Escola Tècnica Superior d'Enginyeries
Industrial i Aeronàutica de Terrassa
UNIVERSITAT POLITÈCNICA DE CATALUNYA

Titulació:

GRETA

Alumne:

Carles Molins Duran

Títol TFG:

Study of a Low Cost System for Measurement and Recording of Temperature and its Applications

Director/a del TFG:

David Gonzalez i Manel Sòria

Convocatòria de lliurament del TFG:

9/01/2015

Contingut d'aquest volum:

DOCUMENT 1.- MEMÒRIA

Agraïments:

A Manel Sòria per haver guiat aquest projecte i haver posat motivació, temps i diners a aquest treball; al Doctor Xavier Ramis del Departament de Màquines i Motors tèrmics de la UPC per l'ajuda prestada tant pel que fa a metodologia com material en l'assaig dels sensors de contacte; a Dicrom per la donació de varis sensors Melexis que varen ser utilitzats per a l'estudi; a NesLab per fer possible el llançament de l'aplicació dedicada a globus meteorològics; als meus companys de carrera, sense els quals no seria on soc; i a la meva família i parella per donar-me suport durant tots aquests mesos. Gràcies a tots.

INDEX

INDEX OF FIGURES

INDEX OF TABLES

1. INTRODUCTION

Temperature measures the average kinetic energy of the particles per mass unit. There is a wide variety of methods in order to measure temperature which varies depending on the price and precision of the equipment. The most common electronic-based sensors are thermocouples, resistance temperature detectors (RTDs), thermistors and IR thermometers. However, sensors alone are not enough in order to make measurements; the original electric signal produced by the sensors must be amplified, filtered (if necessary), converted to digital format and stored.

Usually temperature measurement systems are expensive, especially those of industrial or laboratory use which need a high precision, and can easily cost more than 5000€ (see section 5.2.1. BENCH SYSTEMS). These systems are also bulky and difficult to transport but present a great flexibility and reliability.

On the other hand, low-cost portable devices have also been developed. On this side of the market, handheld devices are dominant. However, they are simple and cannot adapt to every request, and most of them do not have data logging and/or multiple input capabilities. Portable sensor-computer interfaces can also be found but they are not stand-alone, requiring the presence of the computer in order to take the measurements.

Low-cost electronic sensors and the Arduino platform as the signal processor are presented and studied as a low-cost and highly flexible alternative to the commercial systems. With this set up, the user gets a portable, low-cost system, which is stand-alone and greatly flexible. After different sensors are studied and calibrated, two applications will be developed with one or more of the sensors.

2. AIM

Present a low-cost alternative to commercial temperature measurement systems, study its capabilities and develop two applications for the aeronautical industry, one module for drones which does a thermal map of a terrain and one module for atmospheric balloons in order to carry out various temperature scans.

3. SCOPE

-Study of the state of the art in temperature measurements and registering systems.

- Election of different low-cost commercial temperature sensors which will be studied.

-Development of a temperature measuring system in order to assay all the selected sensors for the study, including their wiring and software.

-Analysis of the low-cost sensing components.

-Calibration of the low-cost sensing components.

-Development of two applications using one or more of the sensing components studied previously, including wiring and software.

-Assay of the developed applications in a real environment and analysis of their results.

-Economical and environmental aspects of the applications.

-Conclusions.

4. REQUIREMENTS

-Low-cost. The components studied and the final application must be low-cost relative to existing temperature measurement systems (under 200€).

-Reliable and precise. Depending of the nature of the component elected for the application, a maximum of ±0,75ºC error will be accepted for contact sensors and a maximum of ±2ºC for IC sensors.

-Sufficient autonomy. The device will have to have sufficient autonomy so that it lasts enough for it to be functional and practical.

-Self standing. The final application must be capable of carrying out all the measurements and data logging without any external support such as a computer (the post processing, however, will be carried away with a computer).

5. STATE OF THE ART

Temperature, which is one of the most commonly measured variables, measures the average kinetic energy of the particles per mass unit. There is a wide variety of methods in order to obtain temperature, which varies depending on the price and precision of the equipment and the nature of the object whose temperature is to be measured. The most common electronic-based sensors are thermocouples, resistance temperature detectors (RTDs), thermistors and IR sensors. However, sensors alone are not enough in order to make measurements; the signal produced by the sensors must be processed afterwards in order to have a reliable and precise signal.

5.1. SENSORS

The following pages are dedicated to present and explain the different type of electronic sensors that are commonly used.

5.1.1. THERMOCOUPLES

Thermocouples are widely used in temperature measurement as they are economic and can operate over a wide temperature range. They also have good accuracy, a fast thermal response, high reliability and ruggedness. These characteristics make thermocouples have a great versatility of applications.

Essentially, a thermocouple thermometer is a system consisting of a temperature-sensing element called a thermocouple, which produces an electromotive force (emf) that varies with temperature; and an electrical conductor (extension wires) for connecting the thermocouple to the sensing device. Although any combination of two dissimilar metals will generate a thermal emf, only seven thermocouples are of common use in industry today, due to their mechanical and chemical properties, stability of emf and cost. Thermocouples are designated by means of capital letters, which indicate which metals compose the thermocouple, according to the American National Standards Institute (ANSI).

Type	Positive Metal/Alloy	Negative Metal/Alloy	Temperature Range (°C)
T	Copper	Constantan	-200 to +350
J	Iron	Constantan	0 to +750
K	Chromel	Alumel	-200 to +1250
E	Chromel	Constantan	-200 to +900

Table 1: Basic thermocouple characteristics.

The physical principle under which they operate[1], discovered in 1821 by Seebeck, is based on the electromotive force which is generated if two dissimilar metals are joined in a closed circuit, maintaining the junctions at different temperatures. This thermal emf was called Seebeck emf, in honor of its discoverer.

Figure 1 (a) is a schematic diagram of two electrical conductors, A and B, whose two junctions are exposed to different temperatures, T_1 and T_2. The thermal efm generated in this circuit, E_{AB}, is expressed:

$$E_{AB} = f[A, B, (T_2 - T_1)]$$

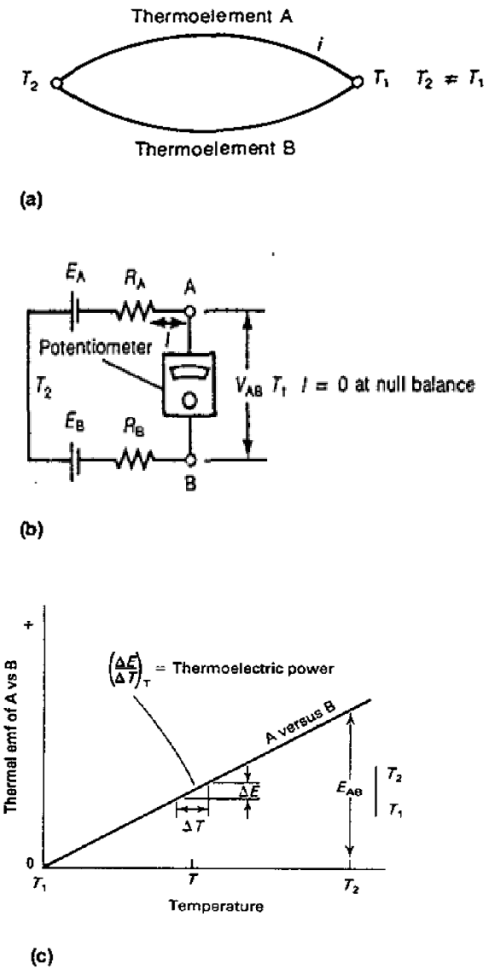


Figure 1: Schematic (a), equivalent circuit in a potentiometer circuit (b) and efm (c) of a typical thermocouple system. Adapted from [1].

In Figure 1 (b) the thermoelements are represented by a battery (E_A and E_B), which is the emf output of each thermoelement with reference to a certain standard; and a resistance (R_A and R_B), which is the resistance of the thermoelements. A bucking voltage is applied at the potentiometer until it is equal in magnitude and opposite in direction to E_{AB}. At null balance there si no current flow, so:

$$V_{AB} = E_A - E_B$$

Then, the measured emf at the potentiometer V_{AB} is the thermal emf of the thermocouple AB.

As seen in Figure 1 (c), the emf of AB thermocouple varies linearly with temperature and can be expressed as:

$$\Delta V = S \cdot \Delta T$$

Where ΔV is the voltage increment, ΔT is the temperature increment and S is the thermoelectric power or Seebeck coefficient.

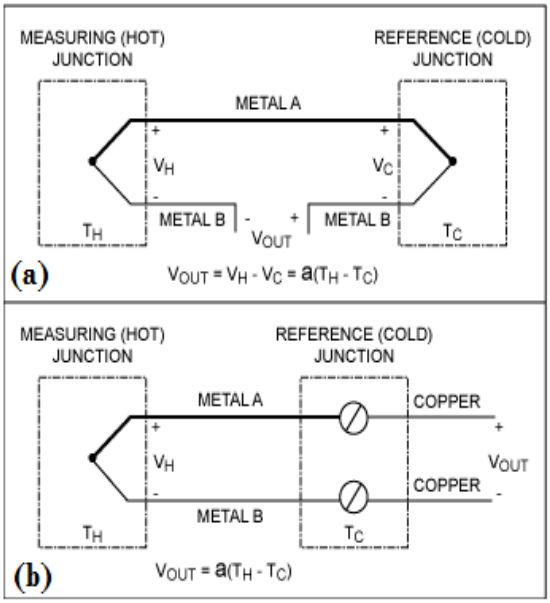


Figure 2: Closed circuit thermocouple (a) and thermocouple attached to two wires (b). Adapted from [2].

However, the closed circuit which can be seen in Figure 2 (a) is not the most commonly used thermocouple configuration [2]. For most applications, Figure 2 (b) configuration is used. This configuration introduces a third metal (known as intermediate metal) into the loop and two additional junctions. As long as these two junctions are at the same temperature, the intermediate metal (in this case copper) has no effect on the output voltage. This configuration allows the thermocouple to be used without a separate reference junction.

In both cases though, since the thermocouple measures temperature differentially, the cold-junction temperature must be known in order to determine the actual temperature measured at the hot junction, process known as cold-junction compensation (CJC)[3]. This temperature can be obtained with any of the other sensors which will be presented in this work. While this fact may induce to doubt about the usefulness of thermocouples, it has to be taken into account that the thermocouple hot junction can be exposed to extreme environments while the reference junction is under an ordinary environment.

It has to be taken into account too that the signal produced by thermocouples is low (in the order of micro volts) which makes necessary an amplifier circuit.

5.1.2. RESISTANCE TEMPERATURE DETECTORS

Resistance Temperature detectors (RTDs) are the most accurate, stable and repeatable of the sensors which will be discussed in this work, although they are the most expensive and fragile too[4]. They also need an external power supply. The physical principle under which they operate is based on the changes in the

electric resistance of pure metals when submitted under variation of temperature [5].

The main three designs of RTD elements are the following:

- Thin film elements, which consist of a thin layer (normally between 10 and 100 angstroms) of resistive material on a ceramic substrate and then coated with an epoxy or glass coating. They are not as stable as other RTD elements and can only be used in a limited temperature range due to the different expansion rates of the substrate and the resistor, which may cause mechanical strains which induce error.
- Wire-wound elements are formed by a wire wrapped around a cylindrical-insulating core forming a coil. The coil diameter provides a compromise between mechanical stability and allowing expansion of the wire to minimize strain and consequential drift. They can have greater accuracy, especially for wide temperature ranges.
- Coiled elements have a coil which can expand freely over temperature, held in place by some mechanical support which lets the coil keep its shape. This leads to the most accurate and with less error due to mechanical strains of the designs.

By making a current go through the RTD, a voltage is generated which can be measured in order to obtain the temperature. The voltage-temperature relation they present is the most linear (see Figure 3) compared to other temperature sensors. This relation can be expressed with the Callendar Van-Dusen equation:

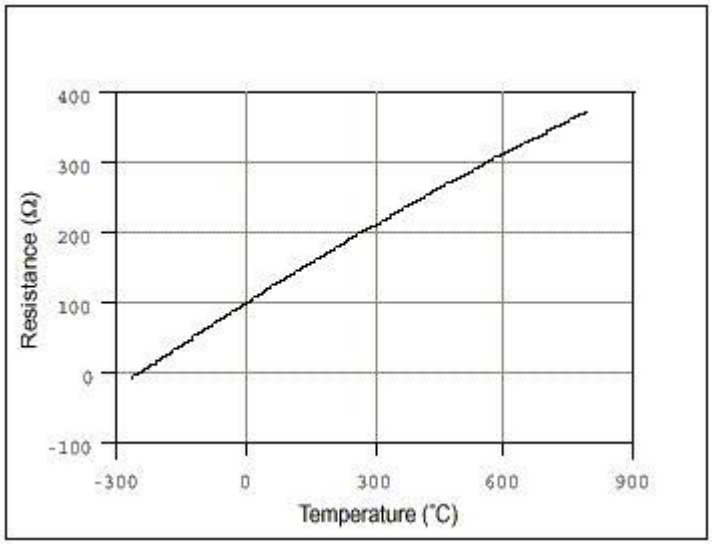


Figure 3: Resistance-temperature curve for a 100 ohm platinum RTD. Adapted from [5].

$$R_T = R_O\left(1 + a \cdot T + b \cdot T_2 + c \cdot T_3 \cdot (T - 100)\right) \qquad for\ T < 0^O C$$

$$R_T = R_O(1 + a \cdot T + b \cdot T_2) \qquad for\ T > 0^O C$$

Where R_T is the resistance at temperature T, R_O is the nominal resistance (resistance at $T = 0$) and a, b and c are constants in order to calibrate the sensor.

The main downsides of this sensors is its cost (due to the fact that they use platinum and its manufacture is relatively expensive), they have a slow response time and low sensitivity. They cannot be used in high temperature applications as it is increasingly difficult to prevent the platinum from becoming contaminated by impurities. Also, they are typically bigger than thermocouples.

5.1.3. THERMISTORS

Thermistors are unfairly regarded as inaccurate sensors due to the fact that in the past had 5% tolerances at its best, but modern thermistors can be highly accurate (about 0.1ºC) at reasonable costs. They have a quick response and a greater sensibility than RTDs [4]. Similar to RTDs, thermistors exploit the principle of material's resistance variation with respect to temperature variation, but instead of using pure metals, ceramic, polymer or oxide semiconductors are used [5].

As a first order approximation it can be assumed that the relationship between resistance and temperature is linear:

$$\Delta R = k \cdot \Delta T$$

Where ΔR is the change in resistance, ΔT is the change in temperature and k is a first-order temperature coefficient of resistance. Depending of the value of k , if it is positive the thermistors is called positive temperature coefficient (PTC) or posistor while if it is negative the device is called negative temperature coefficient (NTC).

For accurate measurements over a wider range of temperatures, the linear approximation is not enough and a more detailed description of the curve must be used. The Steinhart-Hart equation is a third order approximation widely used [6]:

$$\frac{1}{T} = a + b \cdot \ln(R) + c \cdot (\ln(R))^3$$

Where T is the temperature, R is the resistance and a, b and c are called the Steinhart-Hart parameters which must be specified through calibration for each device.

The main disadvantages of thermistors are its non-linearality, the fact that they are more susceptible to permanent decalibration at high temperatures and that they are especially susceptible to self-heating errors.

5.1.4. IC SENSORS

The IC sensors (integrated circuit sensors) are an innovation in thermometry [6]. They supply an output which is linearly proportional to absolute temperature and they can be obtained in both voltage and current output configurations. Some of them even have a built in analog-to-digital (A/D) converter which means that its output can be directly read by a microcontroller.

Except for its linear response, they share most of the strengths and disadvantages of thermistors: they have a limited temperature range, they suffer of self heating and can be fragile. Also, being an IC, they require of an external power source.

5.1.5. INFRARED THERMOMETERS

Infrared thermometers (or IR) differ from the other sensors above in the fact that is a non-contact sensor. This allows its use where the others cannot be employed such as in cases of moving objects, when contact measurements cannot be done due to hazardous reasons, where distances are too great or when the temperatures to be measured are too high for contact sensors.

IR thermometers measure reflected infrared light of an area. They collect emitted energy from the target which then is amplified and converted into a voltage output. With this data and compensating with the ambient temperature, the target's temperature can be computed if its emissivity is known following the Stefan-Boltzmann law [7]:

$$E = \epsilon \cdot \theta \cdot T^4$$

Where E is the energy received, ϵ is the target's emissivity, θ is the Stefan-Boltzmann constant and T is the temperature.

IRs can be quite accurate but their readings can be affected by several factors [8][9]:

- Distance to spot ratio (D:S ratio) indicates the size of the area measured relative to distance away from the object being measured. It is

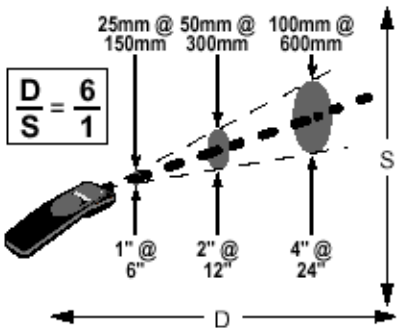


Figure 4: Example of an IR thermometer with a D:S ratio of 6:1.

also described with the Field of View (FOV) parameter. If the target is smaller than the area being measured, the reading will not be accurate as the temperature obtained will not be the one of the target but the average between the target and its surroundings.

- Most organic materials and painted or oxidized surfaces have an emissivity of 0.95, thus most units with fixed emissivity are preset at this value for the emissivity (fixed emissivity devices are usually cheaper). This means that in case of targeting a surface with a different emissivity; the reading will have an error. Emissivity varies depending on the color, reflectivity, shape, etc. Some IR sensors let the user introduce the emissivity of the object to measure for more precise readings.
- Range is another error inducing factor. Even though there is no technical limit to how far away an object can be measured, the farther away from the object the lower the accuracy can be due to interference with air (dust, humidity, etc.). Also, D:S limits the actual useful range.

Material	Emissivity
Asphalt	0.93 to 0.95
Ceramics and brick	0.80 to 0.95
Cloth	0.95
Concrete	0.94 to 0.95
Glass	0.76 to 0.85
Metals, unoxidized	0.02 to 0.21
Painted surfaces	0.74 to 0.96
Paper	0.50 to 0.95
Rubber	0.95
Sand	0.90
Snow	0.82 to 0.89
Soil	0.90 to 0.98
Steel, iron, oxidized	0.65 to 0.95
Steel, stainless	0.10 to 0.80
Water	0.93
Wood	0.89 to 0.94

Table 2: Emissivities for various kinds of surfaces [10].

IR thermometers usage has taken over contact sensors in some fields, especially those where readings errors are not critical. One of their benefits is clear: the fact that they can obtain measures from a distance, which enables temperature readings in situations such as when the object is too hot to touch, difficult to reach or the material could be scratched or contaminated by a standard contact thermometer. Another of its benefits is the how fast the measurements can be taken: while contact sensors need to heat up to the temperature at which the object is, IR thermometers measurements are immediate. These benefits come, as said before, at expenses of higher reading errors, which are typically around 1ºC.

5.2. DATA ACQUISITION

The outputs of the previous sensors are analog signals (except for the IC sensor, which depending of the model can be already digital). In most cases these signals need amplification, they might need noise filters and in the case of thermocouples CJC is compulsory. Also, the signal must be converted from analog to digital in order to be readable by a microprocessor or a computer and, in the later case, drivers and software support is needed.

5.2.1. BENCH SYSTEMS

The temperature sensing systems which are used in industry or research use sensors which are not different in nature to the ones presented above. The principle under which they work is the same; however the cost of the system is way higher than a sensor which could be bought in an electronic-components shop. The higher values of these systems mainly come from the data acquisition sub-system which makes the sensor's data readable to a computer, making possible the post processing. The price might rise too if the system requires a specific software for the post processing itself.

Schematic for a standard thermocouple temperature sensing system is shown in Figure 5.

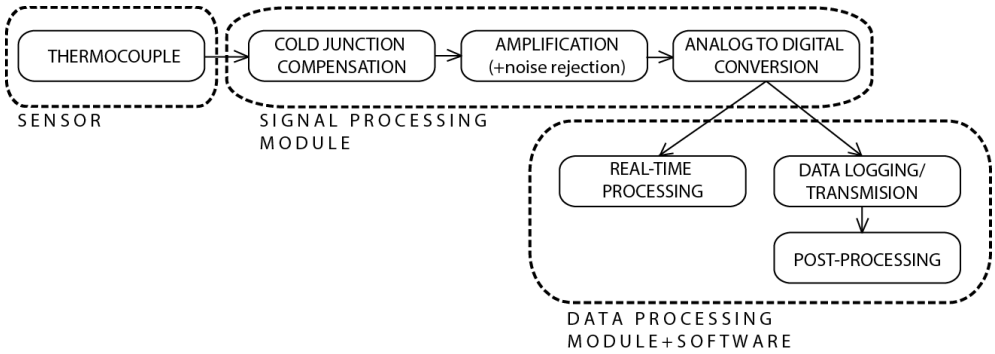


Figure 5: Schematic of a standard thermocouple temperature sensing system.

As an example, if a system such as the one shown above was to be built with National Instruments® components, the items shown in Table 3 would be necessary [3]:

Module	Component	Characteristics	Price
Signal processing	NI 9211	- 4 channels. - +-80mV analogic inputs. - 24bits resolution. - 50Hz noise rejection. - Usable with type J, K, T, E, N, B, R	325€

		and S thermocouples. - -40ºC to 70ºC temperature range. - Operational under 5G vibration and 50G impact.	
Data processing	cDAQ-9132	- Dual-core Intel Atom (1.33 GHz). - 16GB memory, 2GB RAM. - SD storage. - Windows 7 or Linux-RealTime compatible. - 2 USB ports. - 2 Gigabit Ethernet ports.	2930€
Software	LabVIEW	Intuitive, graphical and easy to use system designer with extensive functionality for signal processing, analysis and mathematical modifications.	2920€

Table 3: Components from National Instruments needed for a thermocouple temperature sensing system.

As it can be seen, the price is quite high even though the components chosen are rather cheap relative to other components that the company offers for the same purpose; but the final system is a reliable and flexible platform for temperature sensing which permits easy data analysis right off the shelf. Moreover, the signal processing module and software can be used for a wide variety of other applications.

Figure 6: NI CompactDAQ 9132 system.

However, the user might not need such a versatile system but rather a simpler one; fact that might reduce the price significantly. As it can be seen in Figure 6, another downside of these systems is that they are heavy and bulky, as they are designed for static use; which may make these platforms not useful for some applications which require moving them usually.

5.2.2. PORTABLE SYSTEMS

An alternative to these systems have been developed by various companies in order to attract those customers who are looking for a cheaper and more portable system. In this field there are different approaches to the problem.

5.2.2.1. Handheld Thermometers

One of those approaches is the handheld thermometer systems. These systems are designed to be the paradigm of portable, being small, straight to use, light and ergonomic. They are generally focused in offering a real-time temperature measurement which is displayed through an LCD screen for standalone operation. However, the offer of handheld thermometers is vast and varied: depending of the model they can have multiple or single input channels, they can be used with different types of sensors or they may have a fixed sensor which cannot even be changed, they can have data logging, they may present the temperature data in a temperature versus time graph, etc.

The price of this kind of thermometers is as wide as their variety. For instance, a simple one which displays the temperature of a thermocouple, such as OMEGA®'s HH11B [11] just costs $81,00. OMEGA®'s RDXL-SD series[12] offers up to four inputs with independent displaying and data logging at a price of $320,00. The most complex handheld systems can cost up to $3000, such as OMEGA®'s RDXL 120 series[13], capable of reading 16 thermocouples, RTDs or voltage inputs with real-time graphical display of the measurements versus time and data logging which costs $2945,00.

OMEGA®'s RDXL-SD series Specifications	
Embedded Microcontroller	Custom one-chip LSI device
Display	LCD 82x61mm, backlight green
Channels	4
Sensor compatibilities	Thermocouples type J, K, T, E, R/S and RTD Pt100Ω (European curve)
Resolution	0.1ºC
Sampling time	1 to 3600 seconds
Memory card	1 to 16GB SD
Sampling time of display	Approx 1 second

Table 4: OMEGA®'s RDXL-SD series Specifications from [12].

Figure 7: OMEGA®'s RDXL-SD series (left) and OMEGA®'s RDXL 120 series (right) handheld systems, from [12] and [13] respectively.

In the field of IR thermometers something similar happens. There is a wide variety of IR handheld thermometers, but the most popular ones are those which measure the temperature of a single spot which can be targeted with the aid of a laser beam which points at it. A good example of this kind of model would be the AllQA® Infrared-LC [14], which costs $89,00. More complex and expensive handheld IR thermometers use a thermal camera, allowing the user to take a thermal picture of the targeted area instead of just getting the temperature measurement of one single spot, such as Fluke® VT04A [15], which can be bought for around $800,00 depending on the dealer.

As explained in section **5.1.5. INFRARED THERMOMETERS**, IR thermometers have taken over in some fields. Some of those fields, specially related with handheld devices are listed below:

- Electrical applications: they are used in order to detect hot spots which may induce problems in the electrical net.
- Industrial applications: for steam or refrigerating systems monitoring in order to detect malfunctions.
- Energy management: in order to evaluate heat loss through windows, doors, walls, etc.
- Food safety: IR thermometers are a fast and convenient screening tool for both cold and hot foods for Food Safety and HACCP. No contamination or damage is done to the product and the user can easily take temperature of products moving on conveyors or hard to reach places.

AllQA® Infrared-LC Specifications	
Temperature range	-40 to 100ºC
Accuarcy	+-1ºC
Resolution	0.1ºC
Calibration	Calibrated at 40, 70 and 130ºF
D:S ratio	10:1
Display	LCD with backlight

Table 5: AllQA® Infrared-LC Specifications from [14].

Figure 8: AllQA® Infrared-LC (left) and Fluke® VT04A (right), from [14] and [15] respectively.

5.2.2.2. Portable Data Acquisition Module as Interface with a Computer

Some companies have designed devices which act as a bridge between the sensor and the computer as their solution to the need of a portable system. Similarly to bench systems, these portable systems make sensor data readable to a computer (they act as a signal processing module), with which they communicate through USB connection, Ethernet or WiFi. However, compared to bench systems, these are more limited in capabilities and flexibility. Another downside is the fact that it needs to be attached to a computer in order to carry out its function, which makes it less portable compared to handheld thermometers.

One example of this kind of devices would be National instruments® NI USB-TC01 [16], a simple 1-channel, plug-and-play, thermocouple reader (compatible with J, K, R, S, T, N, E and B type thermocouples), which can be obtained at a cost of 119€. A more complete system would be National instruments®' NI 9215 [17], a 4-input device with USB, Ethernet and WiFi transmission options at a price of 765€ (+70€ or +100€ if Ethernet or WiFi respectively is desired).

5.2.2.3. Other Portable Devices

In the field of IR devices, specialized IR cameras for aerial photography and military purposes must be taken into account. Even though these are not portable as it was defined in the previous sections, these devices are designed to be mounted on high mobility vehicles, thus portable. Unlike the previously described devices, these are bulky and can have a huge weight, and the price can be as high as hundreds of thousands of USD due to the fact that they are highly specialized devices, designed solely for that purpose. These kind of cameras are also used by firefighters, rescue technicians, crop monitoring and other terrain mapping activities.

Figure 9: IR camera installed underneath the cockpit of a helicopter.

6. STUDY OF COMERCIAL LOW-COST SOLUTIONS

Over the next section, a low-cost alternative for temperature measurements will be presented. First of all, a system for signal processing and optional storage/transmission will be presented. Secondly, different electronic-based sensors will be selected and analyzed. Finally, a study of the system using all the different sensors will be carried away.

This study will be conducted measuring temperatures in a controlled environment with the developed system and with a reference thermometer at the same time. The data obtained by the experiment will enable the comparison of the systems and sensors with a reliable source and also will be used in order to calibrate the sensors for future uses in the applications proposed at section 7. DEVELOPMENT OF APPLICATIONS.

6.1. SIGNAL PROCESSING SYSTEM

For the signal processing system, the Arduino™ Microcontroller Development Platform was selected, specifically the UNO model. This decision was made due to the low-cost of the platform, easiness of use, expansion capabilities of the platform, friendly IO management and its open-source nature, features which will be further developed along the next sections.

Figure 10: Arduino UNO, the most standard Arduino development platform, from [18].

6.1.1. ARDUINO MICROCONTROLLER DEVELOPMENT PLATFORM

The current standard Arduino development platform [18] is based on an ATmega328 8-bit programmable microcontroller. The platform also contains built-in features necessary or useful for control, including a timer, internal and external interrupts, onboard USB connector to interface with a personal computer, serial and other communication-protocol capabilities, programmable watchdog timer and energy-saving modes. Some Arduino platforms will not have the same characteristics as the presented below though.

6.1.1.1. ATmega328 Microcontroller Specifications and Memory

The device operates [18] at either 5V level and 16MHz oscillator speed or 3.3V level and an oscillator speed of 8MHz. The microcontroller memory [19] is formed up with 32 kilobytes of flash memory for program storage and initialized data, a Static Random Access Memory (SRAM) which can be read and written from the executing program and an EEPROM which can be used as a non-volatile memory but has to be read byte-by-byte.

It is important to understand the usage of the SRAM by the running programs as usually, when the program crashes for no apparent reason (when the user is sure that the program is well written and consistent), it is due to memory problems with the SRAM. When running a program the SRAM is used for several purposes:

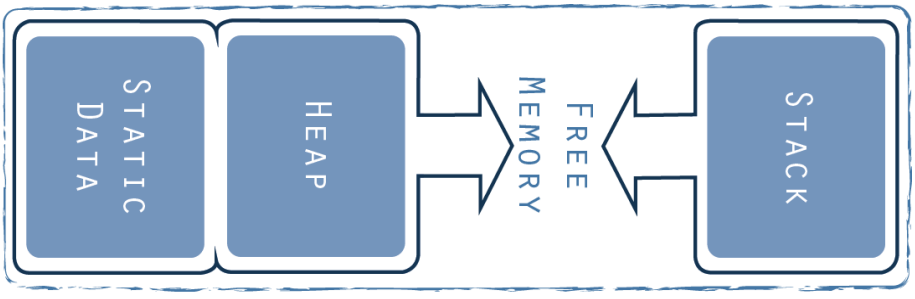
- Static Data: This is a block of reserved space in the SRAM for all the global and static variables of a program.
- Heap: The heap is for dynamically allocated data items. The heap grows from the top of the static data area up as data items are allocated (see Figure 11).
- Stack: The stack is for local variables and for maintaining a record of interrupts and function calls. The stack grows from the top of memory down towards the heap. Every interrupt, function call and/or local variable allocation causes the stack to grow. Returning from an interrupt or function call will reclaim all stack space used by that interrupt or function (see Figure 11)
- 

Figure 11: Diagram of SRAM usage, from [18].

Memory issues will occur when the heap and the stack grow enough to collide, which will lead the stack to partially overwrite the heap and vice versa, leading to malfunction of the program or a total crash.

Arduino model	Processor	Flash	SRAM	EEPROM
Standard board (UNO, Menta, Nano, Boarduino, Uno Ethernet)	ATmega328	32K	2K	1K
Leonardo, Micro, Flora, 32U4 Breakout, Teensy, Esplora	ATmega32U4	32K	2.5K	1K
Mega, MegaADK	ATmega2560	256K	8K	4K

Table 6: Processor and memory characteristics of different models of Arduino boards, from [19].

