (298.15 K).
- The value of β (en K)

- The temperature range in which the correlation between temperature T (measured in Kelvin) and R_T, the resistance (measured in ohms) of the NTC at that temperature, is confirmed:

$$\frac{1}{T} = \frac{1}{\beta}\ln\left(\frac{R_T}{R_0}\right) + \frac{1}{T_0} \quad \text{(III.12)}$$

It follows:

$$T = \frac{1}{\frac{1}{\beta}\ln\left(\frac{R_T}{R_0}\right)+\frac{1}{T_0}} - 273.15 \quad \text{(III.13)}$$

Without these distinctive values, calibration is necessary (refer to the preceding task) to empirically ascertain them. The objective of the work is to develop a universal program that can accurately measure temperature using any NTC thermistor with a known characteristic enormity.

III.6.1 Programme

When initializing the program in Python or Arduino, the activity code necessitates inputting the values of T_0, R_0, and β. This is done to enable the calculation of temperature based on the resistance measurement of the NTC.

III.6.1.1 Python Program

```
import pyfirmata
import time
import math
import matplotlib.pyplot as plt
# Arduino board setup
board = pyfirmata.Arduino('COM3')  # Change '/dev/ttyACM0' to the appropriate port
pin_ctn = board.get_pin('a:5:i')
pin_button = board.get_pin('d:5:i')
it = pyfirmata.util.Iterator(board)
it.start()
# Constants and variables
temp_ref = 0
ro = 0
b = 0
measuring = False
# Lists to store resistance and temperature data for plotting
resistance_data = []
temperature_data = []
# Initialization
def setup():
    global temp_ref, ro, b
    print("Please enter the reference temperature (value between 1 and 100 degrees Celsius):")
    while temp_ref < 1 or temp_ref > 100:
        temp_ref = int(input())
    print("Reference temperature (degrees Celsius):", temp_ref)
```

```
        print("")
        print("Please enter the nominal resistance (value between 1 and 200000 ohms):")
        while ro < 1 or ro > 200000:
            ro = int(input())
        print("Nominal resistance (ohms):", ro)
        print("")
        print("Please enter the constant B (value between 1 and 10000 K):")
        while b < 1 or b > 10000:
            b = int(input())
        print("Constant B (K):", b)
        print("")
        print("Press the push button to start/stop measurements.")
    # Main loop
    def loop():
        global measuring
        val_button = pin_button.read()
        time.sleep(0.01)
        if val_button == 1:
            if not measuring:
                print("Starting temperature measurement.")
                measuring = True
            else:
                print("Stopping temperature measurement.")
                measuring = False
            time.sleep(0.5)  # Adding a small delay after button press to avoid immediate toggling
        if measuring and len(temperature_data) < 30:
            val_ctn = pin_ctn.read() * 1023
            vctn = (val_ctn / 1023.0) * 5.0
            rt = 10000 * vctn / (5 - vctn)
            temp = 1.0 / (math.log(rt / ro) / b + 1 / (temp_ref + 273.15)) - 273.15
            resistance_data.append(rt)
            temperature_data.append(temp)
            print("Resistance NTC (Ohms): {:.1f}, Temperature (Celsius): {:.1f}".format(rt, temp))
            # Plotting the graph
            plt.plot(temperature_data, resistance_data, marker='o', linestyle='-')
            plt.xlabel('Temperature (Celsius)')
            plt.ylabel('Resistance NTC (Ohms)')
            plt.title('Resistance NTC vs. Temperature')
            plt.grid(True)
            plt.show()
            time.sleep(0.1)
    if __name__ == "__main__":
        setup()
        while len(temperature_data) < 30:
            loop()
```

III.6.1.2 Results in the Python Shell window:

