Development of Thin Surface Virtual Sensors for Predictive Maintenance

Fábio Augusto Amaral

Work oriented by:

Prof. Dr. Paulo Leitão

Prof. Dr. Nelson Rodrigues

Prof. Dr. Roberto Neli

Bragança

November 2020

Development of Thin Surface Virtual Sensors for Predictive Maintenance

Fábio Augusto Amaral

Dissertation presented to the School of Technology and Management of Bragança to obtain the Master Degree in Industrial Engineering in Double Degree Program with UTFPR

Work oriented by:

Prof. Dr. Paulo Leitão

Prof. Dr. Nelson Rodrigues

Prof. Dr. Roberto Neli

Bragança

November 2020

Acknowledgement

Initially, I would like to thank my father Iran and my mother Lúcia, who always encouraged me to study, always believed in my potential, taught me the values of a correct person, took care of me with all their love, and did everything that was in your reach to make my dreams come true, father and mother, I love you, and none of this would be possible without you!

I would like to thank my beloved wife Roberta, who has always been with me, encouraging me, giving me support, always willing to listen to my cries, and always offered me advice with wisdom, know that you are my safe harbor and we walk side by side in the walk of life. To you all my love, affection and admiration!

I also thank my advisor, professor Dr. Paulo Leitão, who entrusted me with the challenge of this work, believing in my potential, my sincere thanks for his time, for his words and his vision in the face of challenges, and I believe that I take from this experience a partnership for life, thank you professor!

I am grateful to my supervisor in Brazil, Prof Dr. Roberto Neli, who even at a distance and very busy always found time to help me when asked, my sincere thanks!

I thank my pets that stayed in Brazil during this period, Jimi and Max who with their unconditional love always encouraged me with their memories, and now in the final stretch we were able to meet again, half of you is love and the other half too!

I am grateful to my friends and classmates, fellow travelers, Pedro, Matheus, Marina, and so many others that I cannot name, but as you read, you will know that you are the ones who made the days lighter even when these were difficult, thank you friends!

I am grateful to the Portuguese friends that Bragança gave me, to all the staff at

CEDRI, to my researcher friends, in particular Wellington, Thadeu, Piardi, Jonas, Lucas, and Thiago, who showed me the way forward, taking the form of research, when in the laser hours, thank you very much friends, we will see each other again soon!

I am grateful to friends from other courses, and from outside the IPB who made this stay in Portugal look like a family, special thanks to Leonardo, Romulo, Eduarda, Miguel, Dani, Tati, Adriano, Aninha, Heloisa, Pedro, friends who Bragança gave it to me and will always be in my heart!

To the friends UTFPR gave me, it was a four-year journey together, and now the distance has always cheered for me, you are so many and you know that I have always affectionately called you my children, thank you for everything!

I am grateful to UTFPR in Campo Mourão, especially to my professors in the department of electronic engineering who provided me with all the knowledge necessary to become a qualified and professional person in many ways, and ready to face an international challenge.

I am grateful to IPB for welcoming me so well, a place that was my second home during the double degree, a special thanks to my professors in the Master in Industrial Engineering, who brought another look at my personal and professional training with excellent experiences.

Abstract

In the field of manufacturing, metal stamping and plastic injection are some essential procedures, such that companies in this sector need to optimize these processes to gain a competitive advantage. In this sense, this work is part of the On-Surf project, which aims to develop surface modification processes, which promote advanced solutions within the transformation industry through surface engineering techniques.

This work proposes the study of techniques based on virtual sensors, to monitor the temperature of a plastic injection mold in real time. The method makes use of Computer Aided Engineering (CAE) software to model the injection system, mathematical software to adjust the process equations, and an algorithm developed in Python that calculates the value of the soft sensors from the input of one or multiple physical sensors.

The work makes use of case studies of simple metallic surfaces to define the thermal behavior and associate it with a correlation factor. Then apply the techniques developed in the geometry of an injection mold.

Through the use of soft sensors, it is possible to obtain more temperature points about the mold. Such information is extremely important for the predictive maintenance (PdM) of the machine, since it aims to facilitate the operational parameters decision making, reducing the probability of failures, both in the manufactured parts and in the physical sensors themselves, because the technique guarantees the monitoring of the values in real time.

Keywords: Virtual sensor, soft sensor, On-Surf, CAE simulation, fitting curves, white-box modeling, predictive maintenance.

Resumo

Na área de manufatura, estampagem e injeção de plástico são alguns procedimentos essenciais, de forma que as empresas do setor precisam otimizar esses processos para ganhar vantagem competitiva. Neste sentido, este trabalho é parte do projeto On-Surf, que visa desenvolver soluções avançadas dentro da indústria de transformação através de técnicas de engenharia de superfícies.

Este trabalho propõe o desenvolvimento de sensores virtuais, para monitorar a temperatura de um molde de injeção de plástico em tempo real. O método utiliza um software de Engenharia Assistida por Computador (CAE) para modelar o sistema de injeção, um software matemático para ajustar as equações do processo e um algoritmo desenvolvido em Python que infere o valor dos sensores virtuais a partir da entrada de um ou vários sensores físicos.