6.1.1.2. Arduino Standard Platform Inputs and Outputs

The standard Arduino platform consists of 14 digital pins and 6 analog pins [18]. For the digital, pins 0 and 1 correspond to the serial ATmega communication wires Tx and Rx which can be used for UART TTL serial communication (see section 6.1.1.5. Arduino Communication); pins 2 and 3 can be configured as external interrupts switched on a low value, a rising or falling edge or a change in value; pins 3, 5,6,9,10 and 11 can provide an 8-bit PWM; pins 10, 11, 12 and 13 support SPI communication using the SPI library (see section 6.1.1.5. Arduino Communication); and pins A4 and A5 support I2C communication using the Wire library (see section 6.1.1.5. Arduino Communication). For the 6 analog pins, all have an analog to digital (A/D) converter; and pins A4 and A5 can be used for TWI/I2C communication using the Wire library (see section 6.1.1.5. Arduino Communication).

6.1.1.3. Arduino Expansion

The Arduino board is designed to allow expansion through the connection of auxiliary boards (also known as shields). The shields connect by mating pins which are arranged in the same physical configuration as the Arduino board and simply plug them onto the top of the Arduino board. The shields are then controlled by the Arduino microcontroller and programs which access the shields' pins through the Arduino pins.

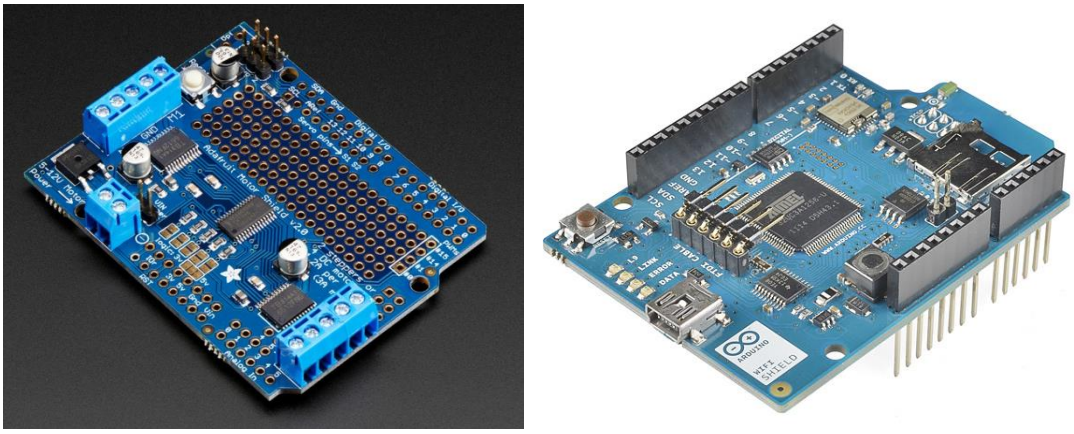


Figure 12: Motor/Stepper/Servo shield developed by Adafruit (left) and a WiFi shield developed by Sparkfun.

There is a wide variety of shields, such as SD loggers, GPS modules, wireless modules, BUS shields, etc. The fact that all shields are physically the same make them stackable one over the other, allowing the user to use more than one shield at a time. Often these shields have open-source libraries which facilitate the operations with the shield's devices, which allow users to quickly integrate new devices as it is developed in the next section.

6.1.1.4. Arduino Software

The microcontroller is programmed with a computer using the Arduino Integrated Development Environment (IDE). It also facilitates real-time serial connection with the board. This software is open-source and available for free download [20]. Through the IDE, the user can program the microcontroller using a C++ based language. The IDE compiles and checks the code for errors and it uploads it to the microcontroller.

Thanks to the fact that it is an open-source project, Arduino benefits from the collective efforts of the Arduino community in order to make it vaster in functionality. There is a huge amount of developers which have interest in the Arduino project or that manufacture hardware for the Arduino and want the user to have an easier experience with it that develop their own Programming libraries. These libraries contain routines to simplify the task of user-level programming and incorporate advanced features, sample code and complete programs available to free-download, use and modify as needed.

The program code is structured as follows:

- First, the libraries which will be used through the program are called and the global variables initialized

- Secondly, a structure called *setup* is entered. All the actions which have to be carried away when initiating the program must be written in this structure, such as the pin modes and the initialization of external devices and serial communication.
- After *setup*, the structure *loop* is found. All the instructions found in this structure will be carried away sequentially until the end of the structure. Reached this point, *loop* will re-start, repeating all its instructions sequentially. This process will repeat indefinitely until the Arduino is shut off.
- Between *setup* and *loop* or after *loop*, *functions* may be called. *Functions* are structures which have certain inputs which are manipulated and then return a variable. *Functions* may be called inside both *setup* or/and *loop*, which causes the code inside the function to run.
- *Interrupts* are also defined in these areas. *Interrupts* are *functions* which activate due to an internal or external event no matter what is happening in that moment. Its name comes from the fact that it interrupts the *loop* instructions when the event presents in order to run the *interrupt*'s code.

6.1.1.5. Arduino Communication

The ATmega microcontroller itself communicates through a two-wire serial (transmit, 'Tx', and receive, 'Rx') UART TTL connection, accessible for the user through digital pins 0 and 1, as explained in section 6.1.1.2. Arduino Standard Platform Inputs and Outputs. The Arduino platform has a chip which converts the output/input signals in order to enable direct USB communication with a PC through the USB port. Some cheaper and smaller platforms do not incorporate this serial-USB chip (neither the USB connection) and a special cable which has the serial-USB chip built-in has to be used in order to link up the Arduino board and a PC.

For communications with external components (peripherals such as sensors) several protocols have been developed. Some of these communication protocols are the I2C, the Dallas 1-Wire protocol or the SPI.

The Inter-Integrated Circuit protocol, also known as I2C, was developed by Philips Semiconductor. It is a two-wire serial (one of the lines is the data line (SDA) and the other one is a serial clock line (SCL)) transfer protocol designed for communications between integrated-circuit chips and microcontrollers. Two IO pins of the Arduino must be designated in order to support this communication. Each I2C device has its own unique identification number and address, allowing multiple devices to be connected to the same I2C lines. Ultimately this enables the user to have multiple devices connected using up just 2 pins.

The Dallas 1-wire protocol, developed by Dallas Semiconductor, uses a single IO pin for communication. This protocol is similar to I2C, each 1-wire device has its own unique ID which means that multiple 1-wire devices can be connected on the same wire. Optionally, the same communication wire can be set to be used as power supplier to the external device, which means that the user can have a device running and communicating with just two wires (the power/communication wire and the ground wire), extremely simplifying the wiring. This protocol was firstly developed for Dallas components, but thanks to its great characteristics it has been adopted by other components developers as a standard.

The Serial Peripheral Interface, or SPI, is a four-wire system developed by Motorola and provides a serial data link that operates in full duplex mode. SPI devices communicate in master/slave mode using three IO pins and the forth one is used in order to enable the master to select with which device it is communicating.

6.2. SENSORS

The following pages are dedicated to present and explain the different type of electronic sensors that have been used for the study.

6.2.1. THERMOCOUPLES

For the thermocouples, a type-K glass braid insulated thermocouple was selected. It was obtained for $9,99 at Adafruit online store [21]. Its temperature range goes from -100ºC to 500ºC (more temperature could damage the glass braid insulation) with an output range of -6 to +20mV.

Due to the fact that Arduino does not have built-in amplification and CJC for thermocouples, an external one was purchased. MAX31850K 1-Wire Thermocouple Amplifier from Adafruit [22] includes all of these features in addition to analog to digital conversion and 1-wire communication support through Dallas 1-Wire protocol. Its price is $14,95 and it was selected due to the fact that it covers all the needs in addition to the added feature of 1-wire capabilities at a low price.

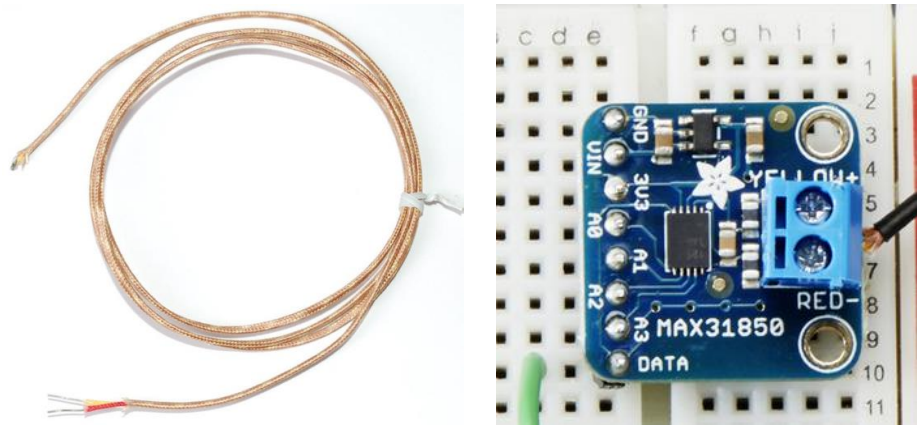


Figure 13: type-K glass braid insulated thermocouple (left) and MAX31850K (right).

However, the MAX31850K's output logical level is set at 3,3V while Arduino works at 5V as a default. Due to the fact that the communications through the one-wire Dallas protocol are forth and backwards, a bi-directional level shifter which adapts the signal from 3,3V to 5V and vice versa is needed. For this job, Adafruit's BSS138 4-channel I2C-safe Bi-directional Logic level converter was obtained for $3,95 [23].

The wiring goes as follows (see Figure 14):

- The thermocouple wires must be connected to the proper wire port of the MAX31850K amplifier.
- The MAX31850K ground pin must be connected to ground, in the example the black wire (Arduino has a built in ground pin). The Vcc pin must be connected to a 5V supply, in the example the red wire(Arduino has a built in 5V pin). Alternatively, this device can be set in parasitic mode, mode in which the Vcc pin does not need to be plugged in and the device powers from the data wire itself (but this can increment the measurements error). The data pin must be connected to the data wire (green wire). If it is desired to plug multiple thermocouples at the same time, the same data line can be shared which means that the user can have any number of thermocouples connected just to one pin of the Arduino.
- For the BSS138 level shifter, the data line from the amplifier/s (green wire) must be plugged to one of the Low-level pins (for example, the A1). The High-level partner of the previous pin (in this case it would be pin B1) must be connected to the input pin of the Arduino (blue wire). Finally HV (high voltage) pin must be plugged to 5V and LV (low voltage) pin must be connected to 3,3V through a 4.7K resistor (yellow wire). Both voltage levels can be delivered by the Arduino itself.

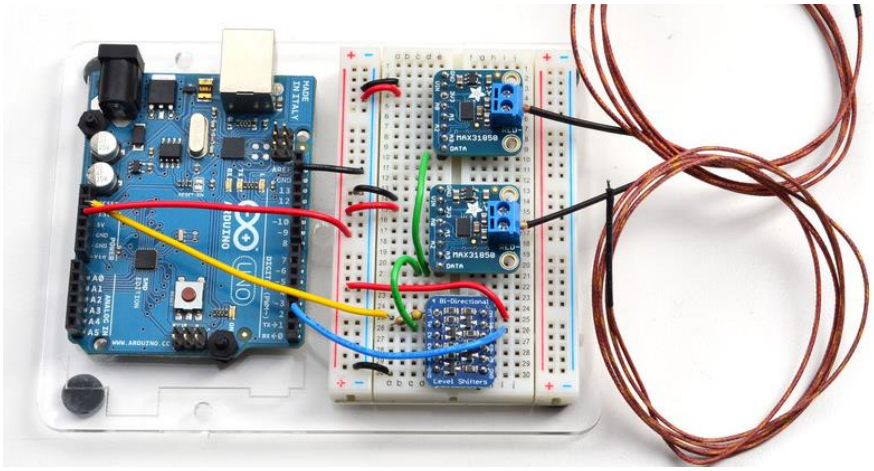


Figure 14: Wiring of the thermocouple system for two thermocouples, from [22].

This system will deliver the temperature data already in ºC thanks to the MAX31850K amplifier. There is also software support in the form of libraries in order to interact with the amplifiers, which makes it a more flexible and easy to use device.

6.2.2. RTDs

RTDs have not been used for this study as they are harder to find, more expensive and more fragile, which led to consider them out of the scope of this work.

6.2.3. THERMISTORS

10KΩ PTC thermistors were bought for 1,20€ at Diotronics, a local shop in Barcelona. These ones were selected as they are the most commonly found and widely used for general purpose. The temperature range is unknown, but for thermistors it is typically set around -80 to 120 ºC.

These are purely analogic devices (nothing such as the thermocouple amplifier is being used which converts the signal to digital) so each thermistors must be plugged in an Arduino analogical input. This means that at its best, 6 thermistors could be connected at a time. The fact that we do not need an amplifier for the thermistors is due to the fact that they are way more sensitive than thermocouples and the outputs are easier to read and less vulnerable to errors.

The wiring consists of a simple voltage division (see Figure 15):

- One end of the thermistors is plugged to the 5V level pin of the Arduino (red wire). The other end is connected at a common node.

- A 10KΩ resistance is connected between the common node and the ground pin of the Arduino (black wire).
- A wire is set from the common node to the analogic input in the Arduino (green wires).

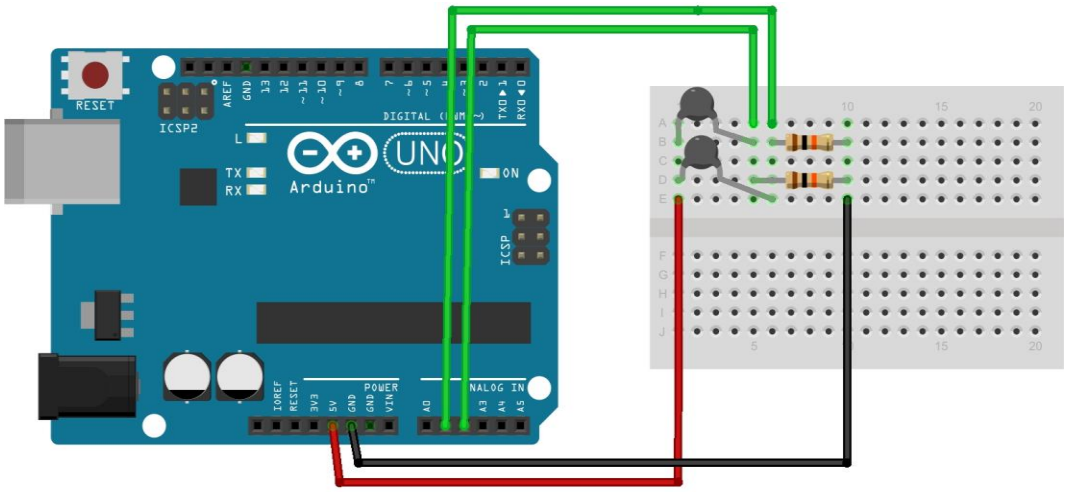


Figure 15: Wiring of a two thermistor temperature sensing system.

This system reads the voltage at the common node, knowing that the voltage at the other end of the thermistors is 5V. The analog read has a 10-bit analog to digital converter, which means that the output of the measurement will be a number between 0 and 1023 proportional with the value of the common node with respect to the supplied 5V. With the experiment that will be carried away, it will be possible to determine the real temperature as it will be possible to calculate the parameters for the different equations presented in section 5.1.3. THERMISTORS. The resolution of the whole system will be calculated too.

It must have to be taken into account that the parameters mentioned can be found in the literature for thermistors with the same resistance values. However these are nothing but approximations, as there will be slight differences in behavior from one thermistor to another, making necessary to carry out an experiment such as the one that will be presented in this work in order to have a minimum-error measurement. With the experiment data it will be possible to define the error that could be induced by using these parameters.

6.2.4. IC SENSORS

For IC sensors, the Dallas DS18B20 was chosen [24]. At a cost of 2,70€ per unit, this digital temperature sensor has a digital output of the temperature with a temperature range that goes from -55ºC to 125ºC with a claimed accuracy of ±0,5ºC for most of its range. It has 9 to 12 bit selectable resolution (setting a higher resolution makes the readings slower) and also counts with a built-in temperature-limit alarm system. Finally, this sensor works with the 1-wire Dallas communication protocol which makes possible to have multiple of them connected on one single input pin, as well as being able to run in parasitic mode, which lets the user get rid of the power line and power the sensor through the data line. Actually, as they use the same communication protocol as the thermocouples' system, both types of sensor can be set up in the same data line without any problem.

The wiring is very simple (see Figure 16):

- The Dallas GND pin is connected to ground (black wire).
- The Dallas Vdd pin is connected to +5V (red wire). This sensor can be connected in parasitic mode in the same way that the MAX31850K can be.
- The Dallas DQ pin is connected to the data wire (green wire).
- The data wire is connected to a digital input of the Arduino (green wire).

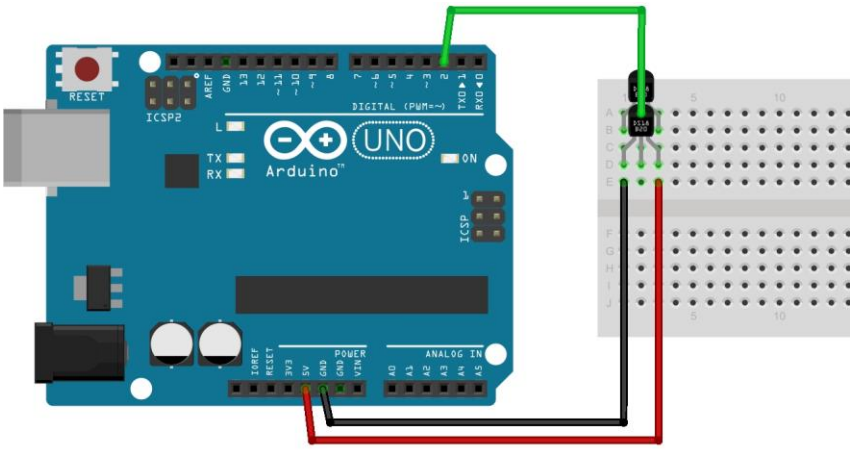


Figure 16: Wiring of a two Dallas DS18B20 temperature measurement system.

6.2.5. IR SENSORS

The melexis MLX90614-ACF was chosen [25]. Some were bought bought through Future Electronics at a cost of 17,45€ each while some others were donated by Dicrom. The price of this sensor is higher due to the fact that it an infrared thermometer and this

technology is usually more expensive (although other IRs from Melexis from the same family as this one with lower capabilities can be bought for just 7,40€). This sensor has a -40 to 125ºC temperature range. The FOV for this model is of 10º. It has a built-in low-noise amplifier and a 17-bit A/D converter with an output resolution of 0,14ºC. They also have included an internal thermometer, as the measurements taken by the device can be affected by temperature variation. The model used for this study has an included temperature-gradient compensator in order to prevent error from this nature in the measurements.

This sensor transmits its data through SMBus communication protocol, which is a derivate of I2C, and this makes possible to have more I2C devices connected to the same data line (it is not compatible with Dallas 1-wire protocol though, so it cannot share data line with those devices).

The wiring of this sensor is as follows (see Figure 18):

- The GND pin is wired to the Arduino's ground (black wire).
- The PWR pin is connected to +5V (red wire).
- SCL pin must be connected to the I2C clock pin on the Arduino, which corresponds to A5 (yellow wire).
- SDA pin must be connected to the I2C data SDA pin on the Arduino, which corresponds to A4 (blue wire).
- Two 4,7KΩ resistors must be plugged between the SDA and SCL to the power line in order to be used as pull up resistors.

Figure 17: MLX90614-ACF (top) and MLX90614 sensors family pin arrangement (bottom).

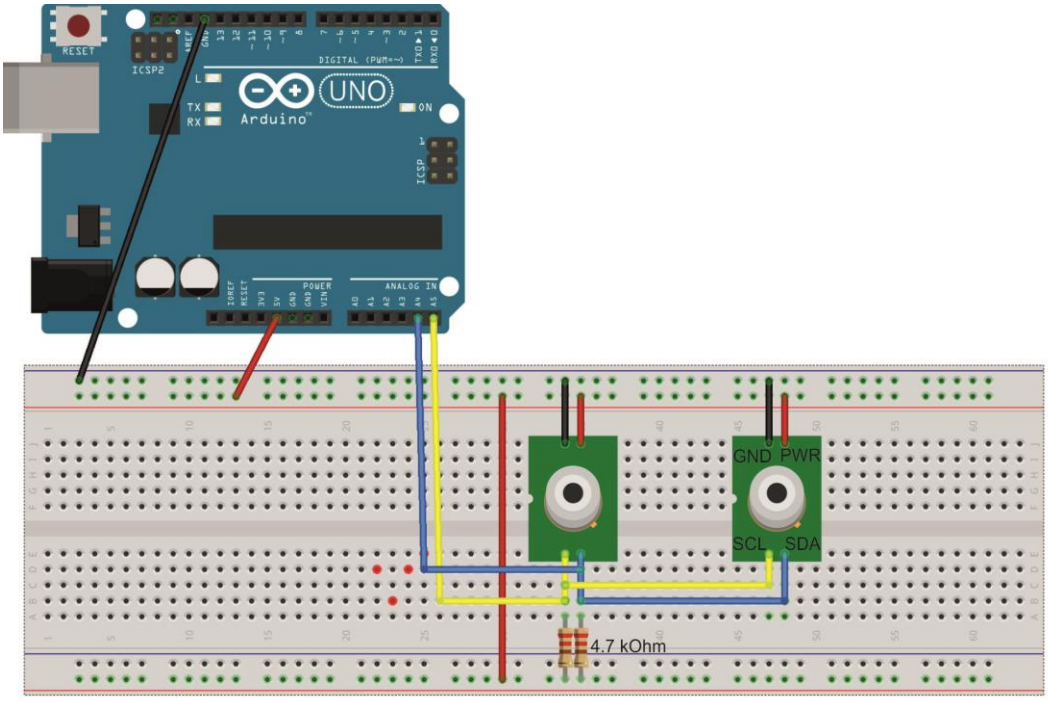


Figure 18: Wiring of a MLX90614 family sensors to an Arduino.

6.3. SOFTWARE FOR INTERFACING THE ARDUINO AND THE SENSORS

In this section, the basic Arduino program instructions in order to make each sensor data readable through the serial terminal is going to be explained in a schematic way. The complete Arduino code can be found at the annex at section 1.1. EXAMPLE CODE. The way this section is going to be structured is explaining for each sensor what libraries are needed, what global variables need to be defined, which lines of code are needed inside setup, what functions and interrupts are used and how the loop works. If needed, extra information will be attached.

6.3.1. THERMOCOUPLE WITH THE MAX31850K

The thermocouple code is actually focused to communicate with the MAX31850K, which uses the Dallas one-wire protocol. The code presented (which can be found at the annex section 1.1.1. THERMOCOUPLES: MAX31850K) will be the one needed in order to make the system showed in Figure 14 work.

6.3.1.1. Libraries

- **OneWire.h:** is the library developed by Dallas in order to support its one-wire communication protocol. As it was developed for Dallas' sensors and not

MAX31850K, this OneWire library must be a slightly modified one that Adafruit developed, which can be found at [22].

- **DallasTemperature.h:** This library facilitates functions which make user-level interaction with dallas-based devices much easier, such as requesting temperatures, getting device addresses, etc.

6.3.1.2. Global variables

- **OneWire:** an oneWire type variable must be defined in order to set which pin will be the responsible to hold the oneWire communication.
- **DallasTemperature(&oneWire):** a DallasTeperature type variable must be defined in order to pass the oneWire reference declared previously to the Dallas Temperature oneWire.
- **DeviceAddress:** several variables will be declared under this variable type, one for each sensor expected to be in our system (in the example code there are two).

6.3.1.3. Setup

In the Setup, the serial communication is started and the Dallas library initialized. Afterwards, the number of sensors is defined and each sensor is assigned to the previously declared global variables *deviceAdress*.

- **Serial.begin(9600):** This command initializes the serial port for terminal-communication with the PC.
- **Sensors.begin():** this command initializes the Dallas Temperature library.
- **Sensors.getDeviceCount:** This function checks for *sensor* type (defined previously) devices on the oneWire pin and returns the quantity of devices it found. As it is set in the example code, this will print this information to the serial-monitor through the Serial.print command.
- **Sensors.isParasiteMode():** This function is a Boolean which returns true if the sensors are connected in parasitic mode or powered through the power pin instead.
- **Sensors.getAddress(*deviceAddress* , *number*):** This line of code gets the address of the sensor which is located in the oneWire line at the position defined by *number* and assigns the address number to *deviceAddress* variable, defined before at the global variables.

6.3.1.4. Functions and Interrupts

For this code no interrupts where used. Three functions where defined in order to request data to the sensors. This economized code as the same actions must be carried away for all the sensors.

- **PrintAddress(*deviceAddress*):** this function prints the address of the sensor referenced to *deviceAddress.*
- **printTemperature(*deviceAddress*):** This function prints the temperature in Celsius and Fahrenheit of the sensor referenced to *deviceAddress*. The Celsius temperature is obtained through the **sensors.getTempC(*deviceAddress*)** instruction from the Dallas library, which returns a float with the Celsius temperature. Before using this function, a temperature request has to be send to the sensors first, as it will be seen in the loop section. In order to convert it to Fahrenheit, the Dallas library function **DallasTemperature::toFahrenheit** is used.
- **printData(*deviceAddress*):** This function prints all the data of the sensor assigned to the *deviceAddress*. Internally, it calls both PrintAddress and printTemperature functions.

6.3.1.5. Loop

The loop is simple, it sends a temperature request to all the devices and when the request is done, it uses the **printData (*deviceAddress*)** function for each sensor in order to show all the information of all the sensors.

- **Sensor.requestTemperatures():** this function sends a request through the oneWire pin in order to get the temperatures from the *sensors*. This action takes some time.

6.3.2. THERMISTORS

Thermistors are not using any communication protocol, nor the signal is digitalized. In order to read them, analog inputs will be used and then that information transformed in order to show a temperature. The code presented (which can be found at the annex section 1.1.2. THERMISTORS) will be the one needed in order to make the system showed in Figure 15 work.

6.3.2.1. Libraries

Math.h was included in order to be able to carry out mathematical operations.

6.3.2.2. Global variables

- **Int thermistor1, thermistor2:** To these variables, the analog pins which are going to be used are introduced.

6.3.2.3. Setup

In the Setup, only the serial communication needs to be started using the **Serial.begin(9600)** command.

6.3.2.4. Functions and Interrupts

For this code no interrupts where used. One function, **Thermister(int *port*)**, was declared. This function reads the signal from the analog pin *port* using the **analogRead(*port*)** standard Arduino function and assigns it to the *RawADC* variable. Afterwards this data is modified and assigned to variable *Temp* through mathematical operations which can be found at [26] based on the equations presented at section 5.1.3. THERMISTORS using average values for the unknown parameters. Temp is displayed by using the **Serial.print** function.

6.3.2.5. Loop

The loop again is simple, as it only has to call the **Thermister(*port*)** function for each thermistor that the system has connected. A delay is included in order to limit the velocity at which the data is showed.

6.3.3. DALLAS DS18B20 IC SENSOR

Due to the fact that the Dallas DS18B20 uses the same communication protocol as the thermocouples, most of the code is already commented in the section 6.3.1. THERMOCOUPLE and only the few differences they present will be commented. The code presented (which can be found at the annex section 1.1.3. IC SENSORS: DALLAS DS18B20) will be the one needed in order to make the system showed in Figure 16 work.

6.3.3.1. Libraries

It works with the same libraries that the thermocouples use. The only difference is that in order to make them run, they do not have to be the modified version that can be obtained at [22], they can be the original ones developed by Dallas.

6.3.3.2. Global variables

A new global variable is defined: *TEMPERATURE_PRECISION*. Dallas sensors resolution can be set to different values (as it will be seen later) and this variable will be used for this purpose.

6.3.3.3. Setup

The only difference with the thermocouples is that with this sensors you can change the resolution using the function **sensors.setResolution(*deviceAddress*, *TEMPERATURE_PRECISION*)**. There is also a command in order to check which resolution is set for a sensor using the command **sensors.getResolution(deviceAddress)**.

6.3.3.4. Functions and Interrupts

A function named **printResolution(*deviceAddress*)** has been added, which uses the **sensor.getResolution** command and prints the value returned to the terminal monitor.

6.3.3.5. Loop

The loop works exactly as with the thermocouples.

6.3.4. MELEXIS MLX90614-ACF IR SENSOR

The usage of the melexis sensor is really eased thanks to an Adafruit library which can be obtained from [27]. The code presented (which can be found at the annex section 1.1.4. IR SENSORS: MELEXIS MLX90614-ACF) will be the one needed in order to make the system showed in Figure 18 work.

6.3.4.1. Libraries

- **Wire.h:** this library enables makes possible the I2C communication with the melexis sensor.
- **Adafruit_MLX90614.h:** This library developed by Adafruit makes the interaction with the sensor much easier to the user.

6.3.4.2. Global variables

A type Adafruit_MLX90614 variable named *mlx1* is defined. This type of variable comes from the **Adafruit_MLX90614.h** library and it sets a new I2C address to the devices.

6.3.4.3. Setup

In the Setup, the serial communication is started and the melexis sensors initialized by using the **mlx1.begin()** and **mlx2.begin()**command.

6.3.4.4. Functions and Interrupts

No functions have been defined for this program.

6.3.4.5. Loop

The loop consists of two basic instructions, ***mlx*.readObjectTempC()** for reading the object temperature (this is, the temperature measured with the IR itself) and ***mlx*.readAmbientTempC()** for reading the ambient temperature (using the internal thermometer). Both return a float with the temperature measurement in Celsius. These commands must be called for each device connected, which is done changing the first 3 letters (*mlx*) for the assigned name of the devices (in this case, *mlx1* and *mlx2*).

6.4. ASSAY OF THE SENSORS FOR ITS STUDY AND CALIBRATION

In the next section, the method that was used in order to study the response of the sensors is presented. Different methods were carried away for the contact sensors and for the IR sensors, due to its different functionality nature, however the objective is common in both assays, compare the measurements of the sensors against the temperature given by a reference thermometer in order to evaluate its linearity, error, repeatability and offset. The results make also possible the calibration of the sensors for better measurements in the future.

6.4.1. ASSAY OF THE CONTACT SENSORS

6.4.1.1. Methodology

The difficulty when assaying the contact sensors is that it is difficult to have them all, as well as the reference thermometer, at the same temperature. In order to accomplish that, they were submerged into a thermically controlled fluid environment, which will be further explained afterwards, as well as held closely together in order to minimize errors due to possible temperature gradients.

The method firstly planned was to set the environment to a specific temperature. The sensors' signal would have been tracked through real-time display in order to be able to know when the environment was at equilibrium and all the sensors' measurements stable. At this point, the measurement of the reference thermometer would be written

down for future comparison with the sensors' measurements, which would be periodically (about every second) registered. This would have been repeated for 4 different temperatures. The advantages of this method are that the environment is for sure at equilibrium, therefore all the sensors are at the same temperature and it also permits to calculate the thermal inertia of the sensors.

However, due to a malfunction in the thermal control of the environment, which made the environment to keep being heated up rather than heat up until a specific temperature was reached, the experiment had to be slightly changed. Instead of staying at different temperature points, a continuous temperature swept was carried away. The main problem with this method was that due to the fact that the environment was never at equilibrium it could not be granted the fact that all the sensors were at the same temperature. However, due to the fact that the environment, as it will be seen in the next section, was agitated, that all the sensors were really close to each other and that through the real-time display it could be observed that none of them was lagging behind due to its own thermal inertia, it was considered that all the sensors were at the same temperature for each instant of time and minimal error would be generated due to this inconvenient. The main problem although was that due to the lack of data logging of the reference thermometer, the temperature had to be noted down by hand periodically (it was done about every two minutes) in order to be able to compare as many points as possible from the logged data of the sensors. While in the previous experiment the post processing would have been quick, in this case the points which were evaluated multiplied which increased the time dedicated to this labor. The temperature swept was carried away twice from 20ºC to 40ºC approximately, changing the physical position of the sensors for each heating in order to diagnose if the environment was not isothermal.