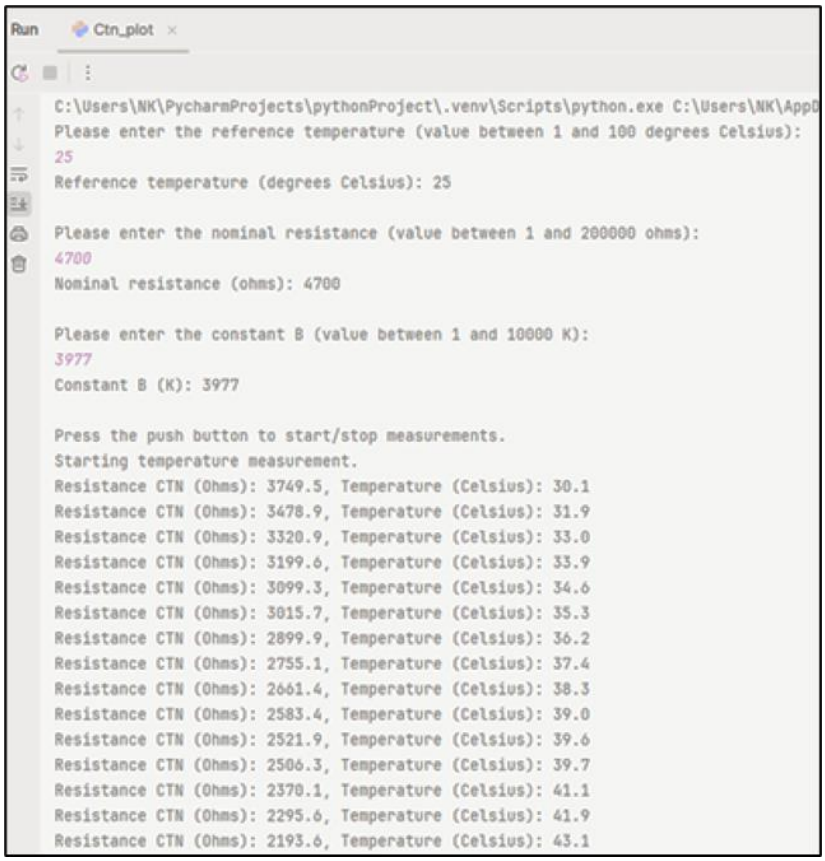

Figure III.17: Results in the Python Shell window using NTC thermistor

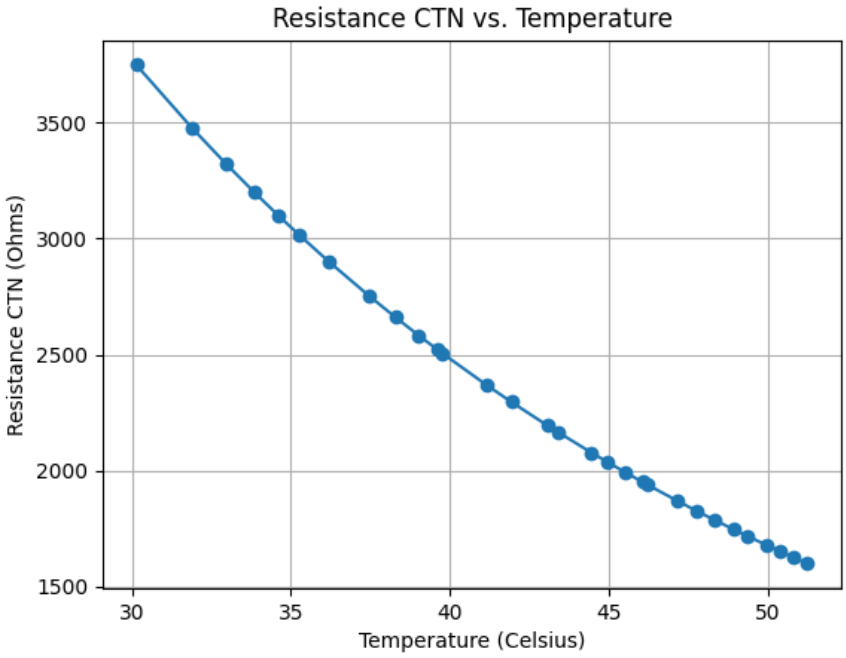

Figure III.18: Resistance CTN vs. temperature.