O trabalho faz uso de estudos de caso de superfícies metálicas simples para definir o comportamento termico e associar a um fator de correlação. A seguir aplicam-se essas técnicas desenvolvidas na geometria de um molde de injeção.

Com o uso de sensores virtuais, será possível obter mais pontos de temperatura sobre o molde. Tais informações são extremamente importantes para a manutenção preditiva (PdM) da máquina, pois facilita a tomada de decisão dos parâmetros operacionais, reduzindo a probabilidade de falhas, tanto nas peças fabricadas quanto nos próprios sensores físicos, devido o monitoramento dos valores em tempo real.

Palavras-chave: Sensor virtual, sensor soft, On-Surf, simulação CAE, curvas de ajuste, modelagem caixa branca, manutenção preditiva.

Contents

List of Tables

List of Figures

Acronyms

R^2	Determination Coefficient
CAD	Computer-Aided Design
CAE	Computer Aided Engineering
CAM	Computer-Aided Manufacturing
CEDRI	Research Center in Digitalization and Intelligent Robotics
CEFAMOL	National Mold Industry Association
CFD	Computer Fluid Dynamics
FEA	Finite Element Analysis
IPB	Instituto Politécnico de Bragança
NASA	National Aeronautics and Space Administration
PDE	Partial Differential Equation
PP	Polypropylene

Chapter 1

Introduction

In this chapter, some data about the mold market in Portugal will be presented, as well as important concepts. What has already been developed in the On-Surf project, and the motivation for the development of virtual sensors will be presented. The objectives of this work, and how its structure is organized will also be presented.

1.1 Background and Motivation

According to the National Mold Industry Association (CEFAMOL), Portugal ranks among the leading manufacturers of plastic injection molds in the world, 8th in the world and 3rd in Europe, with 90% of total production exports [1].

In this sense, investments in technology are being made, such as the On-Surf project, which is an initiative financed by the Portugal 2020 program [2] in the area of surface engineering, aiming to promote a collaboration network between government, industries, and universities towards more competitive industries. The collaboration of the Polytechnic Institute of Bragança in the On-Surf project takes place through the Research Center in Digitalization and Intelligent Robotics (CEDRI).

In particular, this project is part of another 3 projects already accomplished out that were responsible for the development of hardware and software for monitoring the temperature and pressure parameters in thin-films surface sensors applied in metal stamping

and plastic injection molds processes [3] [4] [5]. The scope of analysis for this project is in the field of plastic injection molds.

From the developed project, the first module is a data acquisition system, the module was responsible for acquisition an treatment of the sensors signal [3]. The second module was responsible for acquiring the data from the first module, digitizing and sending the signal to a concentrator using an Olimex ESP32 Poe-ISO micro-controller [4]. The third module is a data concentrator implemented in a Raspberry Pi, responsible for grouping, processing, and storing the data [5]. The developed structure project can be seen in the Figure 1.1.

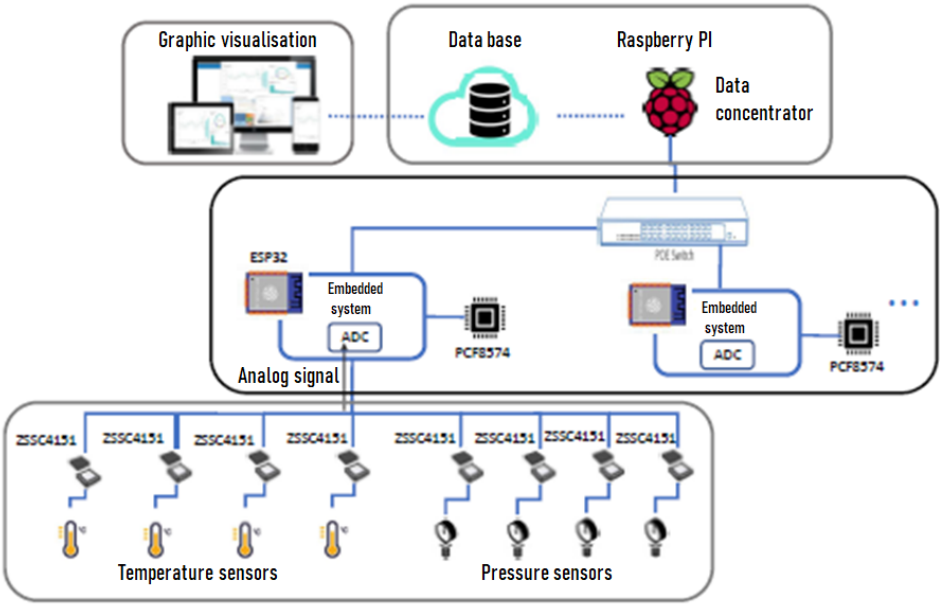


Figure 1.1: On-Surf architecture [4].