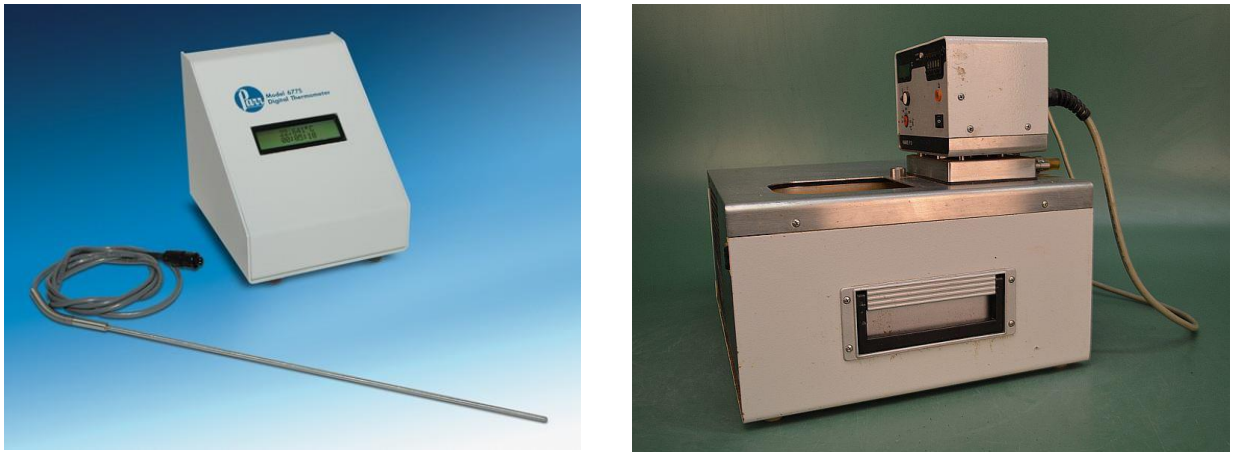


Figure 19: Parr 6775 (left) and Haake thermal bath and F3 circulator (right).

6.4.1.2. Material

The reference thermometer used was the Parr 6775 Digital thermometer [28] (Figure 19) which was already calibrated from 10 to 40ºC, which is its operational range. It is also highly accurate, with an 0.001ºC resolution.

The environment consisted of an adiabatic receptacle, a Haake W15 Thermal bath, which contained fluid thermal silicon. Thermal silicon was used due to the fact that it is dielectric, which permitted to introduce the electronic sensors inside it without damaging them or causing malfunctions. The receptacle had a device, a Haake F3 [29] (Figure 19), which was both a heater (in order to be able to control the environment temperature, although it malfunctioned) and an agitator (so that the temperature of all the silicon contained could be considered isothermal).

Seven sensors were assayed: 2 thermistors, 3 thermocouples using the MAX31850K amplifier and 3 Dallas DS18B20.

All the sensors were subjected close to each other and to the reference thermometer by a self-constructed structure in order to minimize the error caused by possible temperature gradients. The structure was constructed with PVC tubes which held the sensors' wiring, keeping the sensor upside down with a about one centimeter of free wiring in order to fine tune the final position of the sensors. These tubes were held by two perforated methacrylate plates, which had four aluminum legs to keep everything stand up. The selection of materials was not critical due to the fact that the temperatures which

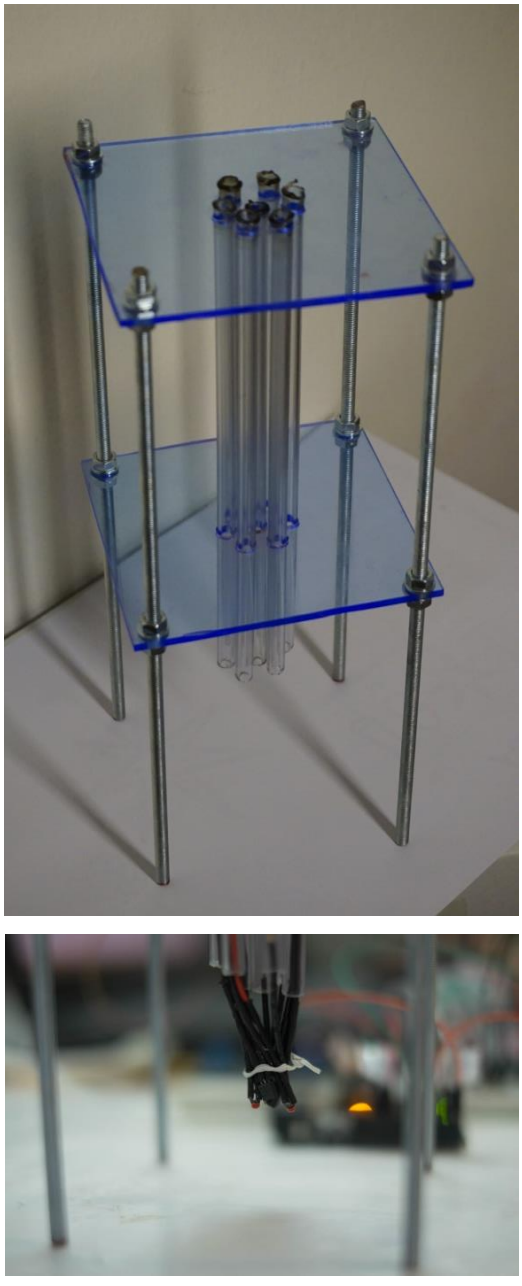


Figure 20: Self designed and constructed structure (top) and detail of how close the sensors are held together (bottom).

would have to hold were not high (up about 40ºC), so the materials were chosen following an easiness of manipulation criteria.

6.4.1.3. Wiring of the sensors with the Arduino

The wiring of all the contact sensors to the Arduino would be the superposition of all the previously presented wirings at section 6.2. SENSORS and can be seen in Figure 21. The three MAX31850K share the same 3V data line (green line), which is level-shifted with the level-shifter and the resultant high voltage (5V) data line is shared also with the two Dallas DS18B20. This data line is connected to pin 2 (so all the dallas and thermocouples will be read through the same pin). There's also a pull up resistor between this data line and the 5V line. All these sensors are fed from the 5V line and are connected to ground. The level shifter also has a 3,3V line connected to the LV (low voltage) pin. For the thermistors, they are connected exactly as in Figure 15 and they are completely independent of all the other wiring.

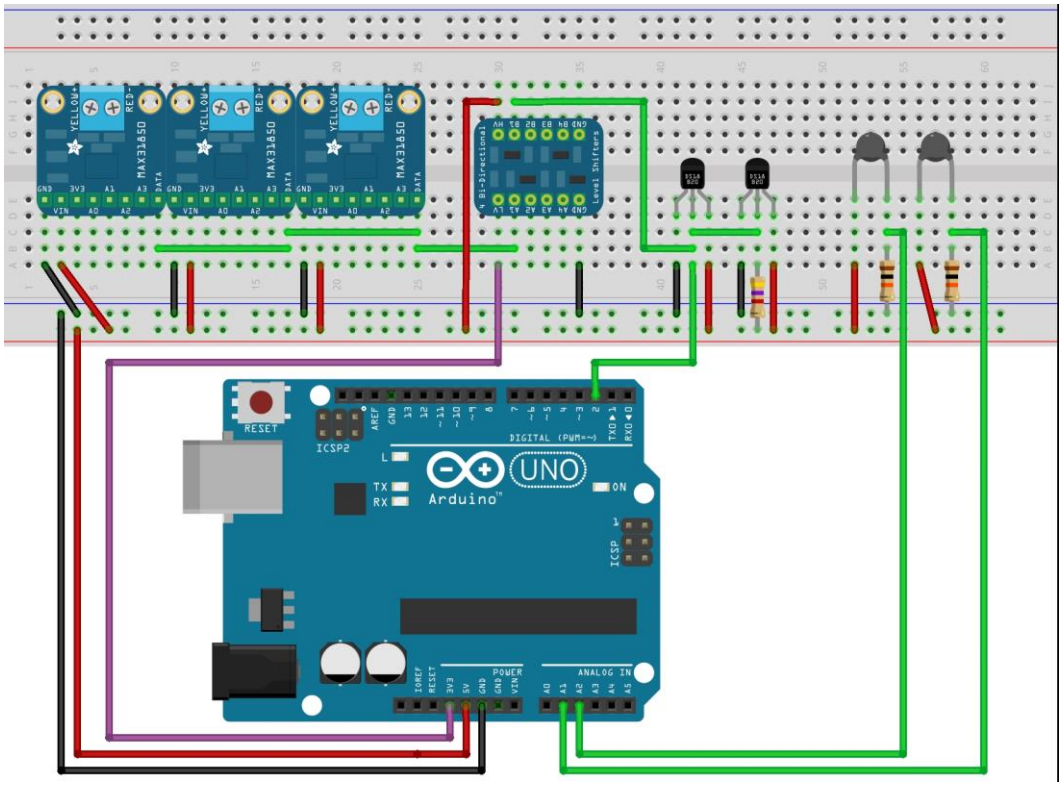


Figure 21: Wiring with the Arduino for the contact sensors assay.

6.4.1.4. Software

The software which had to be developed for this assay had to accomplish a variety of objective. The first objective was to be able to read all the sensors' raw data, which had

to be carried away by the Arduino. This raw data had to be modified in order to be understandable, which was done through the Arduino's libraries mentioned in section 6.3. SOFTWARE FOR INTERFACING THE ARDUINO AND THE SENSORS. The data received and manipulated had to be saved into a log file for post processing, which was done through matLab. Also, this data had to be showed real-time in a graph against time, which was also carried away with matLab.

The fact that some of the objectives had to be carried away with the Arduino and some with computer software (matLab) made necessary the interfacing of these two software in real-time. The code for both Arduino and matLab will be analyzed.

6.4.1.4.1. Arduino code

The code for Arduino is a combination of all the codes for contact sensors which are explained at section 6.3. SOFTWARE FOR INTERFACING THE ARDUINO AND THE SENSORS, with some extra instructions in order to interface with matLab. The full code can be found at the annex section 1.2.1.1. Arduino code.

The program works as follows:

- First of all, it initializes the libraries and variables needed in order to make the sensors function and be able to transform the raw data they send into understandable data.
- In the setup, in addition to what is required to make the sensors work (which is already explained in section 6.3. SOFTWARE FOR INTERFACING THE ARDUINO AND THE SENSORS), serial communication is established with matLab through the **Serial.begin(9600)** instruction.

  Afterwards, it runs a routine in order to check if the communication is correctly set. The Arduino sends a specific character which the matLab will have to read. When matLab reads it, it will answer with another character that the Arduino will have to read. In case that it is successful, Arduino will echo back the character in order to have confirmation that it did so.
- The functions of the code are exactly as presented in section 6.3. SOFTWARE FOR INTERFACING THE ARDUINO AND THE SENSORS.
- The loop is the one that has to be able to ensure real-time communication and measurements lecture. The program will wait until a request comes from the computer, which is carried away using an if were the condition is **Serial.available()**>0. **Serial.available()** is a function which returns true if there is input data in the buffer. When the request is received, it is stored into the

variable *mode.* With a switch the actions that have to be carried away are selected depending of what kind of information matLab is demanding. If *mode* is equal to 'T', matLab is requesting temperature readings, which means that the Arduino runs the ***sensors*.requestTemperatures()** and afterwards the functions **printTemperature(*deviceAddress*)** and **Thermister(*port*)** in order to print all the data from all the sensors. If *mode* is equal to 'M', matLab is requesting the internal Arduino clock time, which is accessed through the command **millis()** and sent with the **Serial.print()** instruction.

6.4.1.4.2. MatLab code

The objective of the matLab code is to send a temperature request and time request to the Arduino and read and store the data into a temporary buffer. Once this is done, this data must be saved into a text file under an understandable and post-processing-friendly structure. It also has to show a real-time graph of all the measurements for all the sensors versus the internal Arduino time. The full code can be found at the annex section 1.2.1.2. MatLab code.

The matLab main code (*calibration.m*) works as follows:

- First of all, the communication with the Arduino is set up. It is asked to the user via a message box (using the **inputdlg** command) at which port of the computer is the Arduino connected. Then the communication serial is set up for correctly connecting with the Arduino using the **setupSerial** function which will be explained afterwards.
- The function **logFile** is called in order to create a log file. This function will be explained later.
- The figure where the real-time graph of the measurements versus time will be displayed is initialized and given all its properties.
- A buffer for all the thermometers and the time is created.
- An infinite loop is entered were all the real-time processes are carried away. First of all, it requests and reads the time by using the **read(,s,'M')** function. It does the same for all temperatures at once using the **readTemp(s)** function. All this data is stored at the buffer, then plotted using the **plot** instruction and stored at the log file separated by tabulates using the **fprintf** function. Then the figure is forced to actualize with the new data points using the **drawnow** instruction.

The functions previously mentioned before will be now explained:

- **setupSerial(*port*)** is a function which sets all the fields for communication correctly for the port set as input variable. First of all it resets all ports, which is useful in case that the code broke before being able to cut all communications. A *serial* object with the Arduino communication port is defined and from this point on the Arduino is treated as a file. It is opened with the **fopen** instruction and then the serial check protocol is run as explained in the previous section. The user receives feedback when all the checkups have been carried away through message boxes.
- **logFile** starts by asking the user a name for the new file which is going to be created. Then it checks if a file with the same name already exists using the **exist** instruction. If it exists it asks the user whether he wants to overwrite or enter a new name. Once the name is decided, it opens the file for future modification using the **fopen** instruction and writes down the first row of the file with the label of each column of information.
- **Read(*s*,*mode*)** sends to the *serial* object *s* a request using the function **fprintf** The request is defined with a character assigned to *mode*. In this program, this is used to request the time, so the *mode* value is assigned as 'M'. After sending the request, it reads the input information from the Arduino using the **fscanf(*s*)** function.
- **ReadTemp(*s*)** is very similar to **read**, but less flexible. It is designed in order to request seven information packs to the Arduino at once (the seven sensor's measurements). It functions exactly as the **read** function, but instead of 'M' it sends a 'T' request and makes seven consecutive reads with the **fscanf** function.

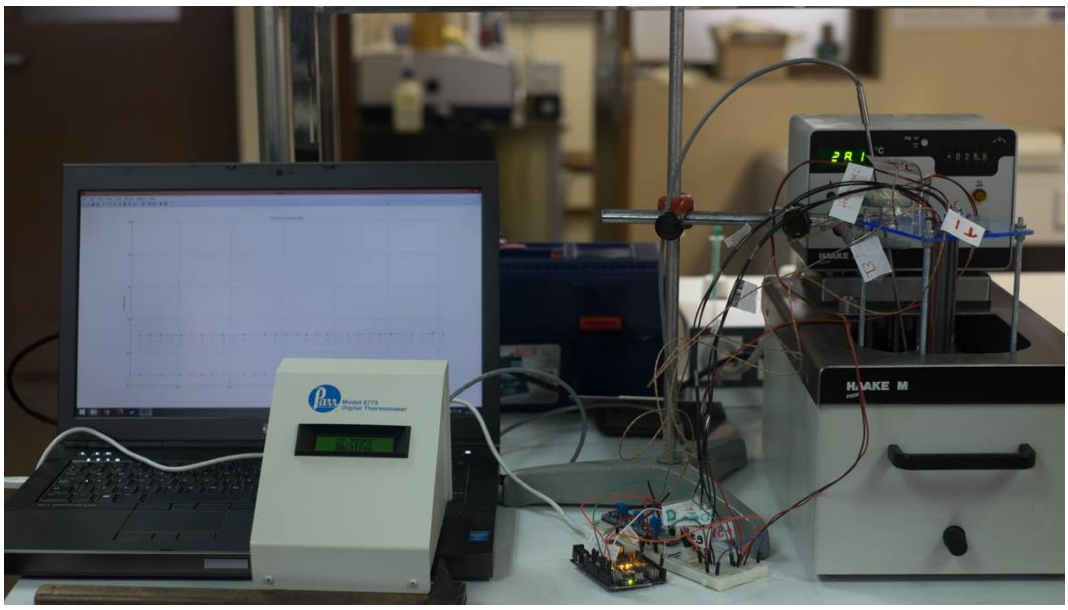


Figure 22: Photograph of all the material set up for the assay of contact sensors.

6.4.1.5. Results

In this section the results obtained will be shown and analyzed for each contact sensor. The full data obtained can be found at the annex section 2.1. DATA FROM THE CONTACT SENSORS ASSAY. The results will be analyzed through graphs which can be found at the end of this section.

In Figure 23 all the data obtained is displayed as a graph, plotting the temperatures of all the sensors versus time. Due to the high quantity of data points, it is difficult to be able to observe in detail the readings, but a general idea of the experiment can be obtained. It must be noted that the thermistors are not displayed. This is due to the fact that its raw analog data was logged and not converted to temperature right away, which means that if displayed with the other sensors data it would be out of scale and made it unpractical to read.

The two heatings and the cooling are easily recognizable. The first heating went from 26,314ºC to 39,034ºC, then the environment was cooled down naturally until 33,559ºC and heated again up to 39,310ºC. It must be noted that the cooling data was rejected as useful for analysis purposes due to the fact that the circulator had to be shut down during this process as a cause of its malfunction, which means that the fluid was not agitated therefore it cannot be considered that the fluid was isothermal. The first heating will be used for comparison and calibration while the second one is for corroboration of the calibration and results.

It can also observe the effects of the quantization of the devices due to the analog to digital conversion. It is specially marked in the thermocouples but also noticeable for the Dallas. In the reference temperature such effect cannot be seen because it is actually a spline, due to the fact that there are less data points for this thermometer because its readings had to be written down by hand.

Due to the fact that there were less reference temperature data points than sensor's data points (as the first one was noted down by hand while the later were automatically registered), in order to compare the sensor's readings with the reference temperature it was assumed that the increase of temperature between the reference temperature data points was constant. This might have introduced an slight error, as this was based up on an assumption, but the data points which could be compared was about 100 times higher, which led to a decrease of the overall error.

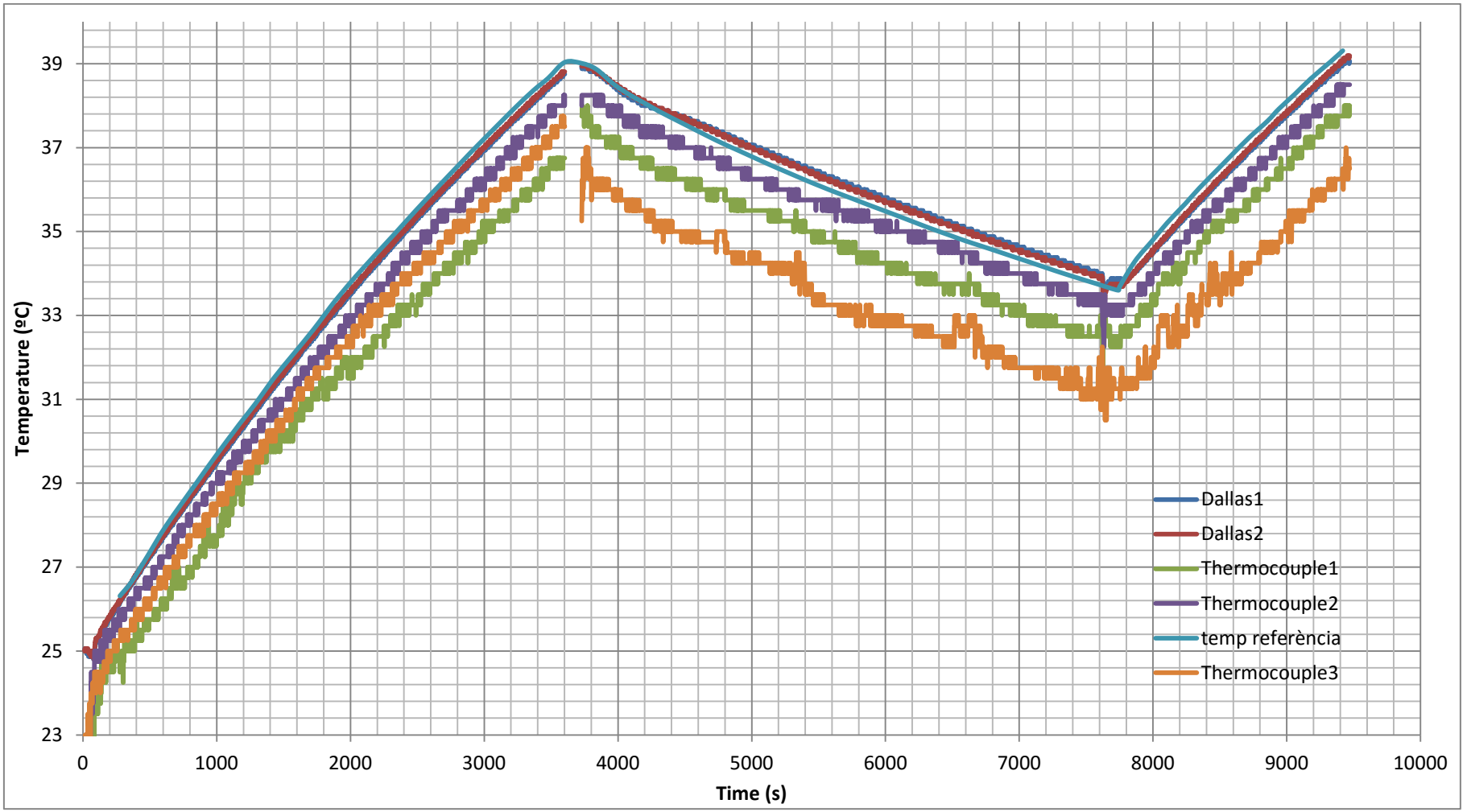


Figure 23: Graph displaying all the data obtained during the experiment.

6.4.1.5.1. Thermocouple with the MAX31850K

The temperature registered by the three thermocouples plotted against the reference temperature can be seen in Figure 24, Figure 25 and Figure 26.

What is more noticeable at first sight is its quantization. The analog to digital converter which is included in the MAX31850K has a 0,25ºC resolution, which although being small, it can be insufficient for some applications. It is also noticeable that the behavior of each sensor varies slightly, especially when comparing the thermocouple 1 readings with the other two thermocouple's readings, being the signal of the later two less noisy.

A linear regression was carried away for each one of the three thermocouples, with the following results:

- Thermocouple 1: $T_M = 0{,}9358 \cdot T_R + 0{,}1987$ [ºC] ;$R^2 = 0{,}9972$
- Thermocouple 2: $T_M = 0{,}9674 \cdot T_R + 0{,}2685$ [ºC] ;$R^2 = 0{,}9991$
- Thermocouple 3: $T_M = 0{,}9607 \cdot T_R + 0{,}0255$ [ºC] ;$R^2 = 0{,}9988$

Where T_R is the real temperature, T_M is the measured temperature by the sensor and R^2 is the square of the correlation coefficient.

It can be observed that the coefficient multiplying T_M is near 1, which means that the sensors have a highly lineal response. However this linearity fluctuates between the different sensors, being the thermocouple 1 noticeably less linear.

The offset varies depending of the thermocouple, being the thermocouple 3 the one nearest to zero with an offset of 0,0255; thermocouple 2 being the one with most offset, with an offset of 0,2685; and thermocouple 3 sitting between both with an offset of 0,1987.

The correlation coefficient is also fluctuant depending of the sensor. Thermocouple 2 is the one with a better correlation of 0,9991; while the first thermocouple's correlation is the worse, due to its noisy behavior, with an 0,9972. Thermocouple 3's square correlation coefficient has a value of 0,9988.

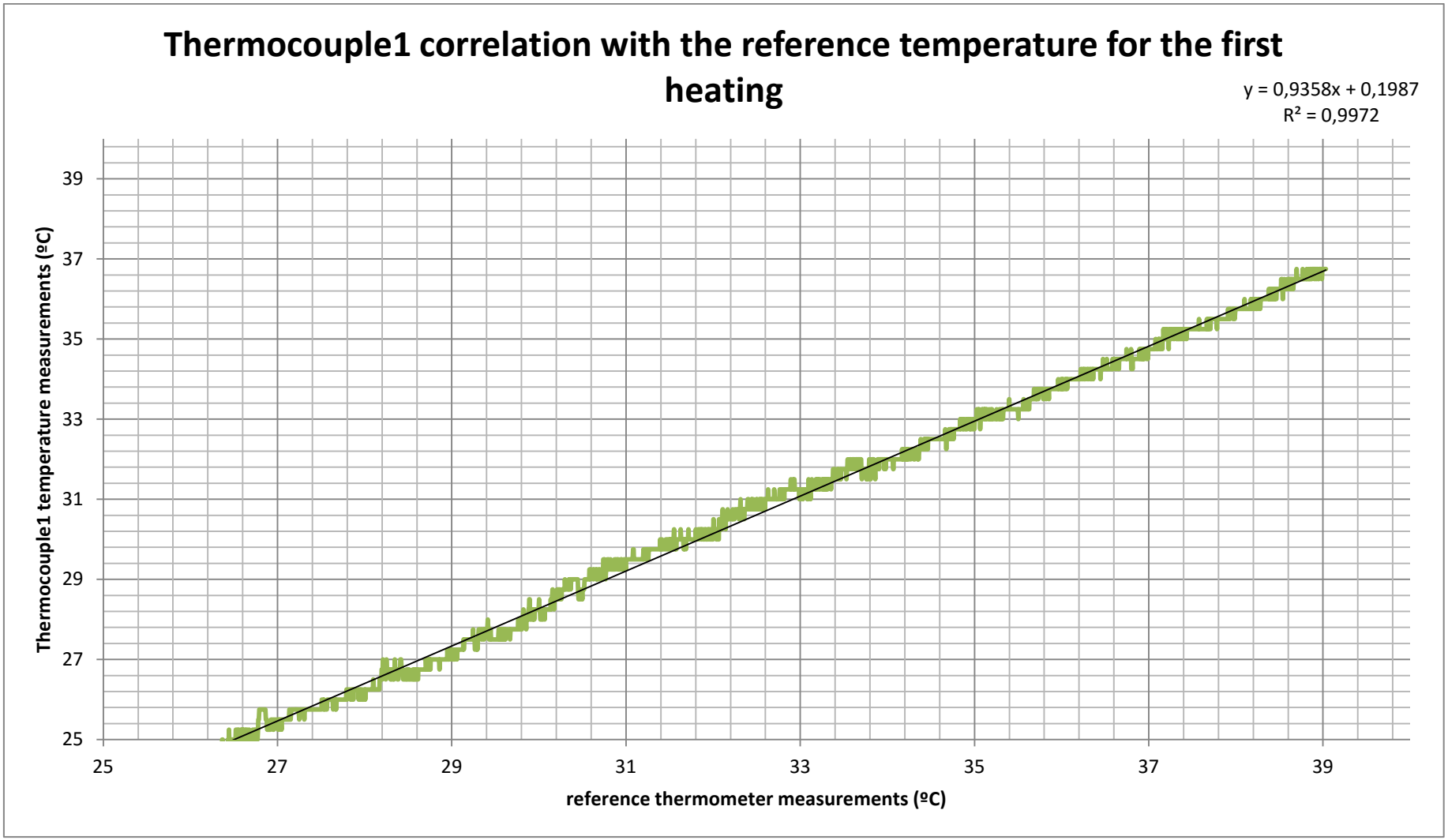


Figure 24: Graph displaying Thermocouple1 correlation with the reference temperature for the first heating.

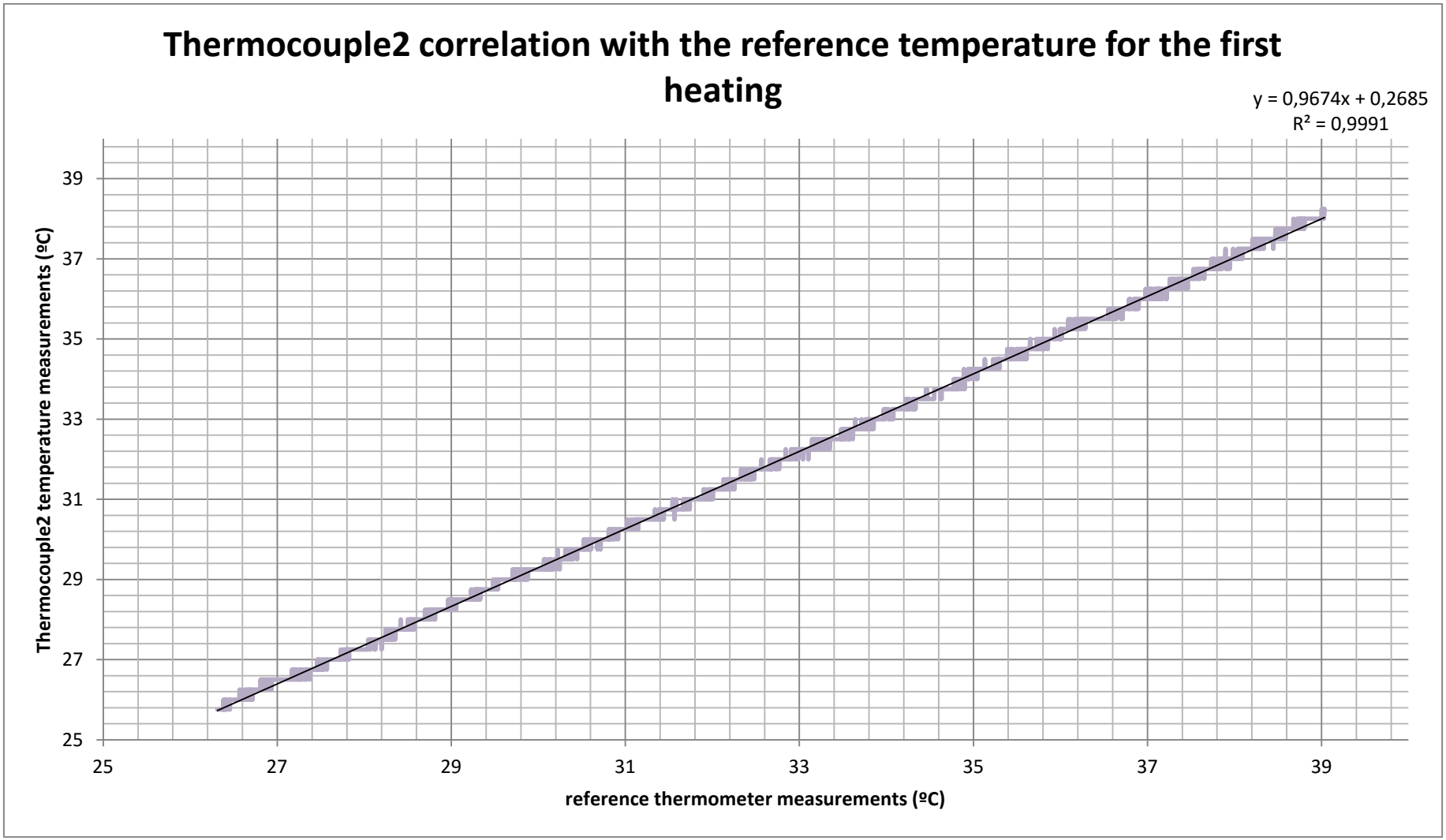


Figure 25: Graph displaying Thermocouple2 correlation with the reference temperature for the first heating.