III.6.1.3 Program in Arduino language

```
// Including libraries
#include <math.h>
// Declaration of constants and variables
const int PinCTN = 5;
const int PinButton = 5;
int ValCTN = 0;
float Temp = 0.0;
float OldTemp = 0.0;
float Rt = 0.0;
float Vctn = 0.0;
int TempRef = 0;
long Ro = 0;
int B = 0;
int ValButton = 0;
int OldValButton = 0;
int State = 0;
int OldState = 0;
// Initialization of inputs and outputs
void setup() {
  Serial.begin(9600);
  pinMode(PinButton, INPUT);
  Serial.print("Please enter the reference temperature ");
  Serial.println("(value between 1 and 100 degrees Celsius).");
   while(TempRef < 1 || TempRef > 100)
  {
    TempRef = Serial.parseInt();
  }
  Serial.print("Reference temperature (degrees Celsius): ");
  Serial.println(TempRef);
  Serial.println("");
  Serial.print("Please enter the nominal resistance value ");
  Serial.println("(value between 1 and 200000 ohms).");
   while(Ro < 1 || Ro > 200000)
  {
    Ro = Serial.parseInt();
  }
  Serial.print("Nominal resistance (ohms): ");
  Serial.println(Ro);
  Serial.println("");
  Serial.print("Please enter the B constant ");
  Serial.println("(value between 1 and 10000 K).");
   while(B < 1 || B > 10000)
  {
    B = Serial.parseInt();
  }
  Serial.print("B constant (K): ");
  Serial.println(B);
  Serial.println("");
  Serial.println("Press the push button to start measurements.");
```

```
}
// Main loop function
void loop() {
  ValButton = digitalRead(PinButton);
  delay(10);
  if ((ValButton == HIGH) && (OldValButton == LOW)) {
    State = 1 - State;
  }
  OldValButton = ValButton;
  if (State == 1) {
    if (OldState == 0) {
      Serial.println("Measuring temperature.");
      Serial.println("");
      Serial.println("NTC resistance in Ohms; Temperature in degrees Celsius:");
      OldState = 1;
    }
    ValCTN = analogRead(PinCTN);
    Vctn = (ValCTN / 1023.0) * 5.0;
    Rt = 10000 * Vctn / (5 - Vctn);
    Temp = 1.0 / (log(Rt / Ro) / B + 1 / (TempRef + 273.15)) - 273.15;
    if (OldTemp != Temp) {
      Serial.print(Rt, 1);
      Serial.print(" ; ");
      Serial.println(Temp, 1);
      OldTemp = Temp;
    }
    delay(100);
  } else {
    if (OldState == 1) {
      Serial.println("End of measurements.");
      OldState = 0;
    }
  }
}
```

III.6.1.4 Results in serial monitor:

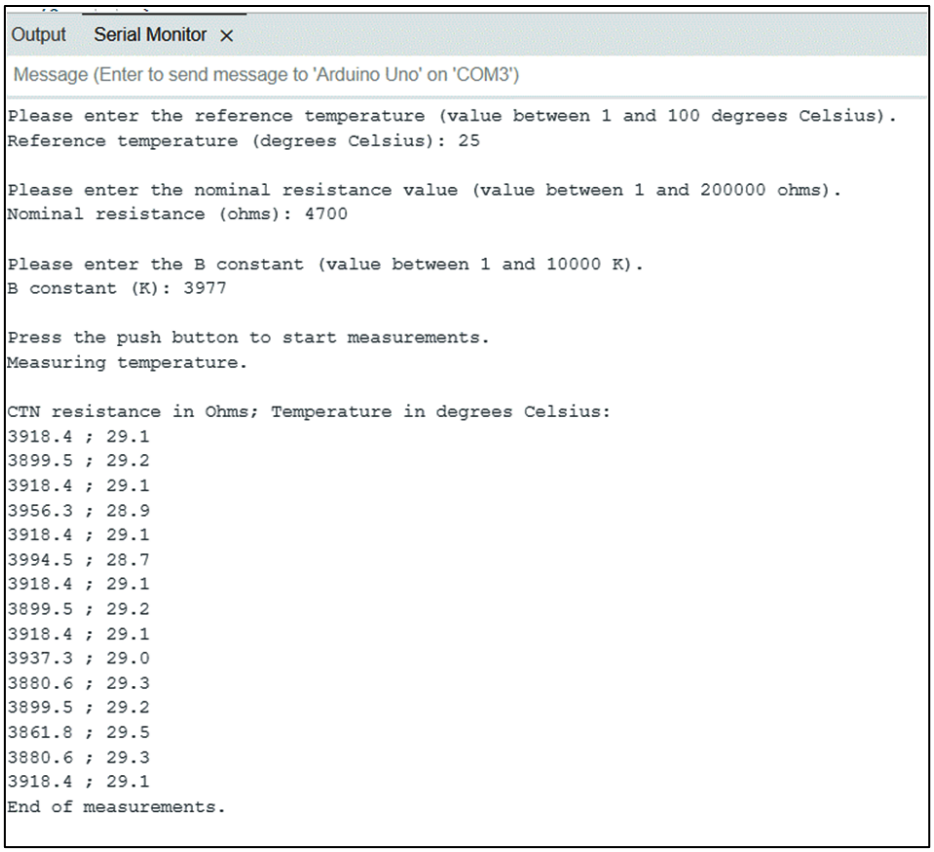

Figure III.19: Results in serial monitor-Resistance CTN vs. temperature.

III.7 Conclusion

This chapter provides a thorough analysis of the use of Arduino for temperature measurement and related applications, highlighting the benefits of various temperature sensors such as the LM 35, and NTC thermistors. It guides users through the process of creating circuits, writing code in both Python and Arduino, and taking measurements through a series of in-depth exercises. uses tasks to demonstrate the process of configuring temperature measuring circuits, setting up the Arduino to read and interpret sensor data, and setting up visual and audible warnings for temperature thresholds. We also cover further advanced subjects, such as sensor calibration using the Steinhart-Hart equation to enhance temperature measurement accuracy.

The thorough method, which comes with component lists, circuit schematics, step-by-step instructions, and programming in Python and Arduino

General conclusion

General conclusion

The work that we carried out as part of this dissertation allowed us to study certain phenomena which govern the operation of temperature sensors based on LM35 and NTC, which is currently attracting a lot of attention, due to its emergence as an alternative sensor to the other conventional sensors. Provide a thorough opportunity to temperature measurement, the Firmata communication protocol, and the Arduino platform, all of which are essential for understand and carrying out temperature sensing projects.

A variety of sensors can detect and transform the average energy of a substance's particles into electrical signals by measuring its temperature. This chapter introduces and describes the principles of operation of several temperature sensors, such as thermistors and the LM35 sensor. It addresses several temperature scales, such as Fahrenheit and Celsius, and highlights the significance of precise temperature measurement in a number of sectors, including quality control, climatology, and medicine.

In order to optimize the the physical characteristics of the sensors used, the simulation results obtained showed that:

- The Arduino platform an open-source electronics platform that combines software and hardware is examined in detail. This platform lets users design interactive electronic creations. Arduino's adaptability and user-friendliness make it suitable for a wide range of tasks, from industrial prototypes to instructional tools. We also introduces the Firmata protocol, which facilitates communication between microcontrollers and host computers, enabling hardware control through high-level programming languages such as Python. Firmata Express, a more user-friendly and efficient variant of Firmata, is especially useful for rapid prototyping and teaching environments.
- The technical and theoretical information to use in real-world scenarios. It focuses on measuring temperature with an Arduino and sensors such as thermistors and LM35. This provides comprehensive instructions for reading temperature data, writing software, and configuring hardware to control devices based on these measurements.
- The use of the Python "pyfirmata" module for communication with Arduino, enabling tasks such as temperature monitoring and alarm activation upon reaching predetermined limits. The practical exercises demonstrate the application of previously covered principles in real-world scenarios, enhancing the reader's understanding of temperature measurement and Arduino programming.

- Measuring and using temperatures is an essential aspect in many fields, from meteorology to sensor technology. Thanks to tools such as: Temperature sensors like the LM35 and Thermistors. We can obtain accurate and reliable data on temperature variations. This information is crucial for environmental monitoring, industrial process control, energy management and many other applications.

Several perspectives can be put forward at the end of this study, such as: By effectively leveraging this data, we can make informed decisions, optimize system performance and help solve many climate and environmental challenges.

. This is certainly an avenue for future research.

Bibliography

[1] Meijer, G.C.M., Smart temperature sensors and temperature-sensor systems. Wiley, Chichester, pp. 151e183, 2008.

[2] Michalski, L., Eckersdorf, K., Kucharski, J., McGhee, J., Temperature Measurement, 2nd ed. Wiley, Chichester 2001.

[3] Taymanov, R., Sapozhnikova, K., What makes sensor devices and Microsystems 'intelligent' or 'smart', Smart Sensors and MEMS, 2nd ed. Elsevier, Oxford. 2017.

[4] George Asche et Coll, les capteurs en instrumentation industrielle, 6iéme édition, Dunod 2006.

[5] Abozaid A., Tsang B., and R. Gerlai, "The effects of small but abrupt change in temperature on the behavior of larval zebrafish," *Physiol. Behav.*, vol. 227, p. 113169, Dec. 2020,

[6] Balmer, Robert T., Modern Engineering Thermodynamics. Academic Press. p. 9. ISBN 978-0-12-374996-3. Retrieved 17 July 2011.

[7] "Fahrenheit temperature scale". Encyclopædia Britannica Online. Retrieved 25 September 2015.

[8] "Kelvin: Introduction". NIST. 2018-05-14. Retrieved 2022-09-02.

[9] BIPM (2019-05-20). "Mise en pratique for the definition of the kelvin in the SI". BIPM.org. Retrieved 2022-02-18.

[10] "NTC Thermistors" Archived 2017-09-22 at the Wayback Machine. Micro-chip Technologies. 2010.

[11] Knoll, G.F. (1999). Radiation Detection and Measurement (3rd ed.). Wiley. p. 365. ISBN 978-0-471-07338-3.

[12] "Thermocouple temperature sensors". Temperatures.com. Archived from the original on 2008-02-16. Retrieved 2007-11-04.

[13] "Technical Notes: Thermocouple Accuracy". IEC 584-2(1982)+A1(1989). Retrieved 2010-04-28.

[14] Ramsden, Ed (September 1, 2000). "Temperature measurement". Sensors. Archived from the original on 2010-03-22. Retrieved 2010-02-19.

[15] Manual on the Use of Thermocouples in Temperature Measurement (4th Ed.). ASTM. 1993. pp. 48–51. ISBN 978-0-8031-1466-1. Archived from the original on 2013-08-14. Retrieved 2012-09-04.

[16] Seebeck. "Magnetische Polarisation der Metalle und Erze durch Temperatur-Differenz" [Magnetic polarization of metals and ores by temperature differences]. Abhandlungen der Königlichen Akademie der Wissenschaften zu Berlin (in German): 265–373, (1992).

[17] "Deciphering Radiation Alarms: Using High Purity Germanium Detectors for Nuclear Security". www.iaea.org. 18 December 2020. Retrieved 6 May 2024.

[18] Vatansever, Fatma; Hamblin, Michael R., "Far infrared radiation (FIR): Its biological effects and medical applications". Photonics & Lasers in Medicine ,2012. ISSN 2193-0643, 2012.

[19] http://www.generationrobots.com/fr/152-arduino.

[20] Biggs, John. "CEO controversy mars Arduino's open future". *TechCrunch*. Retrieved 2017.

[21] "Optiboot Bootloader for Arduino and Atmel AVR". *GitHub*. Retrieved 2015.

[22] S.V.D.Reyvanth, G.Shirish, « PID controller using Arduino ».

[23] C. Tavernier, « Arduino applications avancées ». Version Dunod.

[24] https://industrial-items.blogspot.com/2014/11/resistors-resistivity-color-coding-of.html.

[25] http://www.acm.uiuc.edu/sigbot/tutorials/ 2009-11-17-arduino-basics

[26] Jean- Noël, « livret Arduino en français », centre de ressources art sensitif.

[27] https://www.arduino.cc/reference/en/.