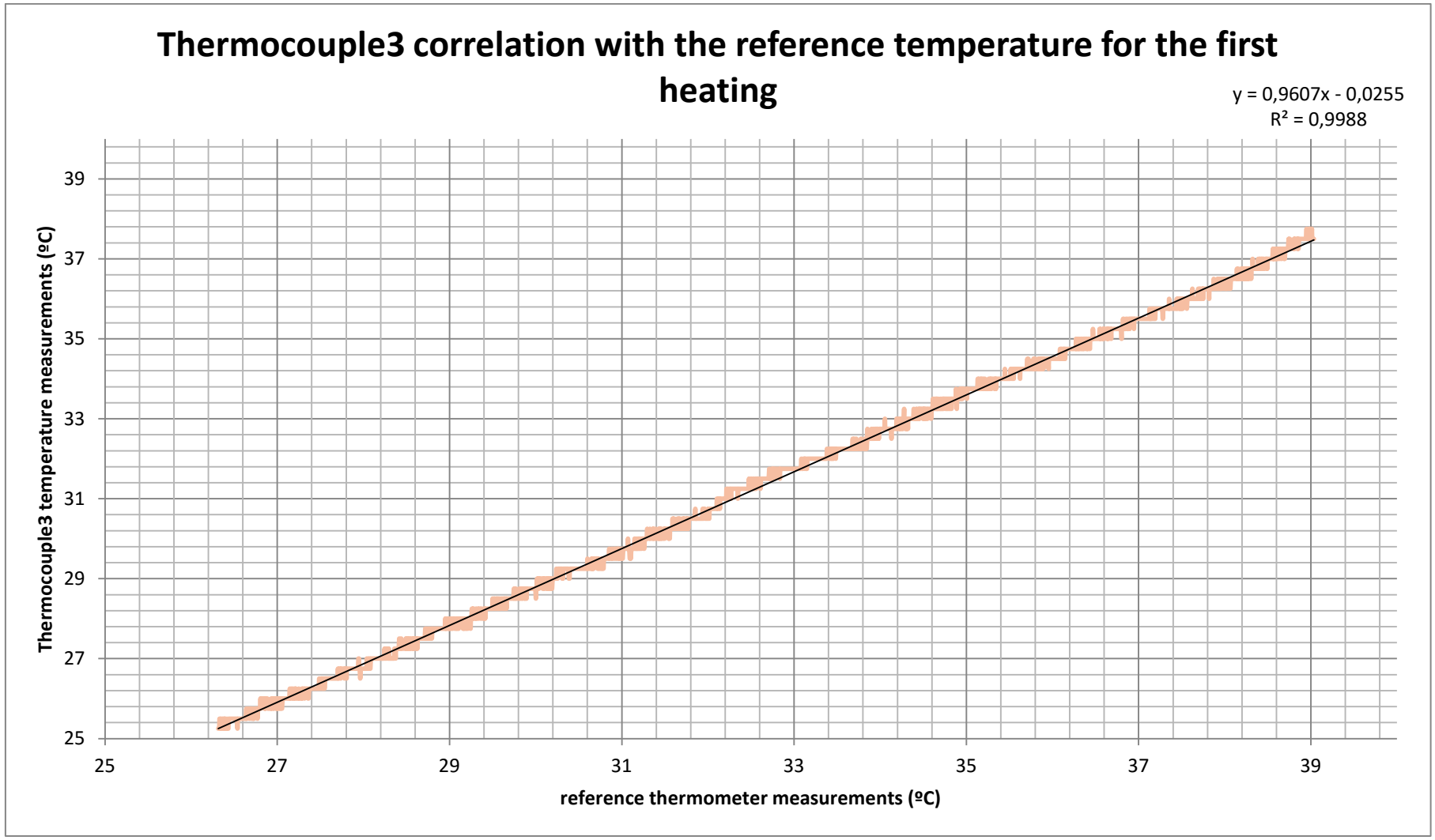


Figure 26: Graph displaying Thermocouple3 correlation with the reference temperature for the first heating.

6.4.1.5.2. Thermistors

For the thermistors, their raw digital data was plotted against the reference temperature, as it can be seen in Figure 27 and Figure 28.

A linear regression was carried away for both of the thermistors. Even though its response is not linear over wide temperature ranges, due to the low temperature and restricted temperature range at which the experiment could be carried away its behavior should be linear. The resulting regressions are shown below:

- Thermistor 1: $Raw = 10{,}265 \cdot T_R + 257{,}22 \quad [int \in [0{,}1023]] \quad ; R^2 = 0{,}9996$
- Thermistor 2: $Raw = 10{,}313 \cdot T_R + 253{,}97 \quad [int \in [0{,}1023]] \quad ; R^2 = 0{,}9995$

Where T_R is the real temperature, Raw is the digitalized measurement of the sensor (which is an integer with a value which can go from 0 to 1023) and R^2 is the square of the correlation coefficient.

With the data obtained, the sensitivity of the sensors and the resolution of the overall sensing systems could be calculated:

- The sensitivity of the sensor is defined as the output voltage change of the tension divisor per unit change of temperature.

  $$S = \frac{\Delta \mathrm{V_{out}}}{\Delta \mathrm{T}} \left[\frac{V}{^{\circ}\mathrm{C}}\right]$$

  Where V_{out} is the voltage output of the tension divider and T is the temperature.

  The raw digital data is equal to:

  $Raw = \frac{V_{out}}{V_{in}} \cdot 1024$ Rounded to the nearest integer

  Where Raw is the digital raw data.

  Hence, the sensitivity can be computed as:

  $$S = \frac{\Delta Raw \cdot \frac{V_{in}}{1024}}{\Delta T}$$

  Where $\frac{\Delta Raw}{\Delta T}$ is the gradient of the linear regression. Hence:

$$S_{thermistor1} = 0{,}0501\ [\frac{V}{°C}]$$
$$S_{thermistor2} = 0{,}0503\ [\frac{V}{°C}]$$

- Resolution, which is the smallest change that the system can detect, can be computed as follows:

  $$Resolution = \frac{\Delta V_{min}}{S}\ [°C]$$

  Where S is the sensitivity ΔV_{min} is the minimum voltage change detectable, which can be calculated as follows:

  $$\Delta V_{min} = \frac{V_{in}}{1024} = 0{,}00488\ [V]$$

  This leads to a resolution of:

  $$Resolution_{Thermistor1} = 0{,}097\ [°C]$$
  $$Resolution_{Thermistor2} = 0{,}097\ [°C]$$

As shown by the previous results, both thermistors have similar sensitivities and a resolution below 0,1ºC. The linear regression parameters are fairly similar with a high squared correlation coefficient of 0,9996 and 0,9995 respectively which means that for the range of the experiment the sensors response can be assumed to be linear.

In general, the end user of these systems will not have a reference thermometer at his disposal which means that he will not be able to find by himself the relationship between the raw digital data he obtains and the real temperature. The user will have to rely on generic parameters (which can be obtained in bibliography) for the Steinhart-Hart equation in order to be able to transform the raw digital data to temperature readings. For this reason, the raw digital data obtained in this experiment was also transformed to temperature using generic parameters for 10KΩ thermistors for this equation in order to observe how reliable these generic parameters are. The results are shown in Figure 29 and Figure 30.

In order to use the Steinhart-Hart equation with generic parameters, firstly the resistance of the thermistor must be deduced from the raw digital data. As the thermistors are used along with a 10KΩ as a voltage divisor:

$$V_{out} = \frac{R}{R_T + R} \cdot V_{in} \rightarrow R_T = \left(\frac{V_{in}}{V_{out}} - 1\right) \cdot R$$

Where R_T is the thermistor's resistance, V_{out} is the voltage output of the tension divider, V_{in} is the input voltage (in this case, 5V) and R is the resistance used in order to make the voltage divider, in this case 10KΩ.

Replacing V_{out} with the previously presented equation which relates it with the raw data:

$$R_T = \left(\frac{1024}{Raw} - 1\right) \cdot R$$

Knowing this parameter, the Steinhart-Hart equation can be directly used in order to find the measured temperature:

$$T_M = \frac{1}{a + b \cdot \ln(R_T) + c \cdot (\ln(R_T))^3}$$

Where T_M is the measured temperature; a, b and c are the generic coefficients and R_T is the thermistor's resistance.

In this case, the thermistors are 10KΩ and the coefficient's value are:

$a = 0.001129148$ $b = 0.000234125$ $c = 8.76741 \cdot 10^{-8}$

The resulting temperature measurements of the thermistors should have a linear response at any temperature range. The linear regressions are shown below:

- Thermistor 1: $T_M = 0{,}9830 \cdot T_R + 0{,}359$ [℃] ;$R^2 = 0{,}9996$
- Thermistor 2: $T_M = 0{,}9857 \cdot T_R + 0{,}1104$ [℃] ;$R^2 = 0{,}9996$

Where T_R is the real temperature, T_M is the measured temperature by the sensor using the generic parameters for the Steinhart-Hart equation and R^2 is the square of the correlation coefficient.

As it can be observed, both thermistors have a really close-to-one linear response (0,983 and 0,9857 respectively). The offset is low for both cases but it is not as consistent, being 0,359 for the first thermistor and 0,1104 for the second one. The correlation for both thermistors is 0,9996; from which it can be deduced that the response is highly lineal and that noise has little effect on the it.

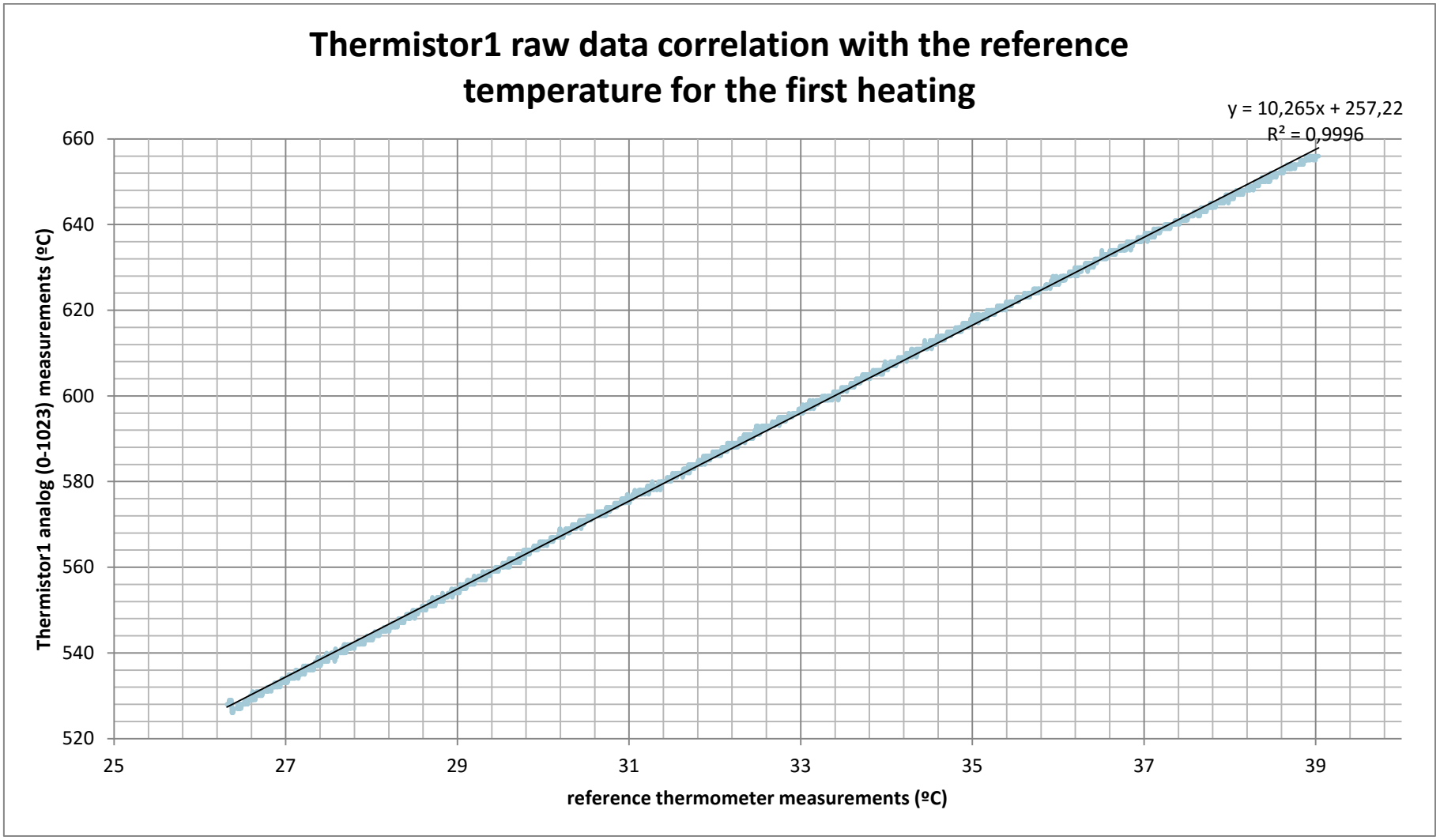


Figure 27: Graph displaying Thermistor1 correlation with the reference temperature for the first heating.

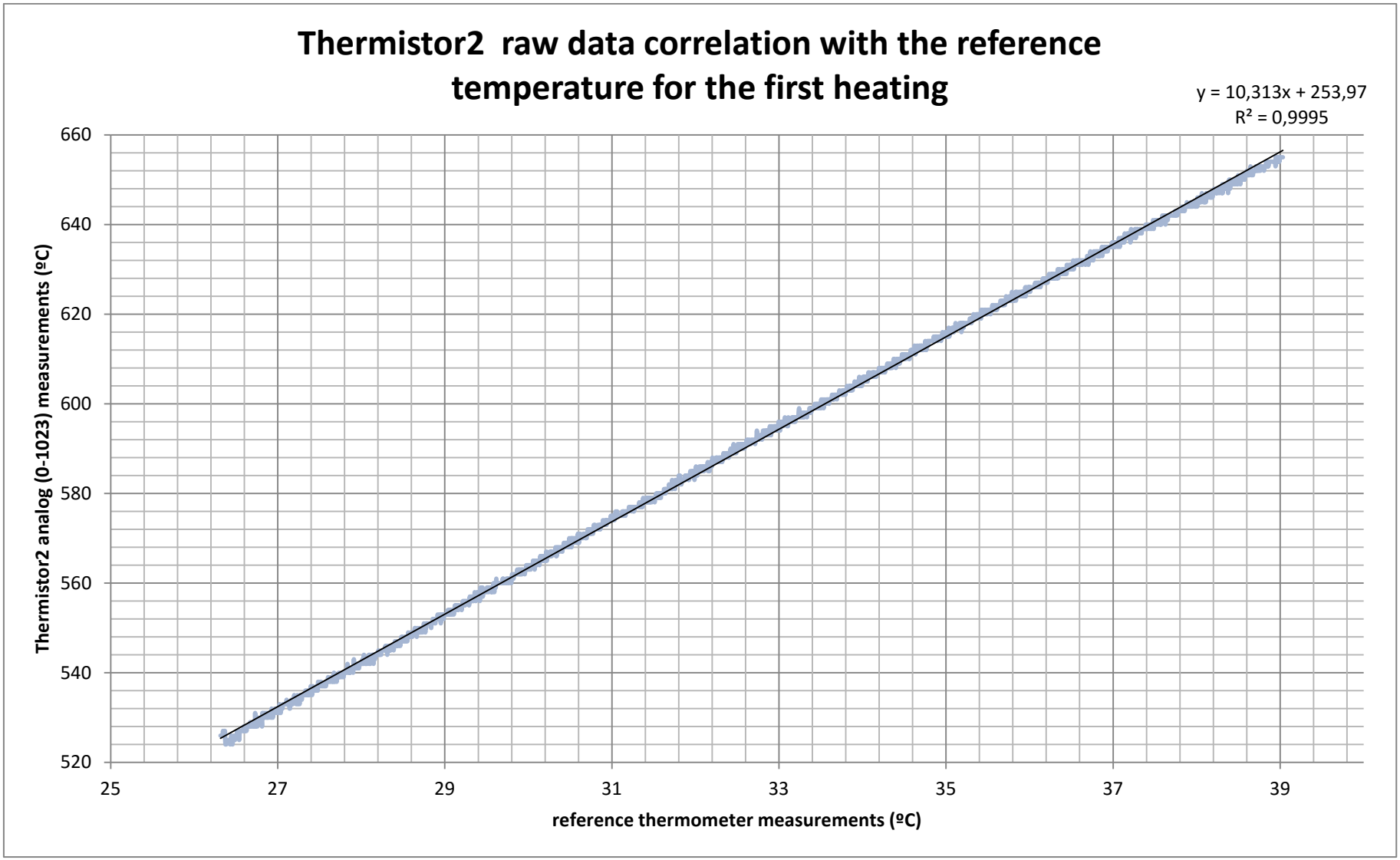


Figure 28: Graph displaying Thermistor2 correlation with the reference temperature for the first heating.

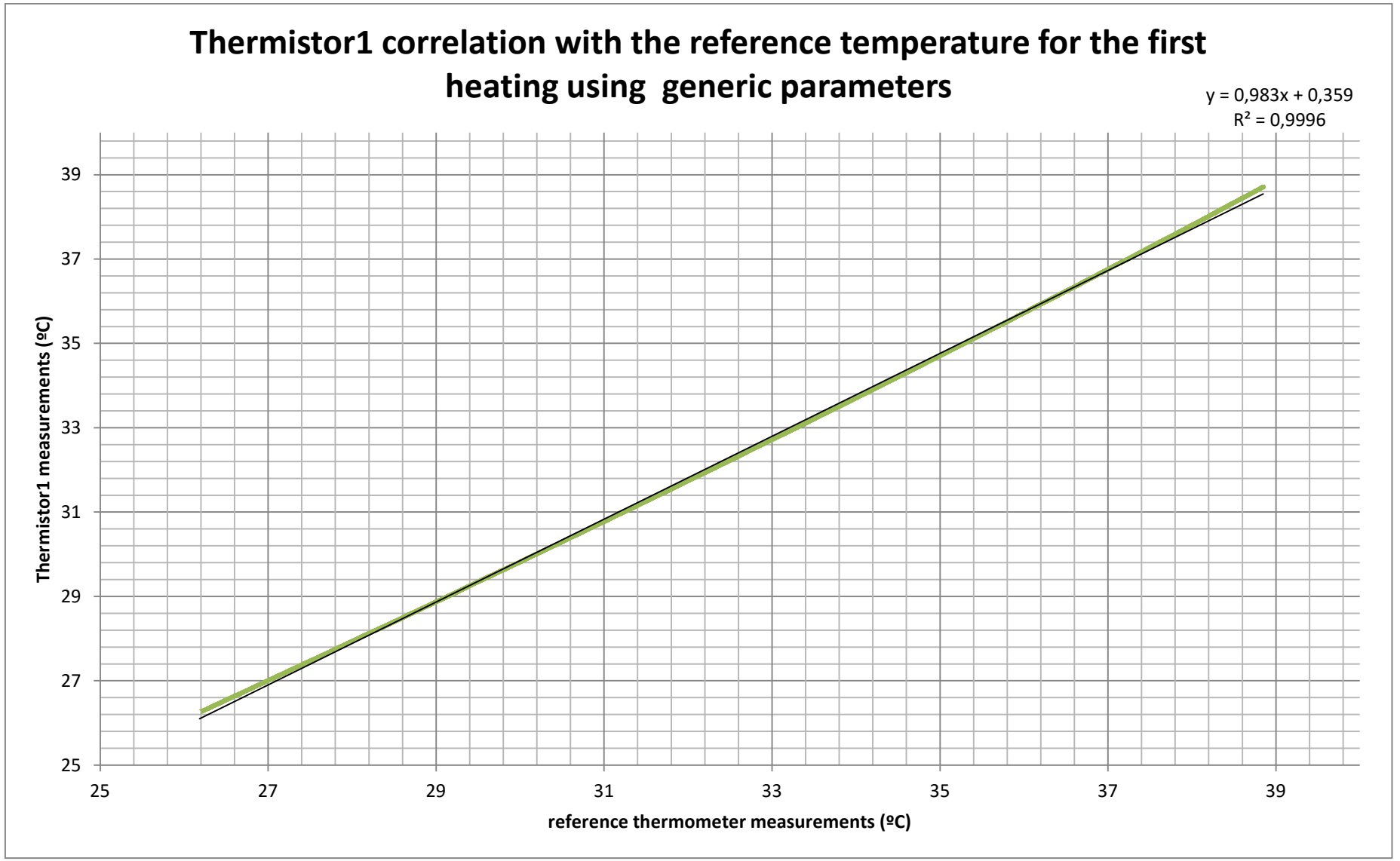


Figure 29: Graph displaying Thermistor1 correlation with the reference temperature for the first heating using a generic parameters.

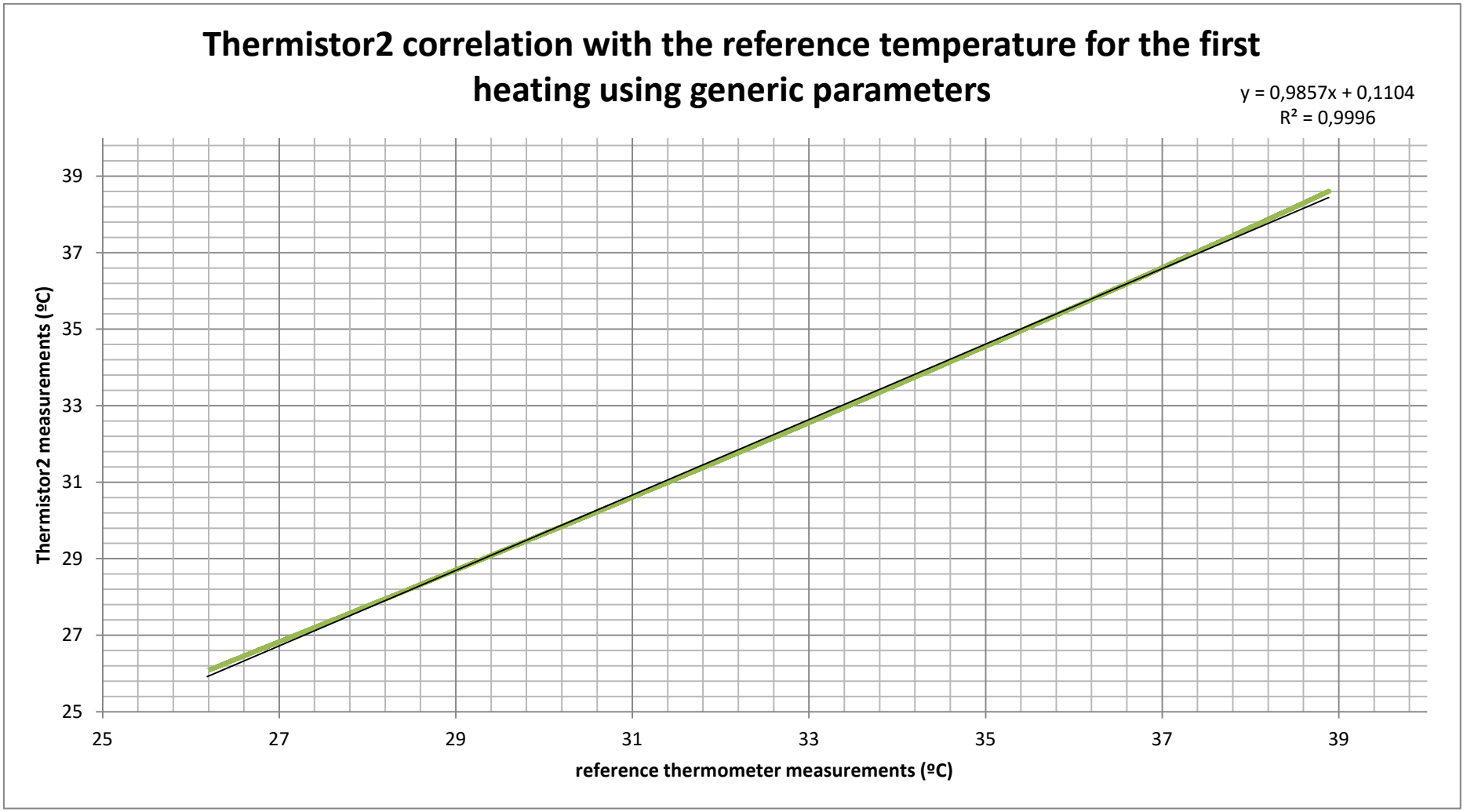


Figure 30: Graph displaying Thermistor2 correlation with the reference temperature for the first heating using a generic transformation.

6.4.1.5.3. Dallas DS18B20 IC sensors

The temperature registered by the two Dallas sensors plotted against the reference temperature can be seen in Figure 31 and Figure 32.

To all appearances both Dallas sensors have a good similar response. Although its quantification being visible, they have a good resolution of only 0,06ºC. Overall their response looks consistent and noise-free.

A linear regression was carried away for each both sensors, with the following results:

- Dallas 1: $T_M = 0{,}9872 \cdot T_R + 0{,}2147 \quad ;R^2 = 0{,}9999$
- Dallas 2: $T_M = 0{,}9889 \cdot T_R + 0{,}2146 \quad ;R^2 = 0{,}9999$

Where T_R is the real temperature, T_M is the measured temperature by the sensor and R^2 is the square of the correlation coefficient.

It can be observed that the coefficient multiplying T_M is near 1, which means that the sensors have a highly lineal response. For the thermistor 1 is 0,9872 and for thermistor 2 0,9889. It is worth noting that both values are really close to each other.

The offset has a value around 0,215, being 0,2147 for the first thermistor and 0,2146 for the second. Again, the offset value for both sensors is really similar.

The correlation coefficient is excellent, being 0,9999 for both sensors, which means that the response is highly lineal and it has low noise effects.

In general, even if in the offset field they are not the best, they have a great linear response. However, the most noteworthy point of these sensors is how similar both responses are. All coefficients are highly similar, which means that the user can expect the same performance from all units of this product, making them really reliable.

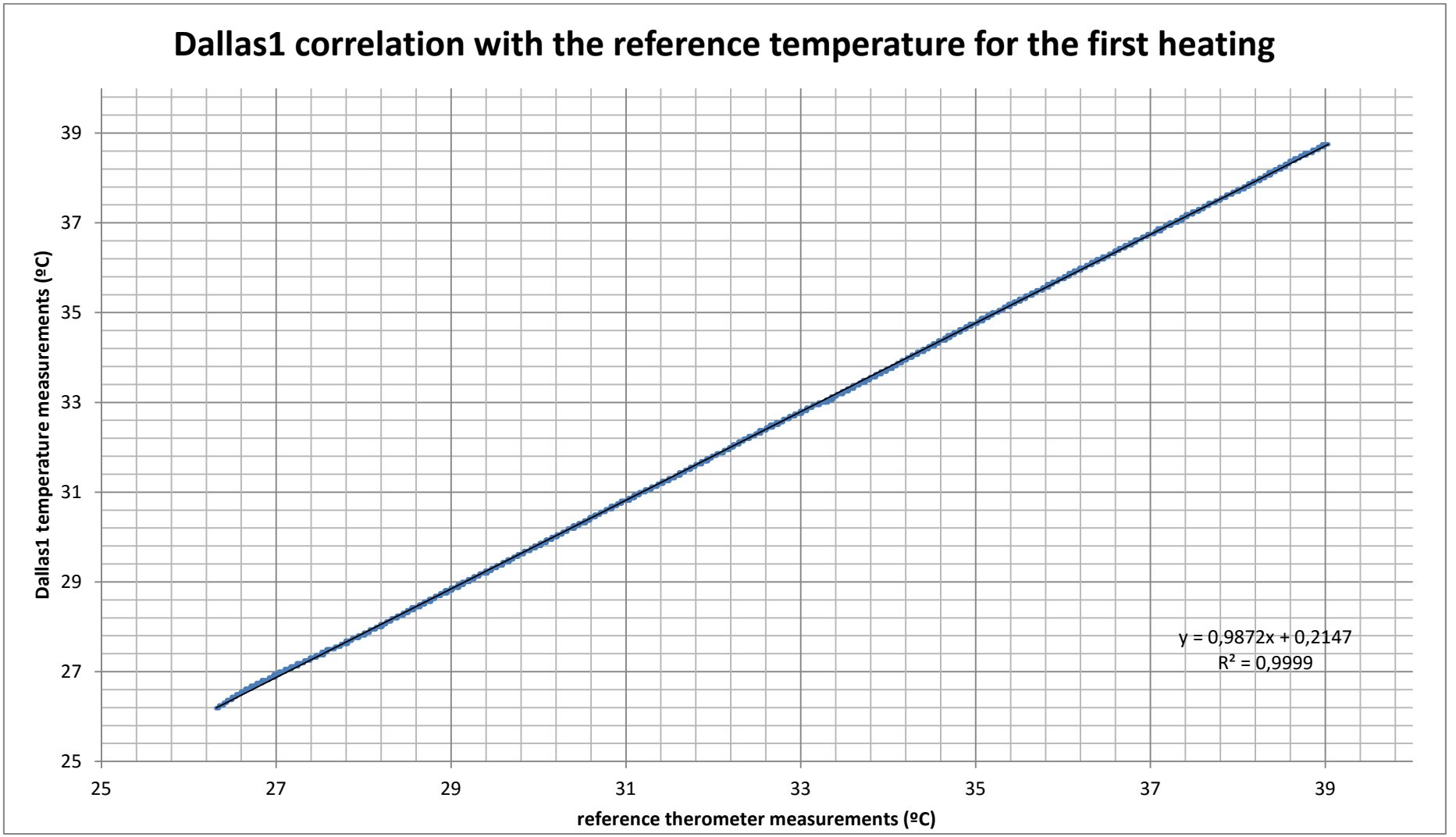


Figure 31: Graph displaying Dallas1 correlation with the reference temperature for the first heating.