[28] Paul Deitel, Python for Programmers, Copyright © 2019 Pearson Education, Inc.

[29] https://github.com/firmata/arduino.

Abstract

This study explores the implementation and utilization of Arduino, an open-source electronics platform, alongside the Firmata communication protocol. Provides a thorough examination of temperature measurement, elucidating the underlying principles and distinct temperature scales. It also presents sensors such as thermistors and the LM35 temperature sensor, highlighting their use in diverse applications. Explores the Arduino platform, focusing on its hardware and software components, adaptability, and the support it receives from the community. This text explores the importance of the Firmata protocol in enabling communication between microcontrollers and host computers. The text specifically highlights the use of the simpler Firmata Express version, specifically designed for educational and prototyping purposes. The practical implementation of Arduino and Firmata is illustrated by the measurement of temperature using sensors such the LM35. The guide provides comprehensive instructions for configuring the hardware and coding the Arduino using Python and the pyFirmata module. This research highlights the efficacy of Arduino and Firmata in developing adaptable and efficient solutions for diverse electronic tasks.

Keys words: temperature sensors : LM35, CTN, Arduino, Python, firmata

Résume

Cette étude explore la mise en œuvre et l'utilisation d'Arduino, une plateforme électronique open-source, aux côtés du protocole de communication Firmata. Le chapitre 1 fournit un examen approfondi de la mesure de la température, en élucidant les principes sous-jacents et les échelles de température distinctes. Il présente également des capteurs tels que les thermistances et le capteur de température LM35, mettant en évidence leur utilisation dans diverses applications. Explore la plateforme Arduino, en se concentrant sur ses composants matériels et logiciels, l'adaptabilité et le soutien qu'elle reçoit de la communauté. Ce texte explore l'importance du protocole Firmata dans la communication entre les microcontrôleurs et les ordinateurs hôtes. Le texte souligne spécifiquement l'utilisation de la version Firmata Express plus simple, spécialement conçue à des fins éducatives et de prototypage. Dans le chapitre 3, la mise en œuvre pratique d'Arduino et Firmata est illustrée par la mesure de la température à l'aide de capteurs tels que le LM35. Le guide fournit des instructions complètes pour configurer le matériel et coder l'Arduino en utilisant Python et le module pyFirmata. Cette recherche met en évidence l'efficacité d'Arduino et Firmata dans le développement de solutions adaptables et efficaces pour diverses tâches électroniques.

Mots clefs : Capteurs de temperature, LM35, CTN, Arduino, Python, firmata.

الملخص

تستكشف هذه الدراسة تنفيذ واستخدام Arduinoبروتوكول Firmata ، وهي منصة إلكترونيات مفتوحة المصدر، جنبًا إلى جنب مع . كما يقدم أجهزة استشعار مثل مستشعرات للاتصالات. لقياس درجة الحرارة، مما يوضح المبادئ الأساسية ومقاييس درجة الحرارة المتميزة ، مع التركيز على مكونات Arduino ، مما يسلط الضوء على استخدامها في تطبيقات متنوعة. LM35 الحرارة ومستشعر درجة الحرارة في تمكين الاتصال بين Firmata أجهزتها وبرامجها، والقدرة على التكيف، والدعم الذي تتلقاه من المجتمع. يستكشف هذا النص أهمية بروتوكول الأبسط، Firmata Express أجهزة التحكم الدقيقة وأجهزة الكمبيوتر المضيفة. يسلط النص الضوء على وجه التحديد على استخدام إصدار من خلال قياس درجة الحرارة باستخدام أجهزة Firmata و Arduino المصمم خصيصًا للأغراض التعليمية والنماذج الأولية.التنفيذ العملي لـ يسلط هذا البحث .pyFirmata ووحدة Python باستخدام Arduino يوفر الدليل تعليمات شاملة لتكوين الأجهزة وترميز .LM35 الاستشعار مثل في تطوير حلول قابلة للتكيف وفعالة لمهام إلكترونية متنوعة Firmata و lmArduinoالضوء على فعالية .

كلمات مفتاحية : حساس الحرارة lm35 ,Python Arduino,CTN, firmata