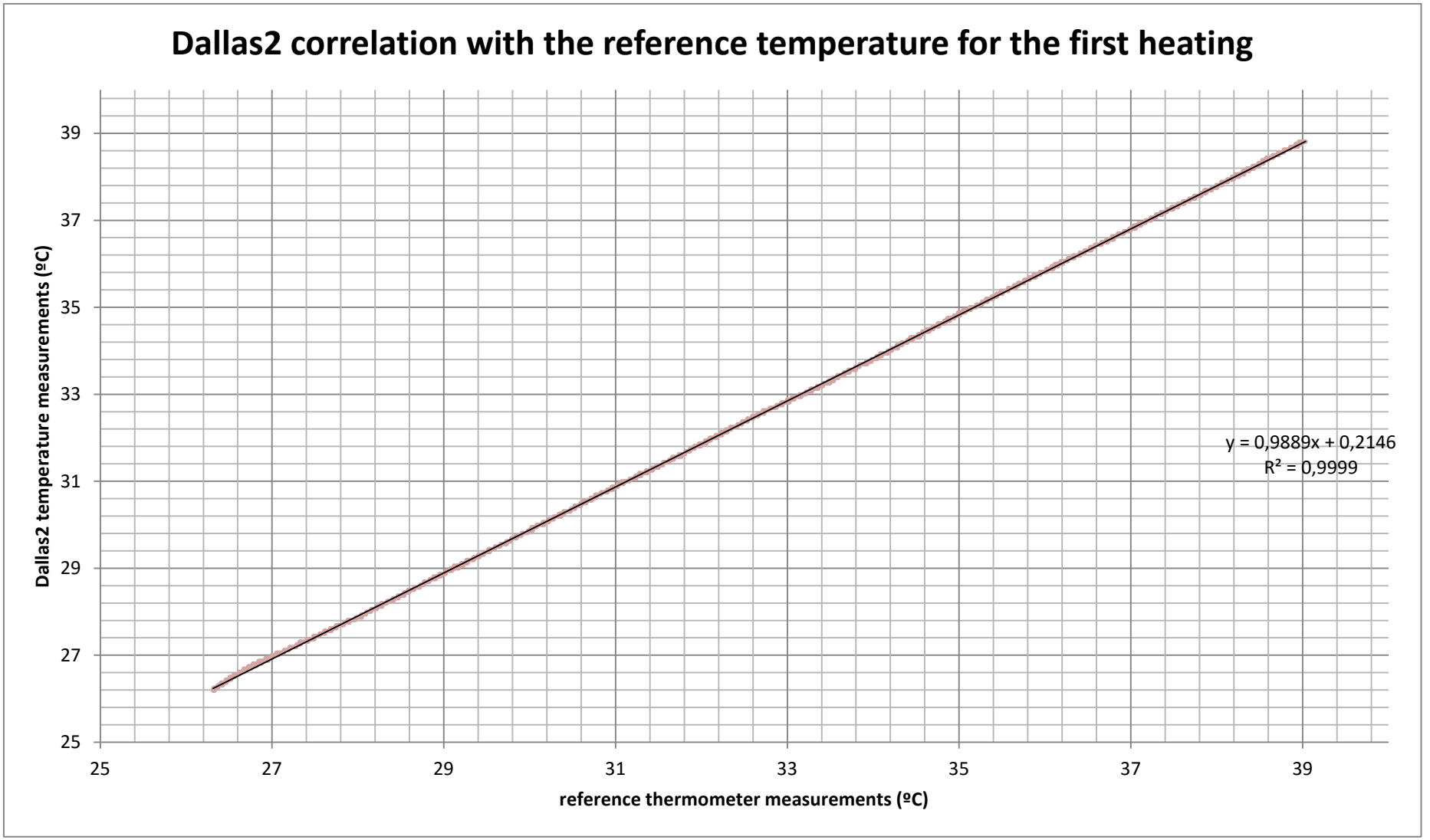


Figure 32: Graph displaying Dallas2 correlation with the reference temperature for the first heating.

6.4.1.6. Analysis of the results and comparison of sensors

For analysis and comparison purposes the error in the sensors' readings have been plotted in Figure 33. It must be mentioned that the error displayed for the thermistors is the one for the temperature obtained using the Steinhart-Hart equation with generic parameters. The error was computed as follows:

$$error = |T_R - T_M| \quad [°C]$$

Were T_R is the temperature measured by the reference thermometer and T_M is the temperature measured by the sensors.

The thermocouples using the MAX31850K amplifier ended up showing a worse response than expected, specially taking into account that it was the most expensive device. Even though its offset was the best for all types of sensors (although highly variant from device to device) and its linearity was not bad, the results varied highly depending on the sensor, which makes them less reliable.

Two of the thermocouples also had poor repeatability as it can be seen in Figure 33 where at time=3600s, with the change of conditions of the experiment (from heating to cooling), the response of thermocouple 1 and 3 changed drastically. Also they were the most affected by noise. It is suspected that both the noise and the repeatability could be produced due to the delicateness of the junction between the sensors itself with the MAX31850K, as it could not be soldered but had to be screwed due to the need of CJC.

The MAX31850K analog to digital converter also made this sensing system the one with the worst resolution of all. However this kind of sensors are really good for harsh environments and for high temperatures, where the other two types cannot operate. Overall, the MAX31850K is useful because it has all the signal manipulation devices built-in as well as 1-wire communication protocol capabilities, but it also does not take profit from the real potential of the thermocouple.

The thermistors on the other hand had a superior performance than anticipated. Being the cheapest devices bought, they had a good linearity, although their offset was the worst out of the three kind of sensors. The most remarkable facts are that they have a good repeatability, both units had a really close behavior and they had great resolution. Also it has been demonstrated that the Steinhart-Hart equation with generic parameters has good results. The main downside of this sensing system is the fact that it has no communication protocol, which means that for each thermistor desired in the system, an analog channel will be occupied; and that its output is analog and not digital, which means that it needs to be manipulated in order to make the data useful.

Finally, the Dallas DS18B20 has proved to be the one with a best performance. With a highly competitive cost and 1-wire communication protocol, it has a similar temperature range as the thermistors with a better linear response, resolution and offset. The most valued characteristics though were its great repeatability and consistency between both Dallas sensors, which had an almost identical response. This makes this device highly reliable as the user can expect the same behavior from all the units he buys.

However, one problem arose with the Dallas. Some days after the assay was carried away, they were mounted in a different system in order to use them, but they did not function and overheated. It is suspected that this malfunction was originated because the Dallas is encapsulated into a non-hermetic plastic capsule and the introduction of the sensor into the fluid environment of the experiment caused at long term a malfunction, despite the fluid was dielectric. In general the Dallas would be best suited for non-harsh environments.

In the table below the most noteworthy characteristics for each kind of sensor are shown:

	Thermocouples with MAX31850K	**Thermistors (using generic parameters for the Steinhart-Hart equation)**	**Dallas DS18B20**
Temperature Range (ºC)	-100 to +500	-80 to 120	-55 to 125
Resolution (ºC)	0,25	0,097	0,06
Linearity	Good (Worst)	Good	Good (Best)
Average offset	0,1642	0,2347	0,2146
Noise effect	High	Low	Low
Repeatability	Bad	Good	Good
Consistency between sensors	Bad	Good	Excellent
Communication Protocol	1-Wire	None	1-Wire
Capacity to work in harsh environments	Excellent	Good	Bad
Delicateness of the wiring	High	Lowest	Low
Cost per unit	19,67€ (including the necessary level shifter)	1,20€	2,70€

Table 7: Main characteristics of each contact sensor.

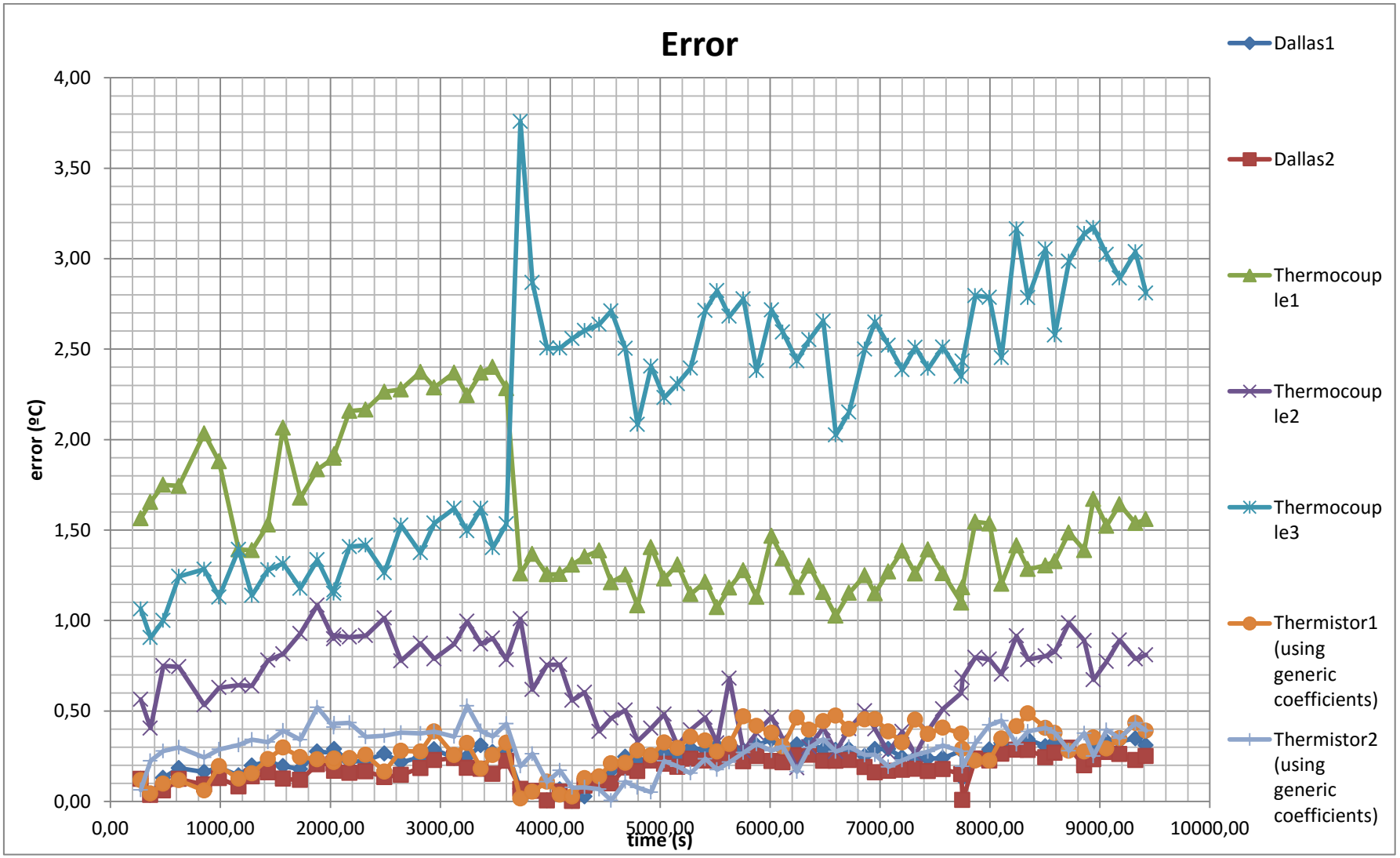


Figure 33: Graph displaying the error of the measurements plotted versus time.

6.4.1.7. Calibration

The linear regressions can be manipulated in order to be used as a calibration for the sensors, making possible to know the real temperature from the measured temperature:

$$T_M = A \cdot T_R + B \rightarrow T_M - B = A \cdot T_R \rightarrow T_R = \frac{T_M - B}{A}$$

This can be used in order to obtain better temperature readings from these sensors, as it linearizes the readings and minimizes the offset, assuming that the sensor has a repeatable behavior. Due to the fact that each sensor is unique, there's one unique calibration for each sensor and a calibration from one sensor is not useful applied to another sensor (even if they are from the same type).

In the next graphs shown from Figure 34 to Figure 40, the error of the calibrated data (blue) is plotted versus the error of the non-calibrated data (red). In the case of the thermistors, the non-calibrated data plotted is the temperature measured by using the Steinhart-Hart equation with generic parameters (red), while there are two calibrated data: the one using this previous data (blue) and another which uses directly the raw digital data (green). It must be noted that the data from the cooling process has been discarded as useful for the reasons presented at section 6.4.1.5. Results.

For the thermocouples, shown in Figure 34, Figure 35 and Figure 36; it can be seen that the error has been diminished by overt 1ºC for thermocouples 1 and 3 (the average error has diminished from 1,78ºC to 0,4ºC for thermocouple1 and from 1,9ºC to 0,63ºC for thermocouple3) and 0,68ºC for thermocouple2 (from 0,8ºC to 0,12ºC).

It must be noted that due to the poor repeatability and high noise of thermocouples 1 and 3, the effectiveness of the calibration is diminished for the second heating, where the behavior has changed radically compared to the first heating, making the average error after calibration worse. Even though, the calibration is still beneficial even for the second heating.

The thermocouple2's after-calibration error is highly interesting as it is, on average, lower than the sensor's resolution which means that it is lower than the quantization error. Ultimately this means that the effect of noise in this sensor is not affecting the final signal as it is covered by the quantization error, which is higher, making the overall precision not worse than 0,25ºC.

The thermistors error can be seen in Figure 37 and Figure 38. It can be seen that due to its repeatability they have a solid and similar performance for the first and second heating. This leads to better calibration results. What is most interesting for these sensors is the comparison between the error obtained by applying the calibration of the raw data and the error obtained by applying the calibration of the temperature data obtained through the Steinhart-Hart equation using generic parameters.

The original average error was 0,26ºC and 0,35ºC respectively for thermistor 1 and 2. Using the calibration of the raw data the average error was diminished to 0,10ºC and 0,08ºC respectively while using the calibration of the temperature data obtained through the Steinhart-Hart equation using generic parameters the average error was diminished to 0,07ºC and 0,07ºC respectively. So overall it is more beneficial to apply the calibration over the corrected data using the Steinhart-Hart equation. This is also beneficial due to the fact that using the Steinhart-Hart equation the non-linearity of the thermistor's response is corrected, which diminishes hugely the error when working on wide temperature ranges.

Similarly to thermocouple2's after-calibration error, the calibrated error of the temperature data obtained through the Steinhart-Hart equation using generic parameters is lower than the sensor's resolution, making the overall precision of these sensors not worse than 0,097ºC.

Finally the Dallas' error can be seen in Figure 39 and Figure 40. The original error of these sensors was already low (0,25ºC for the Dallas1 and 0,18 for the Dallas2) being the lowest of all types of sensors. After calibration, this average error was diminished to 0,04ºC for both Dallas, which is a great precision. Again as with the thermocouple2 and the thermistors, the after-calibration average error is lower than the resolution, which means that the error comes from the quantization error and not from noise.

Overall, the application of calibration over these sensors improves significantly its precision, making most of the sensors useful for high precision applications (below 0,1ºC precision). Bad repeatability leads to a worse improvement of the response, although it still is beneficial. The improvement of the measurements is so good that the average error is below the sensor's resolution for most cases. The Dallas have proven to be the most precise, before and after the calibration.

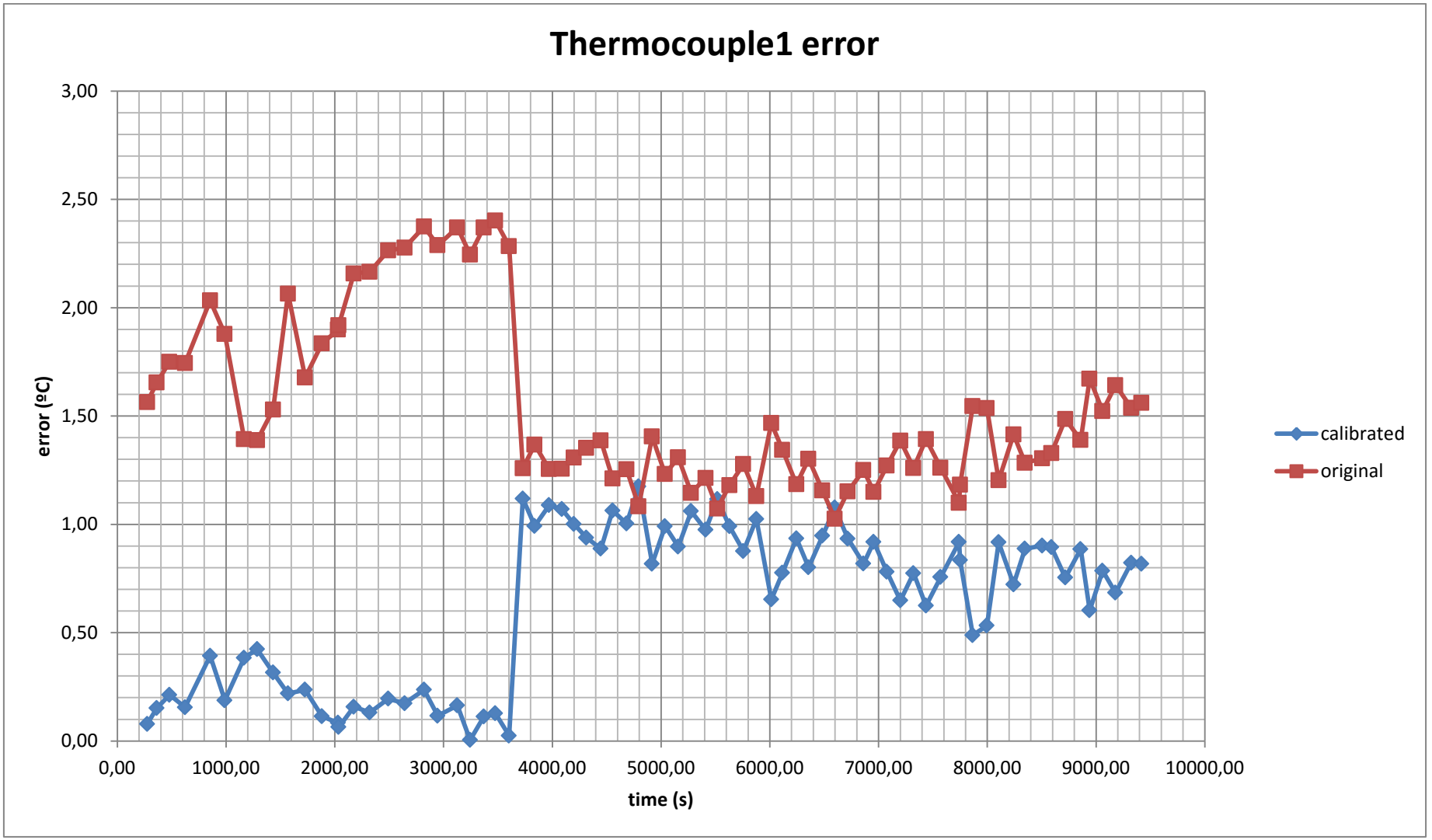


Figure 34: Graph displaying Thermocouple1's original error and after-calibration error versus time.

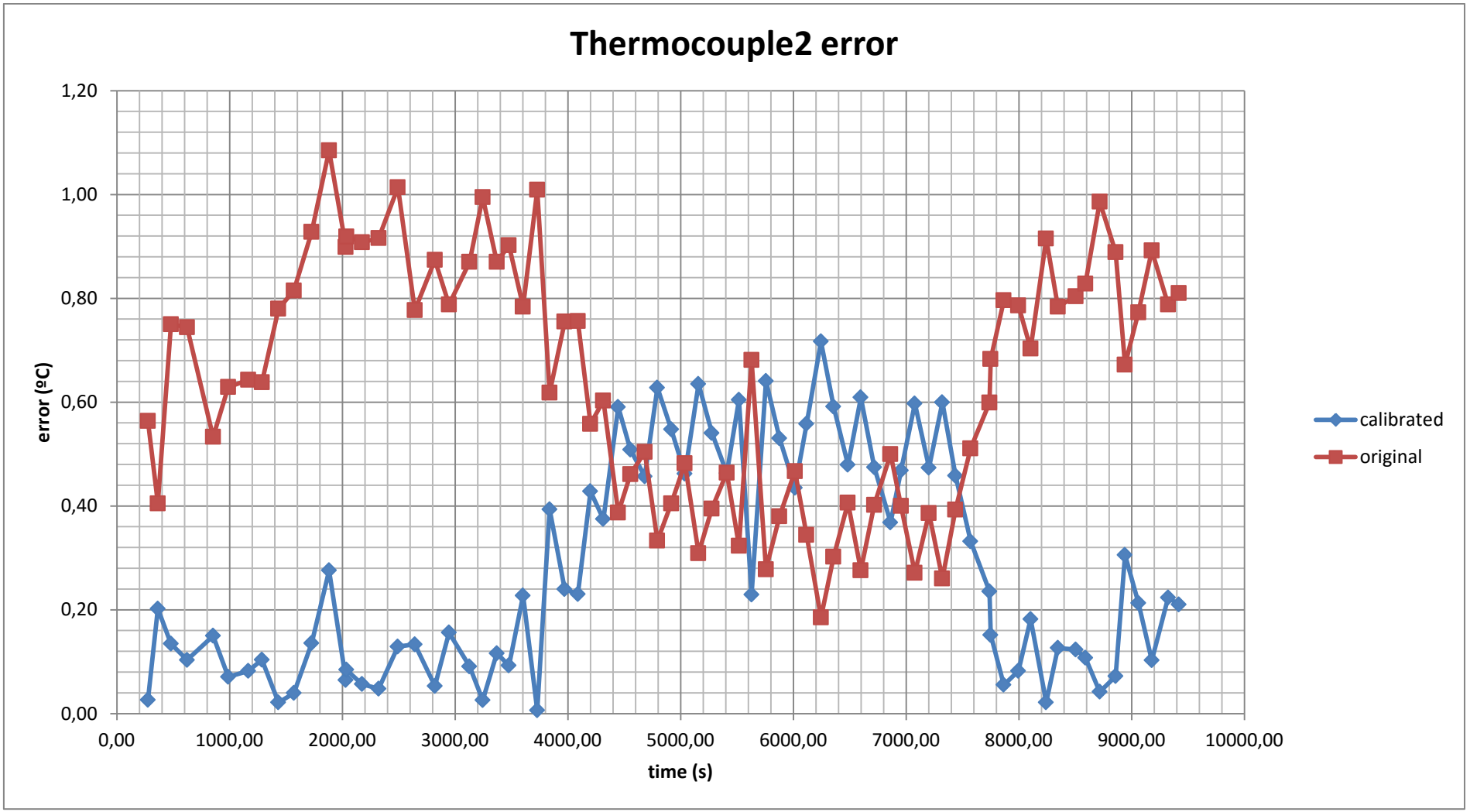


Figure 35: Graph displaying Thermocouple2's original error and after-calibration error versus time.

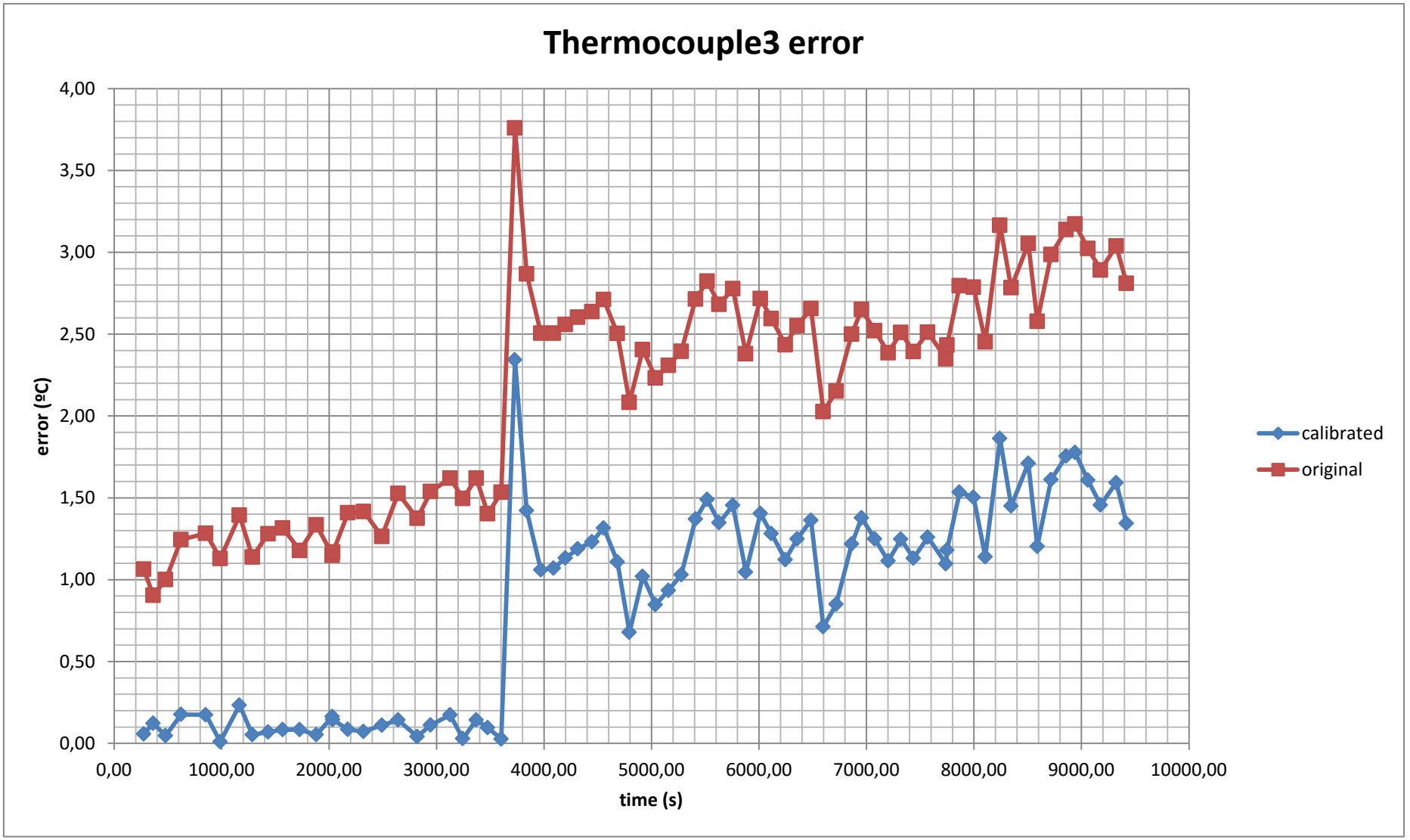


Figure 36: Graph displaying Thermocouple3's original error and after-calibration error versus time.

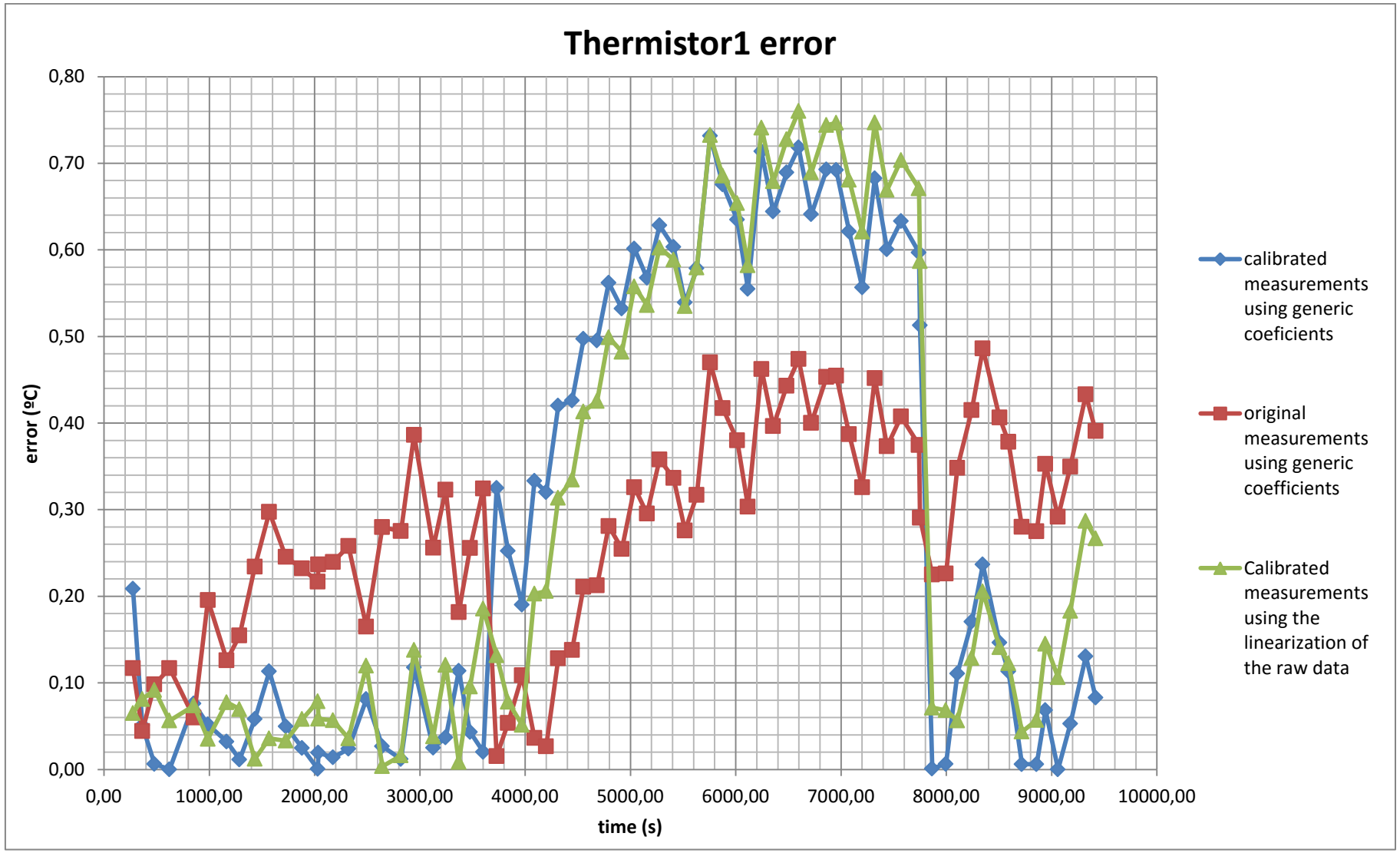


Figure 37: Graph displaying Thermistor1's original error (using the Steinhart-Hart equation with generic parameters), after-calibration error using the Steinhart-Hart equation and after-calibration error using the raw data directly versus time.

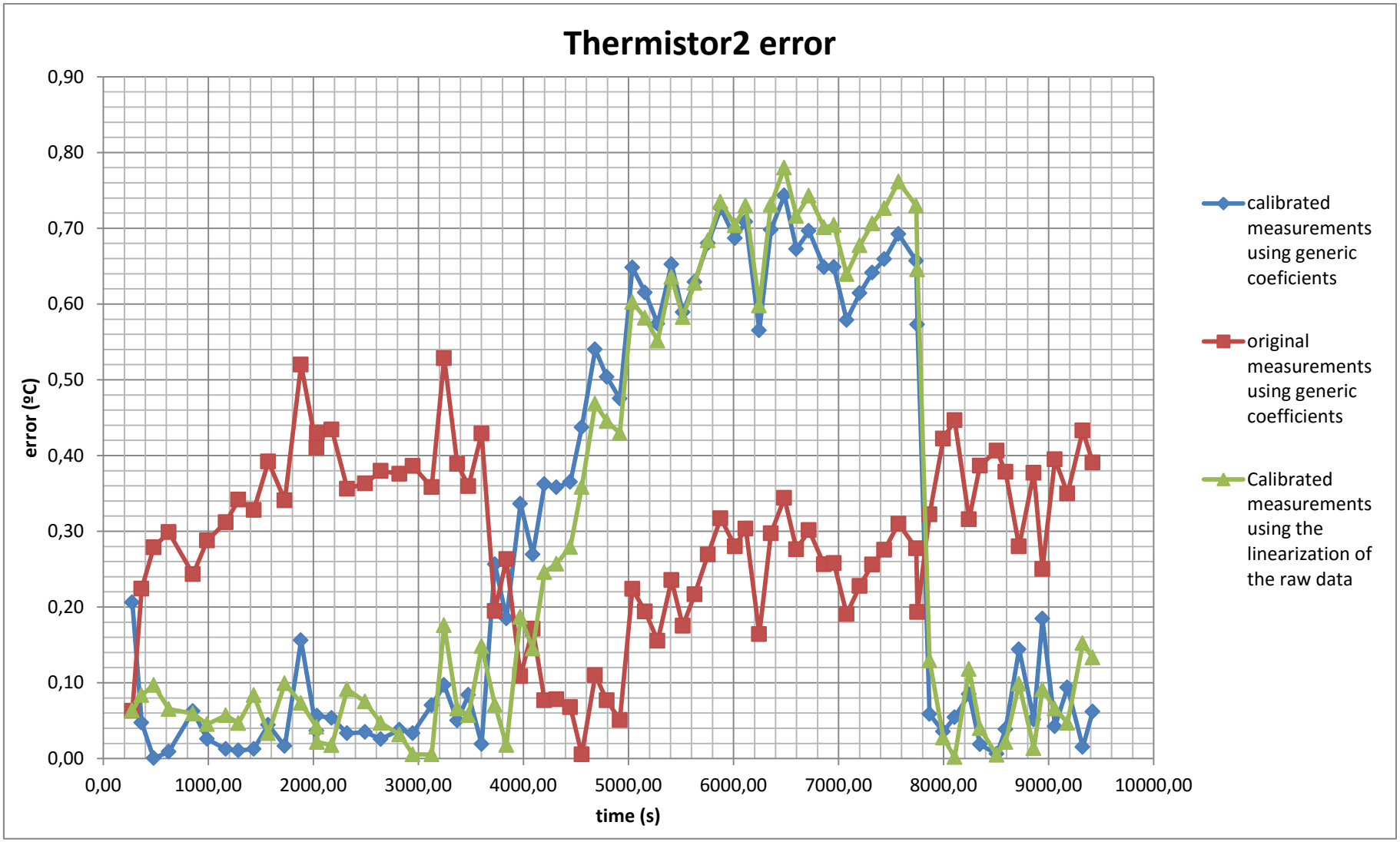


Figure 38: Graph displaying Thermistor2's original error (using the Steinhart-Hart equation with generic parameters), after-calibration error using the Steinhart-Hart equation and after-calibration error using the raw data directly versus time.

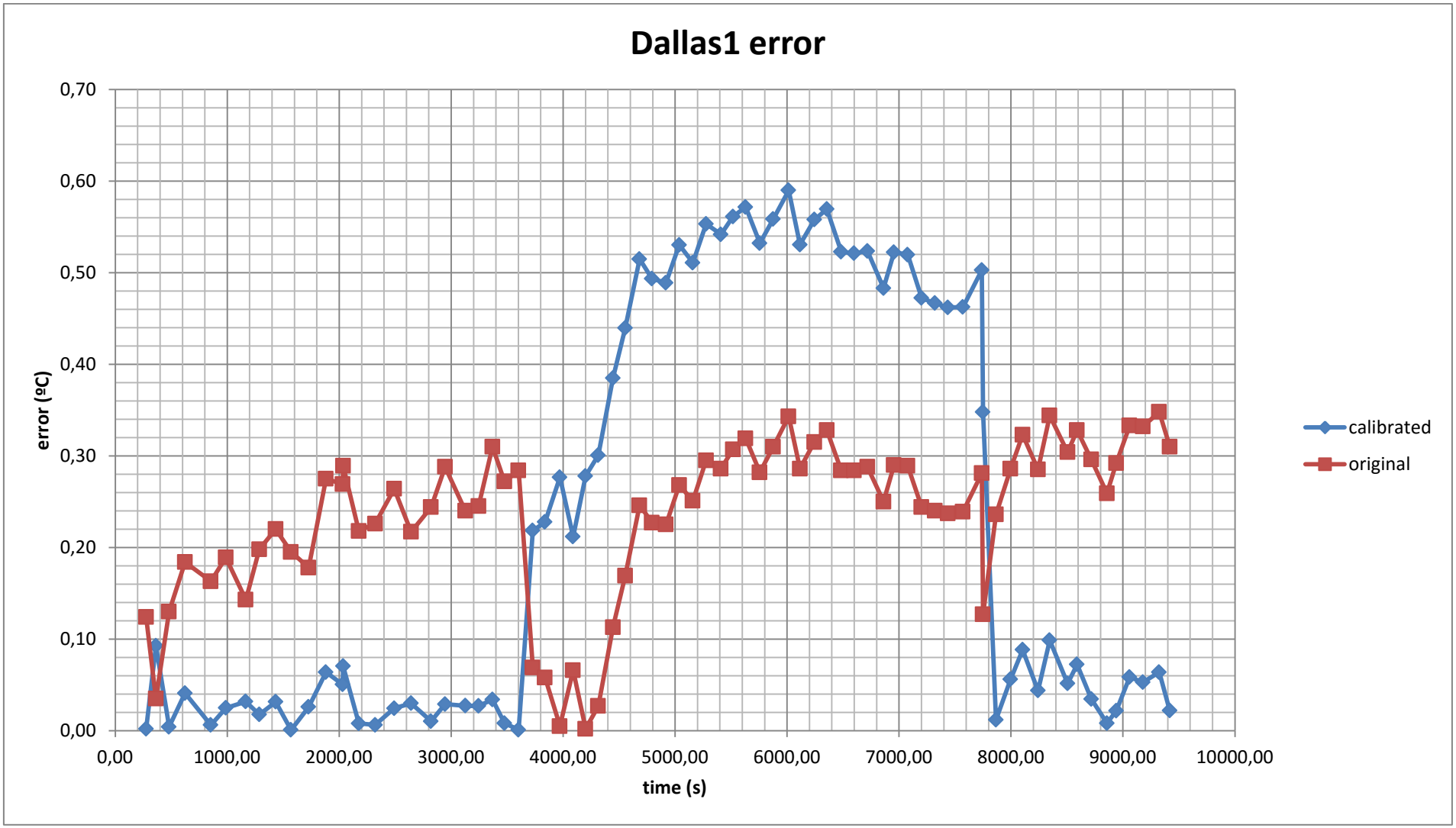


Figure 39: Graph displaying Dallas1's original error and after-calibration error versus time.

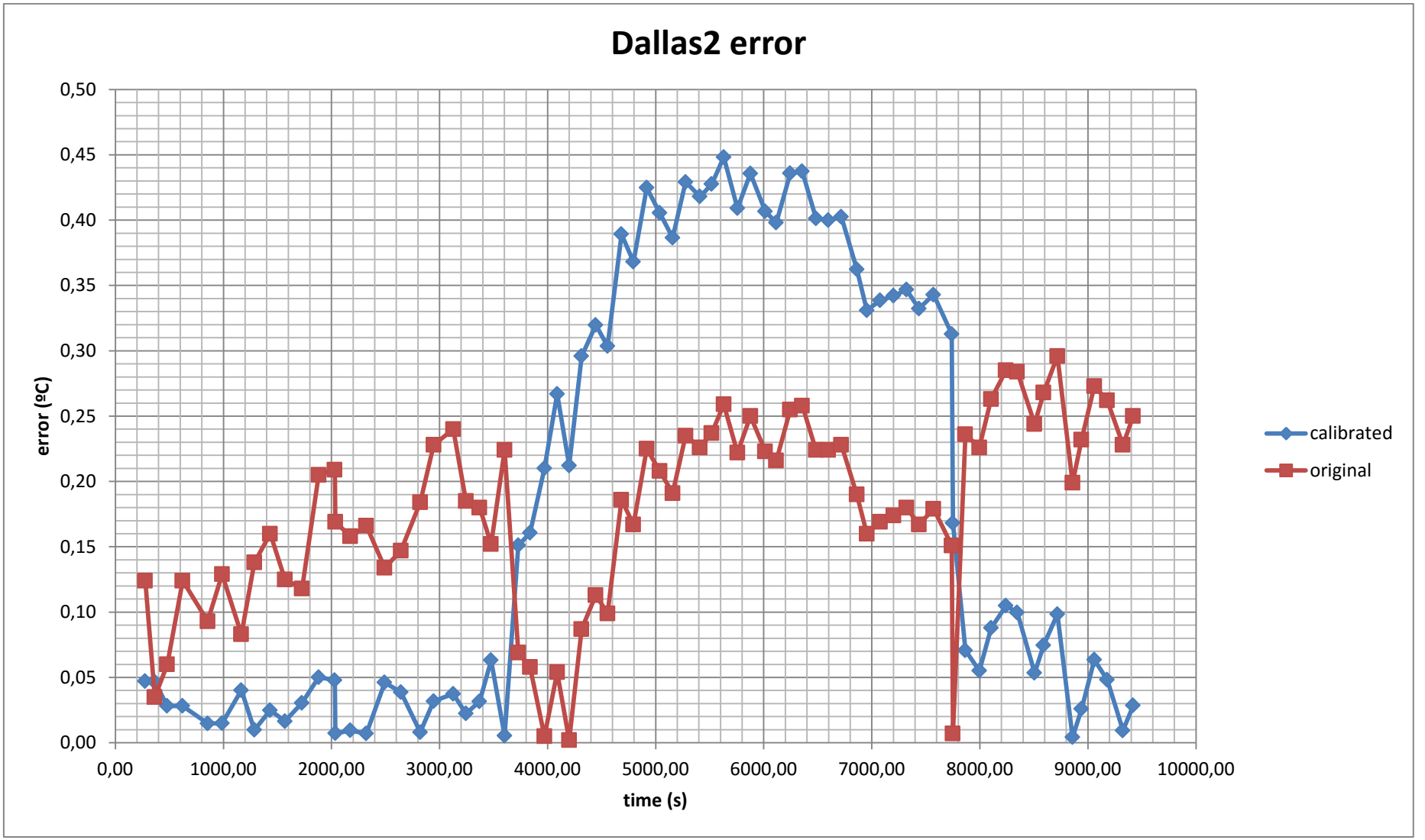


Figure 40: Graph displaying Dallas2's original error and after-calibration error versus time.

6.4.2. ASSAY OF THE INFRARED SENSOR

6.4.2.1. Methodology

The IR sensor measures the temperature of a surface. The temperature measured not only depends on the temperature of the surface but also on the emissivity of the surface's material. This led to two difficulties: first of all, being able to track the real temperature of the surface with calibrated contact sensors and secondly, ensure that the surface had the desired emissivity

The method used to do the assay was based on heating a heater plate and reading its surface temperature, doing a temperature sweep. However, the emissivity of the heater itself was not known so it was covered with two layers of black PVC electrical tape whose emissivity is known to be 0,95, the same as presented in the IR sensor.

In order to know the real temperature of the surface, two of the sensors previously calibrated were used. One of them would be between the heater plate and the first layer of tape and the second one between the first and second layer of tape. They were also placed in different positions in the field of view of the IR sensor. The idea was that in order to validate the assay both reference temperatures should be close to each other, which would mean that there would not be a temperature gradient between tape layers as well as that all the surface under the field of view of the IR sensor would be isothermal.

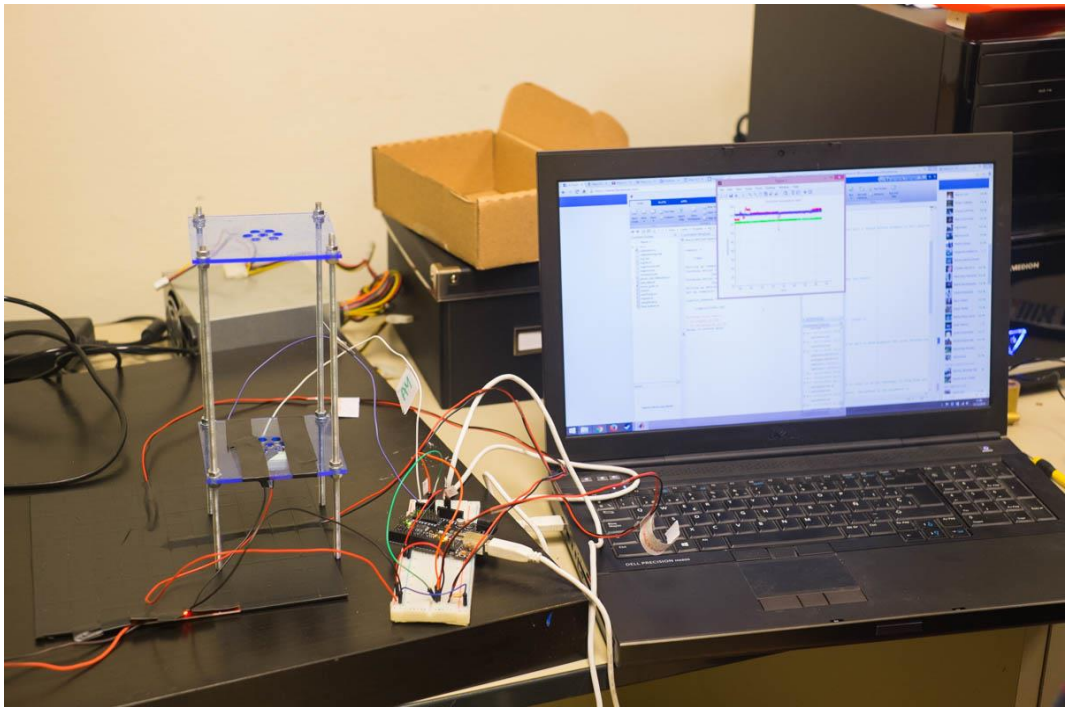


Figure 41: Photograph of all the material set up for the assay of the IR sensor.

As it can be seen in Figure 42, when the plate was heating, both reference sensors would display a different temperature (being the measurement of the sensor in direct contact with the plate higher than the one that was placed between the tape layers), so this experiment was considered invalid. The data for this experiment can be found at annex section 2.2.1. DATA FORM THE FIRST EXPERIMENT.

In order to work this problem around what was done was to do the assay without using the heater, so the surface temperature would fluctuate thanks to natural convection with the room temperature the experiment was being held in. This meant that the experiment was carried away in a much smaller temperature range, but the data obtained was valid as both reference sensors measurements were highly similar.

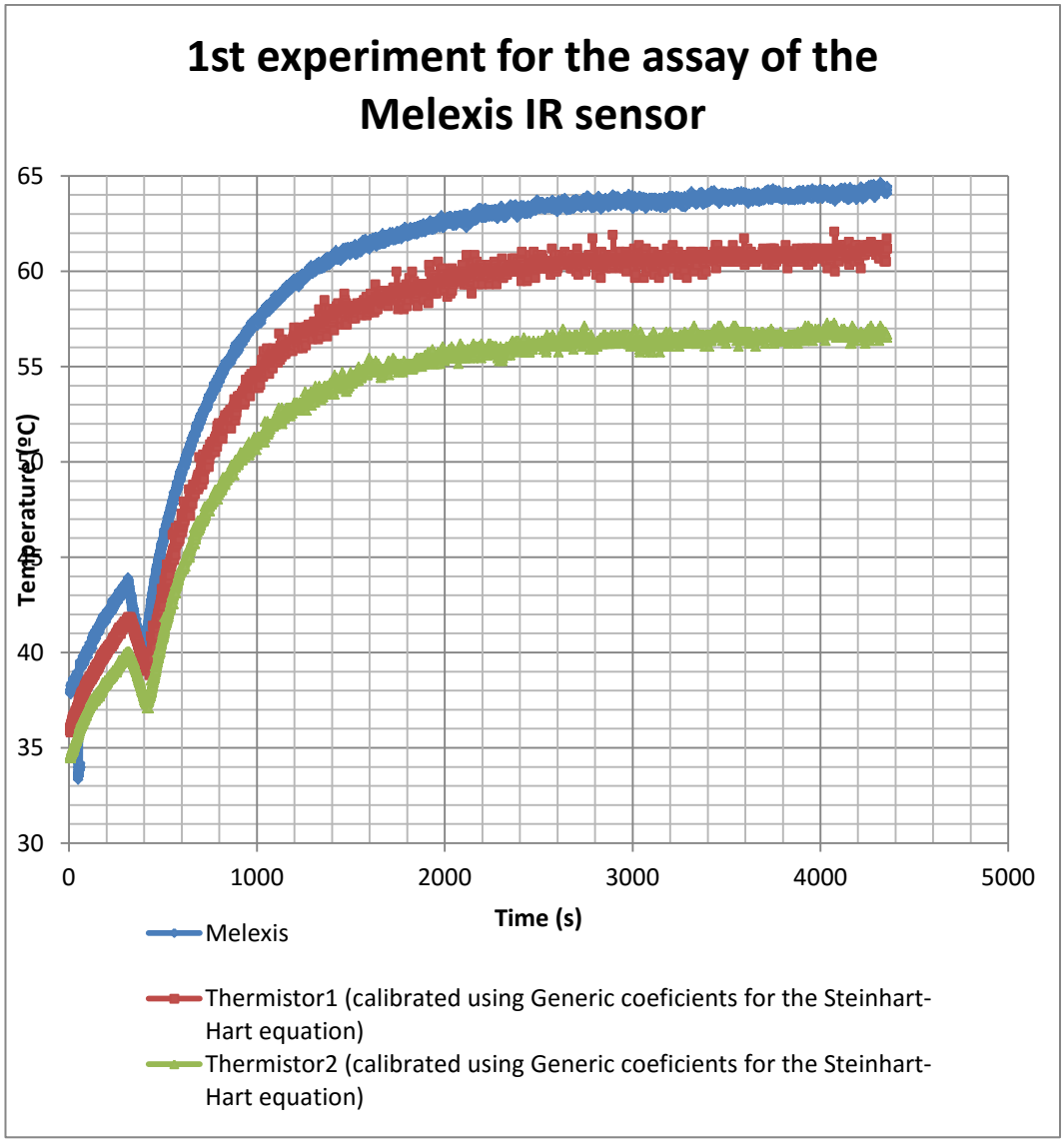


Figure 42: Graph displaying the data of the 1st (invalid) experiment for the assay of the Melexis IR sensor

6.4.2.2. Material

The heater plate used was a PCB Heatbed MK1 (Figure 43), a heater plate usually used as a component for 3D printers. It was chosen due to its low cost (it was acquired for 12,40€), good temperature range (up to 110ºC) and relatively big surface area, which ensures good isothermality at its central area [30].

The black PVC electrical tape used was the Tesa53948.

The IR sensor tested was the Melexis 90614-ACF.

The reference sensors elected were the thermistors because they had a good response after calibration and they were smaller than the Dallas, which made easier to place them between tape layers.

The same structure used for the contact sensors assay was used in order to hold upside-down the IR sensor. Due to the fact that the platforms were held using nuts to a threaded rod, the height at which the IR sensor was could be adjusted.

Figure 43: Photograph of the PCB Heatbed MK1 heater plate.

6.4.2.3. Wiring of the sensors with the Arduino

The wiring of all the sensors to the Arduino would be the superposition of the previously presented wirings for the thermistors and the Melexis sensor at section 6.2. SENSORS and can be seen in Figure 44.

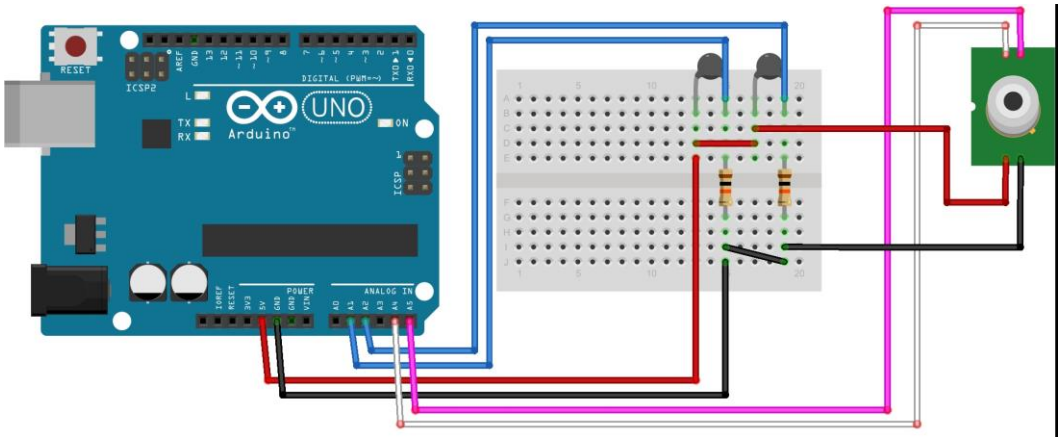


Figure 44: Wiring of the sensors to the Arduino for the assay of the IR sensor.

6.4.2.4. Software

The software which had to be developed for this assay had the same requirements as the contact sensors assay's software, so it only had to be slightly adapted.

6.4.2.4.1. Arduino code

The code for Arduino is a combination of the codes for thermistors and Melexis sensors which are explained at section 6.3. SOFTWARE FOR INTERFACING THE ARDUINO AND THE SENSORS, with some extra instructions in order to interface with matLab. The full code can be found at the annex section 1.2.2.1. Arduino code.

It is basically the same program as for the contact sensors assay but sending to the computer only the thermistors' readings and including the melexis'.

6.4.2.4.2. Matlab code

The matLab code is exactly the same as in the previous assay, with the slight difference that instead of reading seven sensors it was programmed in order to read only three,

which meant slight changes in the main real-time loop and the **ReadTemp(*s*)** function. The full code can be found at the annex section 1.2.2.2. MatLab code.

6.4.2.5. Results

The data obtained for the second experiment was carried away in a 3 and half hours run due to the impossibility to make fast changes to the ambient room temperature. The graphs displaying the data can be found at Figure 45. The full data obtained can be found at the annex section 2.2.2. DATA FROM THE SECOND EXPERIMENT.

At first sight it can be seen that the difference in measurements between the two thermistors is minimal (it must be added that the measurements are calibrated after using the Steinhart-Hart equation), which makes this experiment valid. The Melexis IR sensor shows a positive offset but has a good behavior.

The Melexis IR sensor measurements were plotted against the average temperature measured between the two thermistors, which can be seen at Figure 46. A linear regression was carried away:

A linear regression was carried away for each one of the three thermocouples, with the following results:

- $T_M = 1{,}0103 \cdot T_R + 0{,}0548 \quad [°C] \qquad ; R^2 = 0{,}9831$

Where T_R is the real temperature, T_M is the measured temperature by the sensor and R^2 is the square of the correlation coefficient.

The Melexis has a good proportionality coefficient with respect to the reference temperature of 1,0103. It's offset is actually low, with a value of 0,0548. Figure 45 may induce to think that the offset should be higher, but due to the huge amount of data points displayed it can be a bit deceptive actually.

Overall, the linearity of the response is excellent, although the measurements fluctuate slightly compared to the reference temperature, which is caused because the IR sensors detect almost instant changes in the surface temperature while the thermistors used as reference thermometers do not because of their thermal inertia.

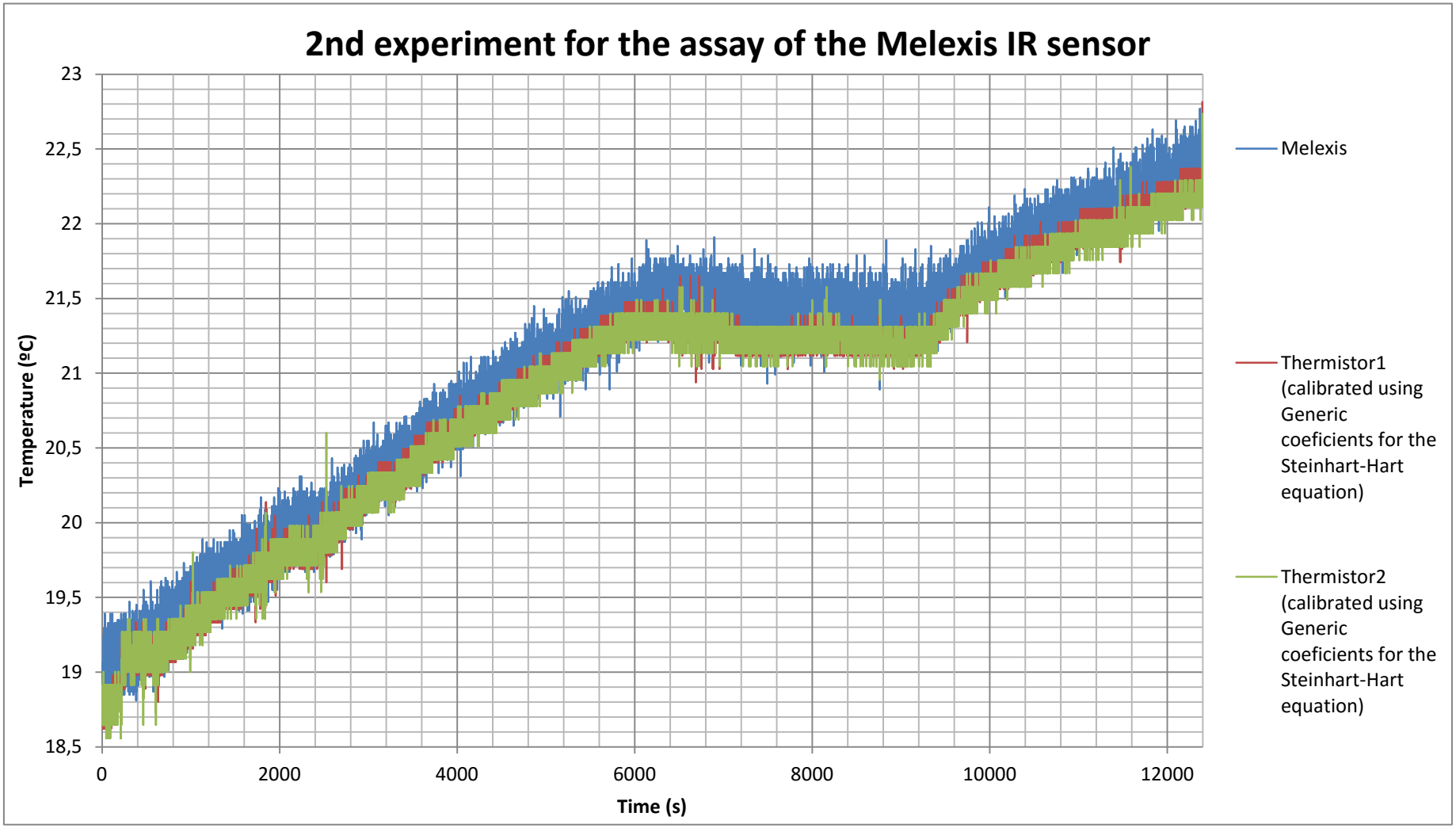


Figure 45: Graph displaying the data for the 2nd experiment for the assay of the Melexis IR sensor.

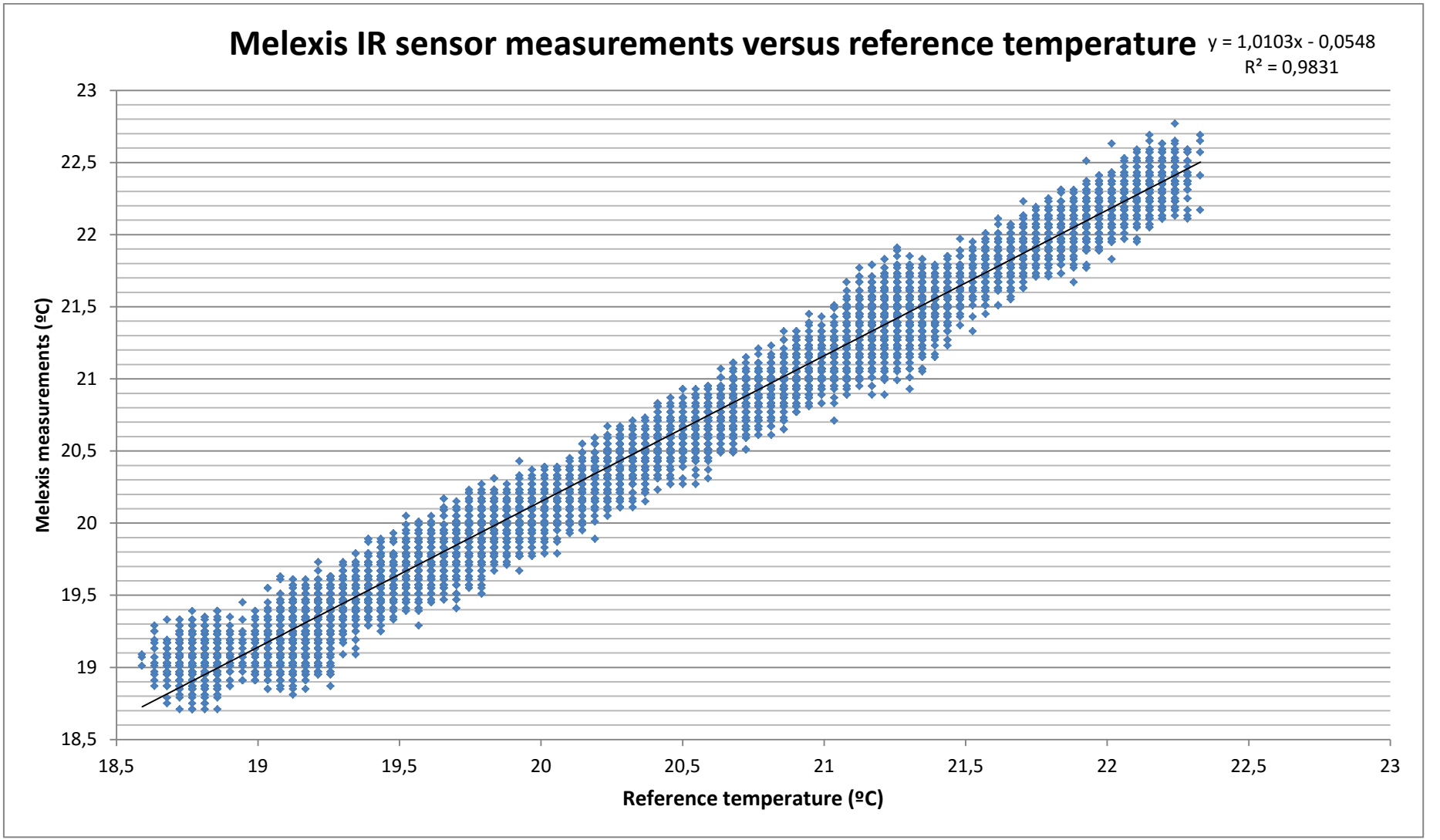


Figure 46: Graph displaying Melexis IR sensor measurements versus reference temperature.

6.4.2.6. Analysis of the results and comparison of sensors

The Melexis 90614-ACF has an excellent linear response. It is highly accurate, being able to compete with the contact sensors in terms of precision at a competitive cost of 17,45€. Even if it is a higher cost than the best contact sensors (the thermistors and the Dallas), it has added benefits due to its nature, specially the fact that it can sense temperature from a distance which makes it useful in situations where the contact sensors just would not be able to operate. It also is the sensor that can detect temperature changes faster due to the fact that it is not bond to thermal inertia and has the widest temperature range of the sensors studied, with a -70ºC minimum and 340ºC maximum..

6.4.2.7. Calibration

Similarly to the contact sensors, the linear regression can be manipulated in order to be used as a calibration for the Melexis sensor, making possible to know the real temperature from the measured temperature:

$$T_M = A \cdot T_R + B \rightarrow T_M - B = A \cdot T_R \rightarrow T_R = \frac{T_M - B}{A}$$

In Figure 47, the error of the calibrated data (blue) is plotted versus the error of the non-calibrated data (red). Due to the high density of data points, the data points displayed for this graph are 1200 randomly selected points of the whole data from the experiment, in order to make it easier to visualize.

Even when not calibrated, the Melexis IR sensor has an excellent response with less than 1ºC of error. In general, it can be seen that after calibration the error has diminished, although due to the low original error it is not as beneficial as in other cases. The average error has diminished from 0,1723 to 0,1063ºC.

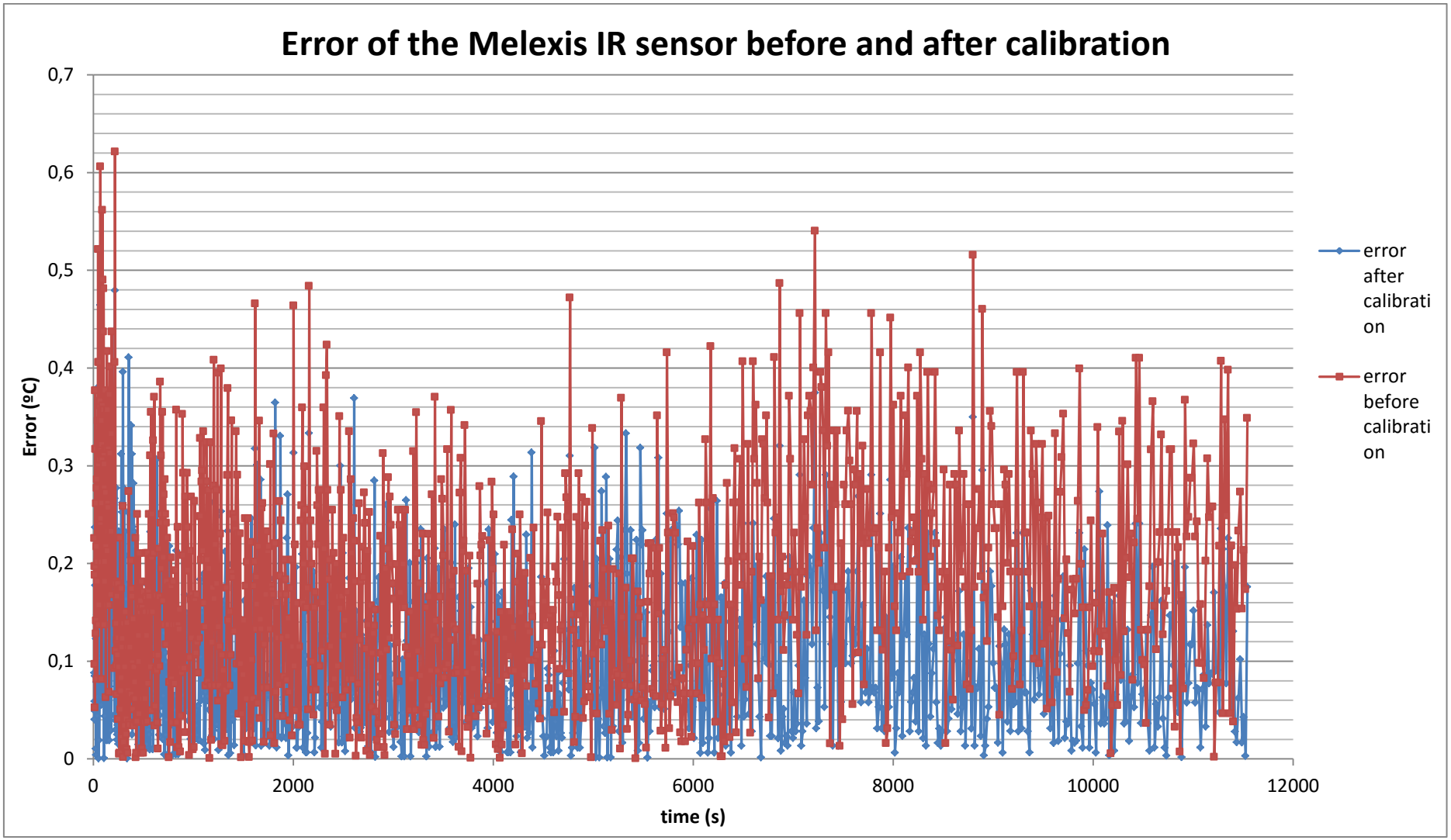


Figure 47: Graph displaying the error of the Melexis IR sensor before and after calibration.

7. DEVELOPMENT OF APPLICATIONS

For this study two applications were developed up to a functional prototype stage. Both applications were of aeronautical nature: the first one is a module for helium balloons that takes temperature samples from inside the module, the atmosphere and IR temperature scans from the sky above the balloon and the second one is a module intended to be installed in drones which enables them to make a temperature map of a terrain.

7.1. HELIUM WEATHER BALLOON TEMPERATURE MODULE

The first application developed was a module designed to be installed in a weather helium balloon which would take different temperature measurements that, coupled with the altitude information, provide useful information for weather prediction and atmosphere monitoring.

The first of the measurements was the outside atmospheric temperature, taken with a contact sensor. This information is usually used in order to know the temperature gradients in the atmosphere at different heights in order to predict the forecast [25]. It was also interesting as the data can be easily compared with the ISA atmosphere model. The second was an IR temperature measurement of the sky above the balloon. This temperature can be useful for the study of the greenhouse effect [31], sky transparency and cloud detection [32]. For this study it was used as a test to the measures at high altitude, where the temperature measured should be near the absolute zero as the IR sensor would be pointing to the space with minimal interference of the atmosphere. The last temperature was the temperature inside the module itself, useful in order to improve the isolation and heating systems of the module, as it will be explained afterwards.

7.1.2. THE NESLAB PROJECT

The development of this application would not have been possible without the aid of a student-project based at ETSEIAT University named Neslab. This student-project develops experiments and launches them with helium balloons up to high altitudes (around 30Km height). The structures, tracking system and launching system are also self-developed.

The project offered to launch this application on one of their launches held on the 22nd of December of 2014. This made possible the test of the module in a real environment of operation, where the data showed in section 7.1.7. was obtained. In order to keep

the weight of the whole payload low (the module developed in this section was launched along with other experiments) the electrical power used by the module developed in this work as well as the altitude information would be supplied by the modules developed by the Neslab team.

7.1.3. ELECTRICAL POWER AND INSULATION

Even if the power system was developed by the Neslab team, it is worth to explain it as it is a critical part of the whole launch. One of the main problems when carrying electrical devices to high altitudes is its batteries. Even if the equipment has a low electrical requirement, batteries capacity hugely diminish when exposed to cold environments. Some sensors can also malfunction due to cold conditions.

In minimize this problem, active and passive measures were taken. A system of electrical resistances was installed which would heat up the interior of the structure were the batteries and electronics were held. Also, the structure itself was made of foam which diminished the heat loss due to conduction and the interior and exterior surfaces were covered with thermal blankets to minimize radiation heat transfer.

The Arduino can be electrically supplied through its input power pin, which accepts a voltage between 6 and 20V (although it is recommended to use a 7 to 12 voltage), and internally it converts this voltage to the 5V used for all the Arduino functionalities. However, it can also be fed through its 5V output but then the input voltage must be 5V as the Arduino does not have any voltage regulator if that port is used as an input. For this project, this second method was used because even though it is more delicate, it does not waste energy transforming the voltage to 5V which means less electrical consumption. Due to the cold problem, the voltage delivered by the batteries would not be a stable value but fluctuate, for this reason a voltage controller was installed in order to ensure a 5V stable power bus.

For power redundancy, two sets of batteries were installed. Both would be able to supply the 5V needed to power the Arduino even at cold conditions. One of the batteries would rest unused and switch on in the event of a failure of the other one. The system was tested in a cooler in order to corroborate that the batteries would provide the electrical power required.

7.1.4. DATA LOGGING

It was set as a requirement for the applications that they had to be standalone. In order to carry out this requirement, the devices needed to have data logging capabilities. For this application this requirement is evident as weight and electrical consumption are a critical element of the module design which makes impossible to include a whole computer for that matters. One way to cover up this necessity is adding to the system an SD logging device, which gives the capacity to the Arduino to save the sensors data in one or more text files into an SD or micro SD card.

7.1.4.1. Wiring for data logging with SD cards

Most SD cards use SPI communication protocol, which means that they need to be plugged to the Arduino pins which support this communication protocol (digital pins 10 to 13). However, the connection between the Arduino and the SD card logger cannot be direct due to the fact that SD card logger pins work on a 3,3V logic level while the Arduino's work on a 5V, which makes the use of a level shifter compulsory. Adafruit's BSS138 4-channel I2C-safe Bi-directional Logic level converter was used for this purpose, as it already have 4 independent level shifting channels. The wiring is shown in Figure 48 and goes as follows:

- Arduino ports 10, 11, 13 and 12 are plugged to the B1, B2, B3 and B4 pins of the level shifter respectively.
- A1, A2, A3 and A4 ports from the level shifter are connected to the SS, MOSI, SCK and MISO ports of the SD card data logger respectively.
- The 5V and 3,3V supply and the GND pins form the Arduino are plugged to the HV, LV and GND ports of the level sifter respectively.

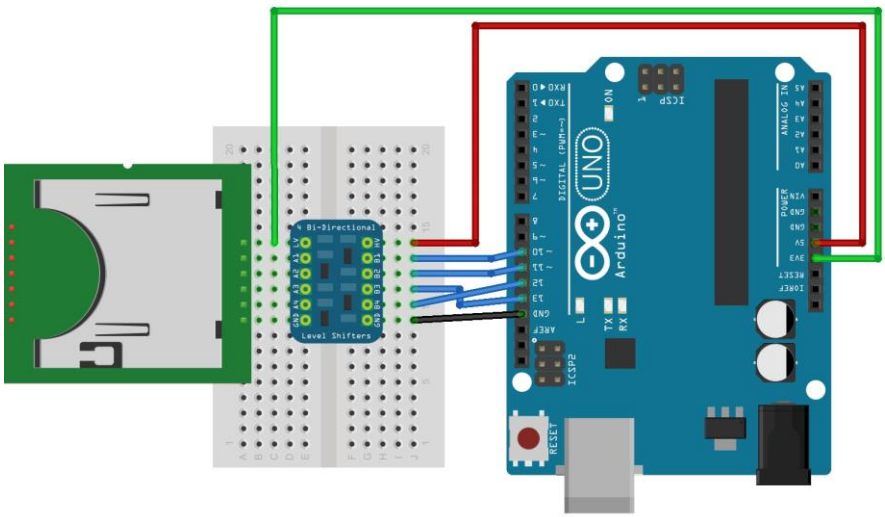


Figure 48: Wiring diagram of the SD card logger with the Arduino.

7.1.4.2. Software for data logging with SD cards

In this section the main instructions needed for data logging to an SD card which works with SPI communication will be explained. Most of these instructions can be seen in the Arduino SDlogging program example.

7.1.4.2.1. Libraries

- **SPI.h:** This library includes helpful instructions for SPI communication in general.
- **SD.h:** This library has all the instructions needed to communicate with an SD card device.

7.1.4.2.2.Global variables

- **Int *chipselect*:** An int variable named *chipselect* is created and assigned to 10. This variable will later be used to define which port of the Arduino is the one that is in charge to select which SPI device is communicating to (this is, the SS port).
- **File *logFile*:** A variable of type file is created, which will be used in order to create the files inside the SD card.

7.1.4.2.3. Setup

In the Setup, the serial communication is started, the SD card initialized and the file where the data is going to be saved created.

- **Serial.begin(9600):** This command initializes the serial port for terminal-communication with the PC, useful for debugging.
- **pinMode(*chipselect*, OUTPUT):** this command will set the *chipselect* port as an output in order to send instructions to the SPI devices connected about which one is it talking to. In this case, as there is only one SPI device this feature will not be used, but it is important in case of holding more than one SPI device.
- **SD.begin(*chipselect*):**With this instruction the SD card is initialized. This function returns a Boolean, true if the SD card could be initialized and false if it did not, so it is usually set as an "if" condition in order to send a failure message through the terminal port in the case of not being able to initialize it.

If the program is wanted to create a new file each time the program executes instead of always writing over the same file, the following process can be used:

- A **char** type variable named *filename* is created.
- With the **strcpy(*filename,string*)** a *string* is assigned to this variable. This string will be the root of the file name. As an example, *string* could be *"log00.txt"*.

- With a ‘for’ loop, the two zeros of the filename will be sequentially changed by adding 1 unit. Inside this loop there is an “if” structure with the instruction **!SD.exists(*filename*)** as condition. If there already is a file saved in the SD card with the name proposed, the “for” loop will continue until a non-existent file name is found.
- Finally, the file is created and opened for writing with the ***logfile*=SD.open(*filename*,FILE_WRITE)** instruction and can be referenced directly with the *logfile* variable.

7.1.4.2.4. Loop

In the loop, data can be written in the file by using the ***logfile*.print()** instruction. Also, the instruction ***logfile.*flush()** is used. This instruction waits for the transmission of outgoing serial data to complete and is used in order to prevent data loss.

7.1.5. MATERIAL

The Arduino used for this module was not the Arduino UNO but the Arduino Nano. The reason behind this decision is that this version of the Arduino is much smaller and thus lighter, making it more suitable for an application where weight and size are critical; and its pins can be soldered, which diminishes the probability of accidental disconnection of lines due to movement of the structure. It also runs on the same microprocessor so all the programs are compatible, as well as the pin’s configuration. The Nano used was not an original Arduino, but a clone developed by Funduino, which was lent by the Neslab team, but can be bought for around 9€, depending of the dealer.

The sensors used were the two 100KΩ thermistors and the Melexis 90614-ACF IR sensor. One of the thermistors would be placed inside the module in order to track the inside temperature of the module, one would be outside the module in order to track the atmospheric temperature and the Melexis IR sensor would be outside the module too but pointing 45º down with respect to the XY plane using the body-axes system. This last sensor was supposed to point directly vertical up to the sky in order to track the temperature of the sky above, but if that was done the balloon would be inside its field of view and would interfere with the readings. For this reason it was given an angle with enough margins so that if the module shacked there was no chance that the balloon would end up inside the field of view of the sensor.

The SD card data logger was the LC Studio SD card data logger, obtained for 1,40€. The Level shifter used, as mentioned before, is the BSS138 4-channel I2C-safe Bi-directional Logic level converter.

The overall cost of the whole module was of 36,54€. The price breakdown can be found at Table 8.

Item	Unitary Cost	Number of units	Total cost
Melexis 90614-ACF IR sensor	17,45€	1	17,45€
100KΩ thermistors	1,20€	2	2,40€
Studio SD card data logger	1,40€	1	1,40€
Adafruit BSS138 4-channel Logic level converter	3,30€	1	3,30€
Funduino Nano	8,99€	1	8,99€
Other (wires, drilled plate, resistances, etc)	-	-	3€
Total			36,54€

Table 8: Cost breakdown of the weather helium balloon module.

7.1.6. WIRING

The wiring of the sensors would be a superposition of the previously presented wirings at section 6.2. SENSORS for the thermistors and the Melexis IR sensor. The SD data logger would be wired as explained in section 7.1.4.1. Wiring for data logging with SD cards. The wiring of all the system can be seen in Figure 49.

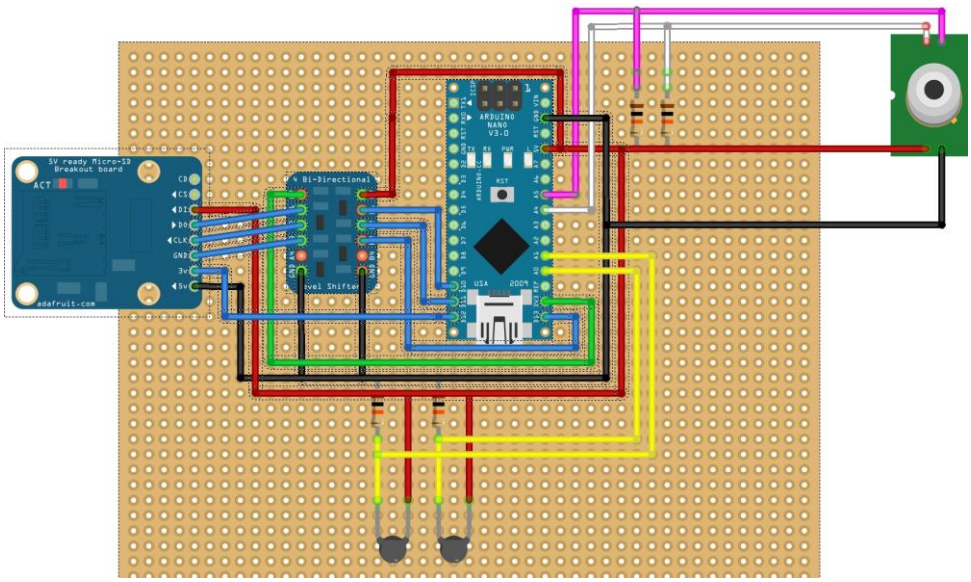


Figure 49: Wiring diagram of the weather balloon module.

7.1.7. SOFTWARE

For this application only the Arduino code will be explained, as the post-process code was as simple as importing the data to matLab and plotting it. However, the matLab code can be found at the annex section 1.3.2. MATLAB CODE (POST-PROCESS).

The code for Arduino is a combination of the codes for the thermistors and the Melexis IR sensor which are explained at section 6.3. SOFTWARE FOR INTERFACING THE ARDUINO AND THE SENSORS, adding the code necessary for data logging into an SD card explained previously at section 7.1.4.2. Software for data logging with SD cards. The full code can be found at the annex section 1.3.1. ARDUINO CODE.

The program works as follows:

- The libraries for communication with the Melexis sensor and the SD logger are included and the global variables bond to the thermistors and the log file defined.
- In the setup, the Melexis sensor and the SD card are initialized and a file created into the latter with a unique file name.
- In the loop, the sensors data is read by the Arduino using the ***mlx*.readObjectTemp()** and the **analogRead(*thermistor*)** instructions. Then, it is saved to the SD card by using the ***logfile*.print()** instruction. A **flush()** is added in order to ensure that the data have been saved and a **delay(1000)** so that there is a spacing of 1 second between every measurement.

7.1.7. TEST IN A REAL ENVIRONMENT

The module was installed in the Neslab launch on the 22nd of December of 2014. The helium balloon carried two payloads, one containing different modules developed by the Neslab team and the one developed in this project and the other developed by old veteran team members from Neslab.

7.1.7.1. Launch site and climatology

The module was launched from Bell-lloc d'Urgell, Catalonia, Spain at 8:43am (367m height from sea level) and recovered the same day at 11:37am near Coromines, Catalonia, Spain (see Figure 50). It reached 29,42Km altitude.

The climatology for that day on morning was good but with a thick fog at an approximate height of 1000m (Figure 53), due to an inverted gradient of the atmospheric temperature. This fact will be visible in the data collected, as it will be seen later. The climate predictions for that day permitted to estimate the flight path and the landing site, which only differed from the actual landing site by 7km.

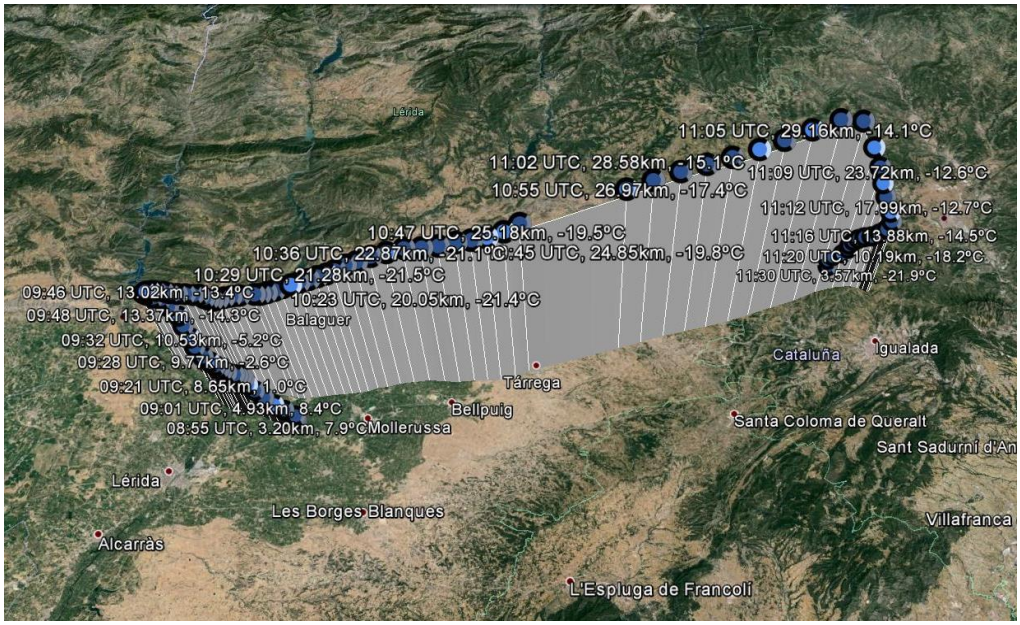


Figure 50: Image of the 22nd of December of 2014 Neslab launch flight path.

Figure 51: Photograph obtained from an on-board camera of the Neslab balloon, when it was about to enter the fog. The payload that is showed is the one developed by the old veteran team members from Neslab.

7.1.7.2. Results

Due to the fact that the altitude data had a higher monitoring time, at the end of the experiment there were more temperature data points than altitude's, so altitude points were created between the actual altitude data points assuming a linear increase in order to match the number of temperature data points.

The data obtained for the three temperature sensors against altitude for the ascending phase can be seen in Figure 52.

. In the same graph the ISA (international standard atmosphere) temperature has been plotted for comparison with the measured atmospheric temperature. The data from the thermistors and the IR Melexis sensor has been calibrated. The descent data has been discarded as it does not add useful information for the analysis. However, all the data of the experiment can be found at the annex section 3.1. HELIUM WEATHER BALLOON MODULE.

The data obtained by the thermistor placed outside in order to track the atmosphere temperature is shown in dark blue. The temperature at ground level (367m from sea level) was about 1ºC. The temperature then increased up to 17,82ºC at 1651 meters height. Afterwards it would decrease until hitting -44,43ºC at 18,2km, where it would start heating until the end of the climbing at 29,42km, where it reached -15,38ºC.

The Melexis IR sensor's data (green) is constantly diminishing but at different rates depending of the altitude. Its measurements start at -27,23ºC and diminish slowly until -33,09ºC are hit at 2621m height (2254m relative to the ground level). After this the measurement variation increases, reaching -74,33ºC at 8,03km height. From this point up, the temperature slowly starts to tend to a temperature asymptote, being the closer value to the limit temperature the last data point taken at 29,42km, -119,50ºC. It is worth noting that in this last stage there is increasing measurement noise as the height increases, although it looks like it stabilizes at 17km where the noise is about ±5,5ºC with some high temperature peaks.

The data obtained by the thermistor placed inside the module is shown in red. From launch to 6,32km height the temperature slightly rises from 17,02ºC to 19,36ºC. From that point until 10,52km the temperature remains fairly stable and then starts to drop until 6,20ºC are reached at 29,42Km.

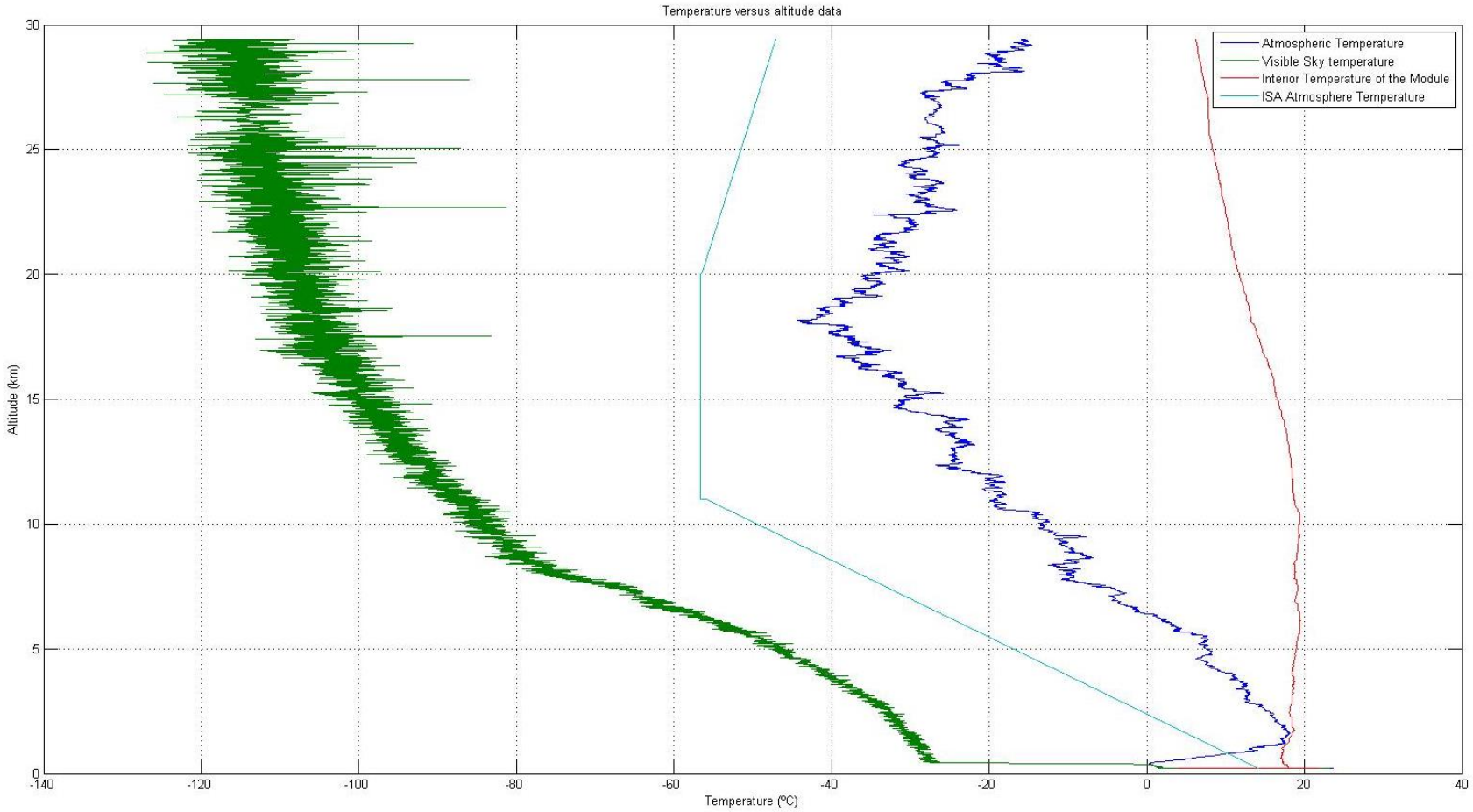


Figure 52: Sensors data and ISA temperature profile versus altitude.

7.1.7.3. Analysis of the results

As it was explained in section 7.1.7.1. Launch site and climatology that day there was an inverse thermal gradient. This fact is reflected in the initial increase of temperature of the outside thermistor. Afterwards the temperature decreases as expected. It can be seen that the thermal gradient is not as shallow as the one for the troposphere proposed by the ISA temperature model, but they are fairly even. However, when the end of the troposphere is reached, the temperature keeps decreasing instead of remaining constant as the ISA temperature model proposes. At 18,2km the temperature starts rising again due to the fact that the layer with maximum ozone concentration is reached. This behavior is modeled by the ISA at 20km and it can be observed that the gradients are fairly even. The higher temperatures of the actual data versus the model data may be due to the low atmospheric pressure of that day.

The Melexis IR sensor measurements firstly decrease at a slow rate. This is due to the fact that there was thick fog so the IR sensor measurement is actually the fog's temperature. As it enters the fog and the sensor has less thickness of it above, the rate at which the temperature decreases increases. When it leaves the clouds, as height is gained, less atmosphere and thus humidity interferes with the IR measurements, slowly reducing the gradient of the measured temperature, tending to stabilize. Although not enough height was gained in order to completely stabilize the measurement, its tendency to a certain value is visible.

Finally, the increase of noise at 8km can be produced for several reasons:

- At 8km height the temperature measured reaches -70ºC, which is, according to the manufacturer, the minimum temperature it can measure. For this minimum temperature, the error of the sensor is ±4ºC [25], which is approximately the noise it is getting.
- Due to the fact that the payload was attached to a single rope, it could spin. This could lead to moments when the IR sensor was pointing towards the sun and others were it was pointing away from it. This fact would increase the noise and could be the explanation to the hot peaks that can be seen especially over 17Km.

The interior thermistor shows a fairly constant value. The warming system as well as the insulation does a great job. However, an interior-temperature control could be designed in order to minimize the electrical consumption, as the temperature does not need to be held so high and it even increases for the first section of the flight. On the other hand, at the end of it, temperature decreases significantly (although not being critical). A simple temperature control could manage the temperature more efficiently, keeping the electronics at its optimal temperature of operation reducing electrical consumption.

7.2. TERRAIN TEMPERATURE MAPPING

The second application developed was a device which could carry out a thermal terrain mapping, designed in order to be able to be installed in most standard professional civil drones. This device is intended to offer a much cheaper alternative to the conventional terrain mapping by using low cost temperature sensors instead of thermal cameras, which have a relatively high cost (see section 5.2.2.3. Other Portable Devices), and mounting them on drones rather than on helicopters, reducing the operation cost.

Figure 53: Photograph of a standard professional civil drone with a camera as payload.

Thermal terrain mapping is used for several purposes, such as crop irrigation control for plantations, vegetation monitoring, fire monitoring (both subterranean and surface fires), high voltage lines monitoring, thermal efficiency of buildings, search of lost people in inhospitable regions, among others.

In order to carry out a temperature mapping of an area, the device relies on the Melexis 90614-ACF IR sensor (previously studied in section 6.4.2. ASSAY OF THE INFRARED SENSOR) to measure the surface temperature of the terrain while the device is mounted on a flying drone. Unlike thermal cameras, which do an instantaneous thermal photograph of an area, the Melexis IR sensor takes punctual measures of the surface's temperature inside its field of view which makes it unable to do a thermal terrain mapping by itself. In order to work this problem around, every 0,2 seconds the device registers a temperature measurement and the position where it was taken obtained from a GPS installed in the device. By flying the drone all around the terrain that is wanted to be analyzed, a full thermal terrain mapping can be obtained. All the data taken is saved into an SD card using an SD logging device, installed in the device.

In order to carry out these operations, the device had the following requirements:

- Capacity to determine its position using a GPS.
- SD card data logging capabilities.
- Power autonomy.

7.2.1. MATERIAL

For this application, the material that was used to form the module will be presented first for convenience.

In order to cover the GPS position and the SD card data logging requirements, the Adafruit Ultimate GPS data logging shield was obtained [33]. This Arduino shield has installed a MTK3339 GPS (a -165dBm, 66 channels, 10Hz update GPS chipset), as well as an SD card logger which comes with an included level shifter. It also has some proto-shield space, which makes possible to solder circuits on the shield itself.

The GPS has an antenna included, so no external antenna is needed. However, it has a connector in case one is wanted to be used if the module is intended to be placed in a confined space with no clear view of the sky. It was considered that for this application an external antenna was not necessary.

The SD card logger works as any other SPI-communication based SD card logger would, and the internal connections of the shield are equivalent to the ones presented before in section 7.1.4.1. Wiring for data logging with SD cards.

The IR sensor used, as mentioned before, is the Melexis 90614-ACF IR sensor. In order to measure the terrain's temperature this sensor needs to be put upside down. This carries along a difficulty, as it cannot be mounted on the proto-shield space because the GPS needs to be pointing upwards. This made necessary to make an additional small platform to hold it in place.

This module was firstly planned for an Arduino UNO. However, even though all the software and wiring would be correct, the program would not work as expected. Finally, after corroborating that the code was actually correct, it was determined that the cause of the problem was a memory issue. As explained in section 6.1.1.1. ATmega328 Microcontroller Specifications and Memory, the Stack used memory grows from top to bottom and the Heap from the Static Data upwards. If both two memories grow to much they finally collide and one overwrites the other, causing program malfunction. As a first solution, the code was optimized and reduced, but the problem was not solved. In order to be able to further reduce the memory used by the program, it should have been rewritten without using libraries and using low-level instructions. However, due to time restrictions, this was discarded as an option.

Instead, an Arduino MEGA was obtained and the code was adapted for this platform. The Arduino MEGA runs on a different microcontroller of the same family (which make programs designed for the Arduino UNO highly compatible with the MEGA), the ATmega1280. It is designed as a platform which is able to support a higher amount of inputs and outputs, as well as more serial communications. However, the reason why it was picked (as this module is actually working with a small amount of external devices) is because it has more memory available (see Table 6). With this increase in memory, the program was able to run perfectly.

The overall cost of the whole module was of 36,89€. The price breakdown can be found at Table 8.

Item	Cost
Melexis 90614-ACF IR sensor	17,45€
Adafruit Ultimate GPS data logging shield	40,87€
Arduino MEGA	35€
Other (wires, drilled plate, resistances, etc)	3€
Total	96,32€

Table 9: Cost breakdown of the termal terrain mapping module for drones.

7.2.2. GPS POSITIONING

In this section, the operation with the MTK3339 GPS through the Adafruit Ultimate GPS data logging shield will be explained. The SD data logging will not be explained as it is completely equivalent to the one explained in section 7.1.4. DATA LOGGING.

7.2.2.1. Wiring of the shield with the Arduino MEGA

Due to the fact that the shield is designed for Arduino UNO and an Arduino Mega was used, the shield had to be adapted for its operation with the MEGA. The main problem is that serial software is used on pins 7 and 8 for communication with the GPS when using an UNO, but software serial is not supported by the Arduino MEGA. Along with some code, in order to work around this problem, the Tx and Rx (transmit and receive) pins of the GPS needs to be bridged and connected to the Rx and Tx of a Serial communication of the Arduino MEGA (Arduino MEGA has 3 independent serial communication channels).

7.2.2.2. Software to interface the shield with the Arduino MEGA

In this section the code needed in order to make the GPS work is explained. This lines of code are the used in this module's program.

7.2.2.2.1. Libraries

- **Adafruit_GPS.h:** This library developed by Adafruit gives the user a straighter way of asking the GPS for information.
- **SoftwareSerial.h:** This library is needed for the serial communication between the Arduino and the GPS.

7.2.2.2.2.Global variables

- **Hardware Serial:** a hardware serial is started. It is defined through which serial ports it will be held. In the case of using an UNO, this would be a Software serial through pins 7 and 8.
- **Adafruit_GPS *GPS*(&*mySerial*):** The GPS object is created. In this line it is also related to the hardware serial defined before (*mySerial*), which will be the channel used to communicate.
- **GPSECHO:** this Boolean can be activated or deactivated in case that an echo of the data received by the GPS is wanted to be displayed to the user, for debugging purposes.
- **Boolean usingInterrupt:** this Boolean activates or deactivates the use of an interrupt which will e explained later.

7.2.2.2.3. Setup

In the Setup, the serial communication is started, the SD card initialized and the file where the data is going to be saved created.

- **Serial.begin(115200):** This command initializes the serial port for terminal-communication with the PC, useful for debugging. It is started at the 115200 baund because it permits a higher frequency of measurements and because the 9600 baund will be occupied by the Arduino-GPS communication.
  - **GPS.begin(9600):** Arduino-GPS communication is started through the 9600 baund serial communication.
  - **GPS.sendCommand(*option*):** This instruction sends set up instructions to the GPS. If *option* is PMTK_SET_NMEA_OUTPUT_*MODE*, it is being communicated to the GPS which kind of data is wanted by changing *MODE* to different options; with PMTK_SET_NMEA_UPDATE_*FREQUENCY* the rate at which the GPS data is updated is set to the frequency defined by *FREQUENCY*; finally if PGCMD_*ANTENNASTATUS* is sent, it is being said to the GPS if an external antenna is being used or not through the *ANTENNASTATUS* line.

7.2.2.2.3. Functions and interrupts

In order to use the GPS it is recommended the use of an interrupt which is called once a millisecond, looks for any new GPS data and stores it.

7.2.2.2.4. Loop

In the loop, first it is checked if new information is received through the **GPS.newNMEArecieved()** instruction. The GPS data is transmitted to the Arduino

through the **GPS.parse(GPS.lastNMEA())** instruction. After parsing it, the information can be accessed through instructions which come from the Adafruit library, such as GPS.hour, GPS.minute and GPS.seconds, which are time information; GPS.fix and GPS.fixquality which are satellite-connexion information; GPS.latitude and GPS.longitude, which gives the location; among others.

7.2.3. WIRING

In this section the wiring for the module will be explained. As a shield is being used it is a really simple operation:

- The shield is plugged onto the Arduino MEGA. This sets up all the connections with the SD card and the GPS.
- The serial bridge explained at section 7.2.2.1. Wiring of the shield with the Arduino MEGA is done (in the module it was connected to the Arduino MEGA Serial1).
- Due to the fact that Arduino MEGA has its own SPI dedicated pins, the Melexis sensor does not need to be plugged to the analog pins, but it is plugged to the SDA and SCL pins (pin 20 and 21). Power and ground are also connected.

7.2.4. SOFTWARE

For this application, the device's Arduino code as well as the post-process code will be explained.

7.2.4.1. Arduino Code

The code for Arduino is a combination of the codes for the Melexis IR sensor which are explained at section 6.3. SOFTWARE FOR INTERFACING THE ARDUINO AND THE SENSORS, adding the code necessary for data logging into an SD card explained previously at section 7.1.4.2. Software for data logging with SD cards and the one necessary for communication with the GPS previously seen at section 7.2.2.2. Software to interface the shield with the Arduino MEGA. The full code can be found at the annex section 1.4.1. ARDUINO CODE.

The code works as follows:

- The libraries and global variables are initialized.
- In the setup, the sensor, the SD card and the GPS are initialized. The SD logging file is created. The type of information that is wanted is sent to the GPS, as well as its frequency (5Hz for the application).

- Once every millisecond, an interrupt will force the actualization of the GPS's data.
- In the loop, if new position data is received from the GPS, it is parsed. If there is a fix (the GPS received Satellite data), the GPS position information and the temperature registered by the Melexis are saved into the log file in the SD card.

7.2.4.2. MatLab Code

The MatLab code can be found at the annex section 1.4.2. MATLAB CODE (POST-PROCESS). The code imports the data from the log txt file and assigns the longitude, latitude and the temperature to different vectors. Then, a 100x200 mesh is created using the **meshgrid** instruction. The mesh divisions should be modified as function of the area scanned. With the MatLab's griddata function, the experimental data is plotted on the grid and from this data, the grid's node data are calculated. With this system, instead of having punctual values of temperature, the value of temperature at the nodes of the grid is approximated and then a linear approach between nodes is carried away, giving a continuous temperature measurement. Finally the results are plotted. This plot can be exported to jpg and then be added as a layer on Google earth, giving a temperature mapping over the satellite photograph of the terrain.

7.2.5. TEST IN A REAL ENVIRONMENT

Due to lack of time, the module could not be tested in its planned real environment (mounted on a drone). However, before-flight tests were carried away, which had similar conditions. This tests were carried away in order to debug the code and look for errors before mounting the device on a drone, as when mounted debugging becomes much slower.

In order to recreate a real environment, the device was attached to the end of a telescopic stick. This would recreate the fact of flying, enabling to see the measurements acquired by the Melexis IR sensor from an approximately 3 meters height from the ground. The device was not fully autonomous; as this activity was carried away for debugging thus the Arduino was connected to a PC through a long USB cable.

7.2.5.1. Test site

The test was carried away at 1:15pm on 23rd December 2014 at Esplugues' Pompeu Fabra Park (C/Professor Barraquer). This site was decided for its vicinity and variable terrain materials (sand, concrete, plants, …). As the sun was low due to winter time, the shadows from the buildings south to the park generated two temperature areas in the park too, due to the fact that the day before had rained and the side which was under

the shadow did not dry. In order to see if the device was capable to “see” relatively small hot spots, three people sat on the floor at different positions in order to check if the IR sensor measurement changed.

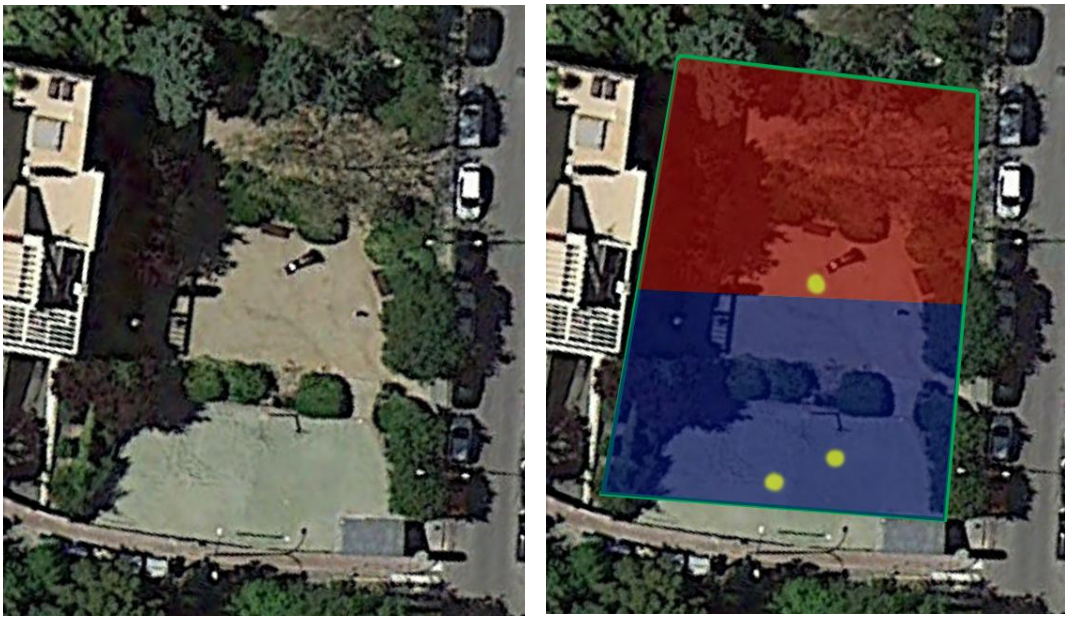


Figure 54: Photograph of the test site (left) from Google Earth and diagram (right) showing the shadowed area (blue), the sunlit area (red) and the hot spots (yellow).

7.2.5.2. Results and analysis

The data obtained in the test can be found at the annex section 3.2. TERRAIN TEMPERATURE MAPPING. The data was post processed with the MatLab code explained at section 7.2.4.2. MatLab Code. The plotted results can be seen at Figure 55.

It can be observed that the upper half of the graph is overall hotter than the lower half. This is due to the fact that, as explained in section 7.2.5.1. Test site the lower part of the terrain was shadowed while the upper one was sunlit.

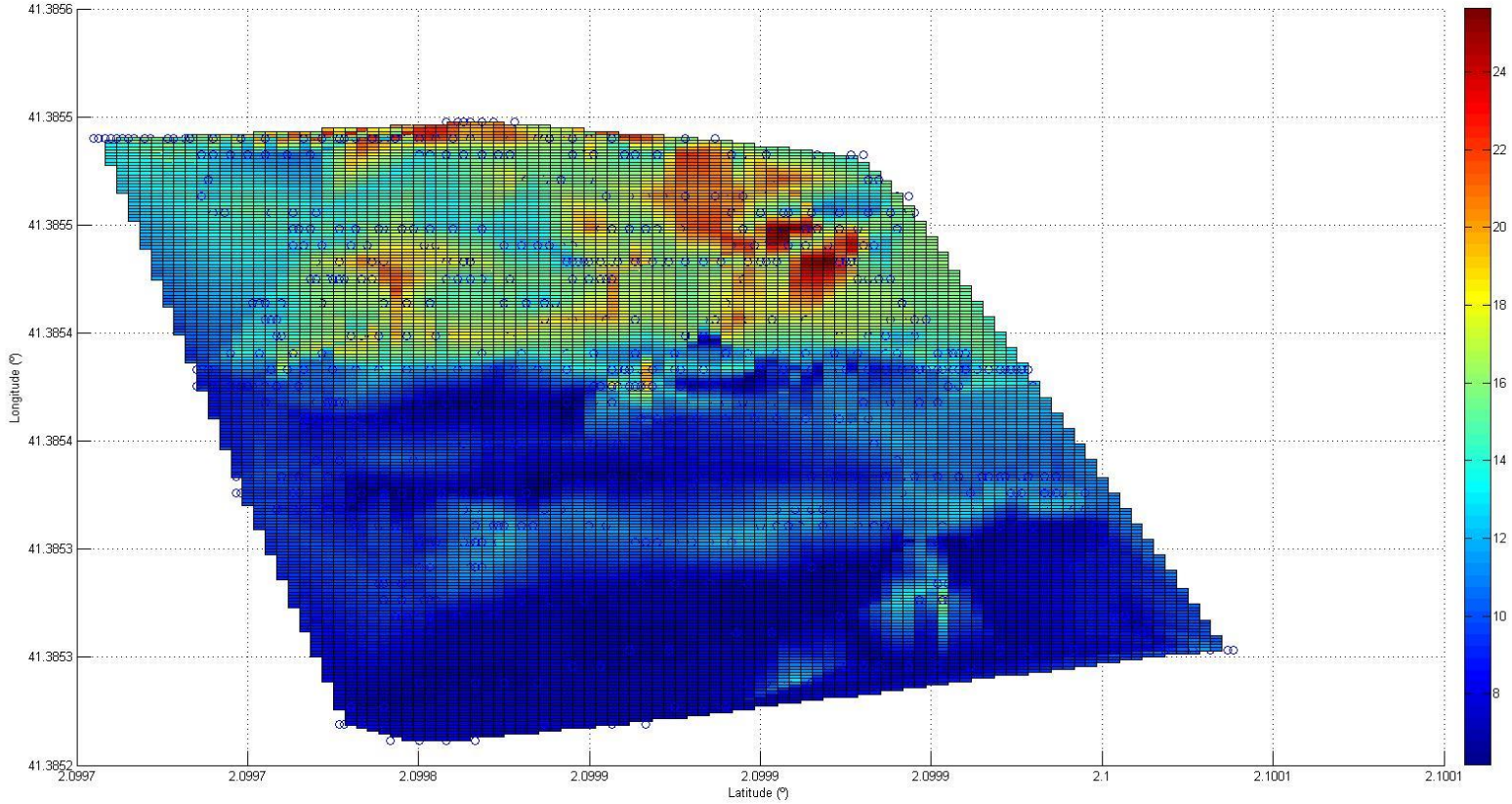


Figure 55: **Graph displaying the plotted results for the Thermal terrain mapping for drones test. Color scaling represents temperature readings. In dark blue both the grid and the data points can be seen.**

In Figure 56 the resulting plot is superimposed to the Google Earth's satellite image of the terrain scanned. With this image details of the results will be further analyzed. It is worth noting that the MatLab's plot does not have a good scaling between the longitude and the latitude, which forces to adjust the image so that it fits well with the actual area scanned.

On the lower part of the cold area two hot spots are visible where two people were sitting. The spot's temperature is about 8ºC higher (around 16ºC). This is not a human-body temperature but this is due to the fact that the temperature measured is not the person's but the clothes' he was wearing. It can be seen that the upper right spot is bigger. This is because there was more density of measurements in that region than in the lower left spot. The third hot spot can be seen in the limit between the hot region and the cold region, in the middle of the map, as a 20 to 22ºC spot.

It also can be observed that there is an horizontal line of hotter measurements across the cold area. These hotter readings are caused by the presence of vegetation, which retains the temperature better than the ground. This fact can be also observed in the hotter region, where those plants could reach up to 25ºC.

It must be mentioned that as it can be seen in the image, some vegetated areas are not hotter. This is caused by the fact that the maximum altitude obtainable by the telescopic stick was about 3m height and that vegetated areas are trees. On this areas the sensor scanned the area under the tree's crown as it was impossible to do it otherwise, so it is actually the ground temperature, not the trees'.

Although not being visible neither in the graph nor on the image, there is a cold spot in the middle of the terrain near the hot spot forced by the presence of the person. This cold spot is the measurement of temperature of a toboggan slide, which was made of polished metal. The temperature measured in this zone was even under zero at some points. Rather than being the actual temperature of the slide, this measurement is caused because of the high emissivity of polished metals, which along with the set up emissivity of the Melexis IR sensor causes these low measurements.

Overall the results are satisfactory and could be useful for different applications, although they are highly dependent of data concentration and, as seen by the toboggan slide, of emissivity. Even if the height at which it was tested was not as high as if mounted on a drone, it is believed that such good results will repeat.

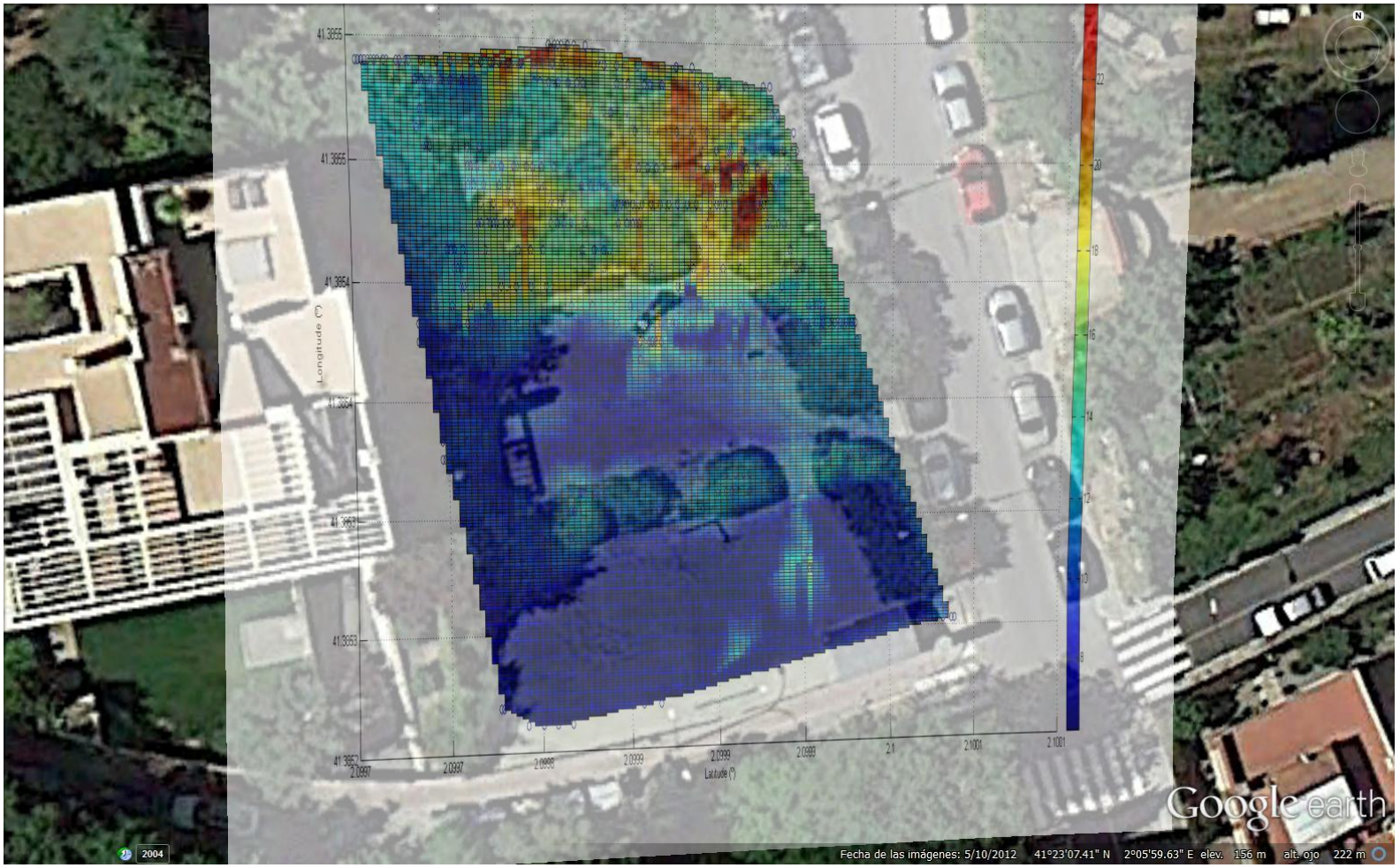


Figure 56: Image of the terrain termal mapping test superimposed in Google Earth over a satellite image of the terrain.

7.3. ECONOMICAL AND ENVIRONMENTAL ASPECTS

The weather balloon model presented reliable and good measurements at highly affordable costs. Even if the structure and the balloon itself, which were not covered in this work, the cost of electronics were cut to its bare minimum obtaining good and useful information at an excellent cost. Also, the low weight of the electronics used contributed to be able to obtain more height and include more experiments to the payload.

The thermal terrain mapping module presents a great alternative to the current-state aerial thermal imaging. At the cost of a higher time of operation in order to obtain results, this module not only relies on a much cheaper system in order to get the thermal data but is also intended to be used on a platform with far lower operational costs than the helicopters used nowadays. Also, this has a great impact to lowering the environmental impact, as drone energy consumption is way lower than a helicopter's. It must be added that the utilization of unmanned vehicles makes it also more suitable for high risk operations, such as fire monitoring.

8. FUTURE DEVELOPMENT

8.1. ASSAY AND CALIBRATION OF SENSORS

In order to have a more reliable analysis of the sensors, some improvements could be developed for the assays.

In the case of the contact sensors, the use of a functioning heater with controlled temperature would make possible the initial planned experiment, which would mean a better controlled environment with more reliable data. Also, obtaining a reference thermometer with higher temperature range would mean that a higher range of temperatures could be analyzed, which would provide information of the sensor's response at lower and higher temperatures than studied. This would also mean a better calibration. In the pursue of analyzing more range, the experiment could be carried away also in a cooling chamber.

For the Melexis IR sensor, the experiment platform developed should be further developed in order to be able to obtain a fully isothermal surface. Also, the position where the reference sensors should be laid down should be studied in order to be able to measure exactly the surface temperature, even if the heater is on. In addition to this, the Melexis IR sensor could also be heated and cooled in order to make the sensor operate at different temperatures, as the error it presents is also function of its own temperature. This would lead to a full analysis of the senor over a wide temperature range.

8.2. HELIUM WEATHER BALLOON TEMPERATURE MODULE

The main part that could be improved of this application is the IR sensing. Due to the fact that it is not known where the sensor is pointing, it is not possible to monitor the effect of the sun in the data taken. In order to overcome this problem, multiple IR sensors could be placed pointing in the same angle but installed in different faces of the payload, ensuring that there would always be one pointing to the sun and one away. In order to know afterwards when one sensor was receiving direct light from the sun, photo sensors could be installed near each sensor.

In addition to this, the IR sensors could be installed in sticks coming out from the payload rather than on the sides of it, making unnecessary to incline them and obtaining a better direct view of the sky.

Also, redundant contact thermometers could be installed. It would also be beneficial to receive real-time data from the module, rather than saving it on an SD card and having to recover it. The real time data could be used in order to make atmospheric analysis in real time and the loss of the payload would not lead to loosing the data.

8.3. DRONE THERMAL TERRAIN MAPPING MODULE

The main immediate future development of this module is to make the platform to install it on a drone. The attachments need to be designed and optimally it should be installed on a gimbal in order to ensure that the sensor is always facing downwards.

After being able to install it on a drone, the experiment could be further assayed in a real environment. Not only the real-environment behavior could be analyzed, but the data obtained would be of more quality due to the fact that it would be easier to obtain a homogeneously spaced data with good density, as the drone's position and velocity is easier to control. This would also make possible to improve the post process program.

Finally, real-time analysis could be implemented. By using remote communication, the data could be sent to the computer and obtain a thermal terrain mapping in real-time, which would make the device more useful overall but specially in fire control and search of lost people.

After all the application's characteristics were fully developed, the Arduino code could be rewritten at a low-level in order to make possible to run it in an Arduino UNO, or better, an Arduino Nano. This would reduce the cost of the device as well as its weight. Alternatively, a microcontroller could be built with the same elements and connections used in the application as the Arduino MEGA, which would have an overall lower cost, weight and size.

9. CONCLUSIONS

Temperature is one of the most commonly monitored physical properties in industry. The systems used for this purpose in order to have good and reliable measurements are expensive, and usually they are not suitable for field work due to its bulkiness. Transportable or handheld devices exist, but they are usually restricted in capacities or have a poor flexibility.

Low-cost temperature sensors interfaced with Arduino offer a good alternative to these systems. Arduino is a great platform in order to be used as a data acquisition system for sensors for a number of reasons:

- It is low-cost.
- It is open source.
- It has available a number of different communication protocols which make multiple sensors reading possible.
- It can be easily expanded.
- It is highly flexible to the user's needs.
- Data logging or data transmitting is possible including the correct device.
- It can be used as an interface between the sensors and the computer to obtain and process real-time data.

The sensors which can be used with Arduino are vast. The sensors tested in this work were thermocouples type k interfaced with the MAX31850K amplifier, 100kΩ thermistors, the Dallas DS18B20 IC sensor and the Melexis 90614-ACF IR sensor.

All the sensors demonstrated that great precision can be obtained with them, with the exception of the thermocouples interfaced with the MAX31850K which did not have such a good response, although they would be suitable for a lot of applications never the less (especially those were high temperatures are expected). Overall they had a great lineal response and low error out of the box and after being calibration their results are comparable to those of a commercial temperature sensing system. Their cost was between 1,20€ and 17,44€.

With the applications developed it has been demonstrated that these sensors used with Arduino are useful for real applications. The sensors offer good and reliable results and Arduino has the flexibility to adapt to the application's needs. The systems developed offer an alternative to systems which have a higher cost of acquisition as well as operation.

10. BIBLIOGRAPHY

[1] T.P.Wang, "Thermocouple Materials," *ASM - The Materials Information Society,* pp. k88 -k107, 1990.

[2] Maxim Integrated, "Implementing Cold-Junction Compensation in Thermocouple Applications," 26 April 2007. [Online]. Available: http://www.maximintegrated.com/en/app-notes/index.mvp/id/4026. [Accessed 03 November 2014].

[3] National Instruments, "Cómo Realizar una Medición con Termopares," 19 August 2013. [Online]. Available: http://www.ni.com/white-paper/7108/es/. [Accessed 02 November 2014].

[4] A. Tong, "Improving The Accuarcy of Temperature Measurements," 2001. [Online]. Available: http://www.picotech.com/applications/temperature.html. [Accessed 2014 November 03].

[5] National Instruments, "Guía para Realizar Medidas de Temperatura con RTDs," 19 August 2013. [Online]. Available: http://www.ni.com/white-paper/7115/es/. [Accessed 03 Noveber 2014].

[6] Agilent Technologies, "Practical Temperature Measurements," 26 January 2012. [Online]. Available: http://cp.literature.agilent.com/litweb/pdf/5965-7822E.pdf. [Accessed 03 November 2014].

[7] M. Massoud, Engineering Thermofluids: Thermodynamics, Fluid Mechanics, and Heat Transfer, Berlin, Germany: Springer, 2005.

[8] Scigiene Corporation, "Infrared Thermometers," [Online]. Available: http://www.scigiene.com/pdfs/Infrared%20Thermometers.pdf. [Accessed 03 November 2014].

[9] OMEGA, "Introduction to Infrared Thermometer," [Online]. Available: http://www.omega.com/prodinfo/infraredthermometer.html. [Accessed 03 November 2014].

[10] OMEGA, "Digital Thermometer HH11B," [Online]. Available: http://www.omega.com/pptst/HH11B.html. [Accessed 02 December 2014].

[11] OMEGA, "RDXL-SD Series Portable Thermometer/Data Loggers with SD Card and Thermocouple Input," [Online]. Available: http://www.omega.com/pptst/RDXL-SD_SERIES.html. [Accessed 02 December 2014].

[12] OMEGA, "Compact Portale Data Logger," [Online]. Available: http://www.omega.com/pptst/RDXL120.html. [Accessed 02 December 2014].

[13] AllQA, "AllQA Infrared Thermometers," [Online]. Available: http://www.allqa.com/infrared.html. [Accessed 06 December 2014].

[14] Fluke, "Fluke VT04 Visual IR Thermometer," [Online]. Available: http://www.fluke.com/fluke/m3en/thermometers/infrared-thermometers/fluke-vt04.htm?PID=77085. [Accessed 06 December 2014].

[15] National Instruments, "Dispositivo de Medidas de termopares NI USB-TC01," [Online]. Available: http://sine.ni.com/nips/cds/view/p/lang/es/nid/208177. [Accessed 07 December 2014].

[16] National Instruments, "Sistema de Medidas NI 9215," [Online]. Available: http://sine.ni.com/nips/cds/view/p/lang/es/nid/209871. [Accessed 07 December 2014].

[17] Arduino, "Arduino UNO," [Online]. Available: http://arduino.cc/en/Main/ArduinoBoardUno. [Accessed 7 December 2014].

[18] B. Earl, "Learn Adafruit: Arduino Memories," Adafruit, [Online]. Available: https://learn.adafruit.com/memories-of-an-arduino/arduino-memories. [Accessed 07 December 2014].

[19] Arduino, "Arduino Software," [Online]. Available: http://arduino.cc/en/Main/Software. [Accessed 07 December 2014].

[20] Adafruit, "Thermocouple Type-K Glass Braid Insulated," Adafruit, [Online]. Available: http://www.adafruit.com/products/270. [Accessed 08 December 2014].

[21] Adafruit, "Adafruit 1-Wire Thermocouple Amplifier - MAX31850K," Adafruit, [Online]. Available: https://learn.adafruit.com/adafruit-1-wire-thermocouple-amplifier-max31850k/overview. [Accessed 08 December 2014].

[22] Adafruit, "4-channel I2C-safe Bi-directional Logic Level Converter - BSS138," Adafruit, [Online]. Available: http://www.adafruit.com/products/757. [Accessed 08 December 2014].

[23] Maxim Integrated, "DS18B20 Programmable Resolution 1-Wire Digital thermometer Datasheet," Maxim Integrated, [Online]. Available: http://datasheets.maximintegrated.com/en/ds/DS18S20.pdf. [Accessed 12 December 2014].

[24] Melexis, "MLX90614 family Single and Dual Zone Infra Red thermometer in TO-39 datasheet," Melexis, [Online]. Available: http://www.adafruit.com/datasheets/MLX90614.pdf. [Accessed 08 december 2014].

[25] Arduino, "Arduino playground: Reading a thermistor," Arduino, [Online]. Available: http://playground.arduino.cc/ComponentLib/Thermistor2. [Accessed 11 December 2014].

[26] Adafruit, "Using Melexis MLX90614 Non-Contact Sensors," Adafruit, [Online]. Available: https://learn.adafruit.com/using-melexis-mlx90614-non-contact-sensors. [Accessed 12 December 2014].

[27] Parr Instruments, "Plain Jacket Oxygen Bomb Calorimeter Instruction Manual," [Online]. Available: http://aui.ma/personal/~S.ElHajjaji/Labmanual/MSDS/InstructionManuals/BombCalorimeterManual2.pdf. [Accessed 14 12 2014].

[28] haake, "Haake Circulators Instruction Manual," [Online]. Available: http://www.geminibv.nl/labware/haake-q-koelwaterbad/haake-f3-series-circulators-manual-eng.pdf.

[29] ReRap, "PCB Heatbed," [Online]. Available: http://reprap.org/wiki/PCB_Heatbed. [Accessed 30 12 2014].

[30] J. Tyndall, Heat Considered as a Mode of Motion, London: Longmans, green and Co., 1868.

[31] Kitt Peak National Observatory, "Building a Thermoelectric Sensor to Measure IR Sky Transparency for Cloud, Cirrus and Dust Detection, at Night," [Online]. Available: http://www.noao.edu/staff/gillespie/projects/cloud-detector.html. [Accessed 02 01 2015].

[32] Adafruit, "Adafruit ultimate GPS logger shield," [Online]. Available: https://learn.adafruit.com/adafruit-ultimate-gps-logger-shield. [Accessed 02 01 2015].

[33] Science Miseim UK, "Warming world: Satellites and weather balloons," [Online]. Available: http://www.sciencemuseum.org.uk/climatechanging/climatescienceinfozone/exploringwhatmighthappen/2point1/2point1point2.aspx. [Accessed 02 01 2015].