The thin surface sensors are an alternative to measure the temperature and pressure parameters inside the mold injection chamber [6]. However, there are still limitations imposed concerning the positioning of the sensors, due to the geometry of the mold injection chamber. Therefore, some points are not possible to be measured even with the thin surface sensors, to overcome this problem, this work proposes the use of temperature virtual sensors.

Virtual sensors are developed by a model that estimates, in real time, the desired

variable from measured real data. The model input data are the variable values that influence the desired variable [7].

The implementation of virtual sensors can use direct or indirect input parameters, in this case, the direct value is available. Virtual sensors are a high-tech alternative for obtaining measuring points with high reliability in locations that are inaccessible through the use of physical sensors [7].

The real-time monitoring promoted by the implementation of virtual sensors provides valuable information from the point of view of predictive maintenance, allows the mold engineer to make decisions regarding machine maintenance before breakages occur [8].

1.2 Objectives

This project aims to develop a thin temperature virtual sensor applied in a plastic injection mold cavity. This work proposes a method that use the real temperature parameter as input information to calculate the virtual sensors value. The study took into account a system with one real sensor as an input parameter for inferring multiple virtual sensors, and another with multiple real input sensors for inferring multiple virtual sensors.

For this purpose, the technique contemplates 3 stages of development, first obtaining the thermal behavior of the model through a CAE software, then adjust the obtained curves and obtain the equations that represent the phenomenon at the desired point using Matlab and Microsoft Excel, and lastly to simulate the use of virtual sensors in software developed in Python, through which it will be possible to observe the performance of the technique in performance and precision parameters.

To reach the objective, it was necessary to develop 5 case studies (including the mold geometry) to understand the behavior of the temperature on metallic surfaces, and then apply it in an injection mold. For this, these steps were followed:

- Use CAE software to develop the physical model and find the temperature behavior in each case study;

- Find equations that represent the results found by the CAE software;
- Develop a code to simulate the virtual sensors, using the equations to obtain the values in the selected points;
- Compare the results in code simulation with the CAE software.

1.3 Document Structure

The present dissertation is divided into 6 chapters an has 2 appendices.

Chapter 1 where this text is inserted, contains a introduction to the main theme, contextualizing the Portuguese mold market, a brief of the On-Surf project and the other works already developed, the problems that motivation to develop virtual sensors, specific objectives, and the document structure.

Chapter 2 presents a state of art, containing a literature review starting with the representativeness of the Portuguese mold industry in the international market, and presenting the thermoplastic injection process, as well as the used machine and its parts. Then it presents the in-mold sensors technology, the virtual sensors concepts and implementation types, the CAE software simulation package and how they work, and finally the heat transfer phenomena.

Chapter 3 describes the methodology to develop the virtual sensor, which consists in 5 case studies, being 4 simple geometries and the geometry of the mold. The technique make uses of 3 steps in a 4 different software, to obtain the expressions who represents the temperature virtual sensors.

Chapter 4 shows the implementation of the methodology to develop the virtual sensor, the software configurations step by step in 5 case studies.

Chapter 5 presents the results obtained in each step of the method, results from the CAE software, the process of adjusting curves and obtaining the expressions that represent the virtual sensors, and the performance test of the equations.

Chapter 6 describes the conclusions, and the future works.

Appendix A has the original project proposal.

Appendix B has the codes developed to simulate the virtual sensors performance tests.

Chapter 2

State of Art

This chapter will describe the technologies and knowledge that guided the research. A brief description of the mold market in Portugal will be presented, as well as, how is the plastic injection process, the relevance of in-mold sensors, what is a virtual sensor and the types, what is and how the CAE software works, and the heat transfer process.

2.1 Plastic Injection

Portuguese mold exports in 2019 reached an estimated value of 682 million euros, which demonstrates the competitiveness of the sector, proving the capacity to adapt to technological and market developments as well as the needs of its customers. Its main markets are Spain, Germany, France, Czech Republic, Poland and the USA. Currently, mold industry has 536 companies, mostly small and medium-sized, with a focus on the design, development, and manufacture of molds and special tools. The sector employs approximately 11,000 workers [1].

The main feature of this industry is its high production capacity, with pillars in cutting-edge technology and specialized labor. The success of its evolution is directly linked to a solid commitment to innovation and high technological intensity, providing a competitive relationship between quality and delivery times, and according to the ability to adapt to customer needs [9].

Portuguese molds have customers in different sectors of activity, but the auto industry is the main customer. However, companies are constantly betting on market diversification, with opportunities in the aeronautics, medical devices, electronics, packaging, energy and environment sectors [9].

The injection process of thermoplastics by mold allows the production of complex parts in the industry. For a part to be produced with good quality, multiple parameters need to be adjusted and configured [10].

The most common defects observed in the injection process are incomplete injections, opaque surfaces, stains from burning, sucking or bubbles, junction marks, adhesion in the nozzle or cavity, oblique marks, silver lines, burr on the part, black spots or color degradation, and excessive deformation or contractions [11].

The final quality of the part depends on many physical tests that involve the injection temperature of the plastic to the time of the injection cycle. The plastic injection process has more than 30 factors that need to be taken into account, such as a property of the material and external humidity, and this whole process is time-dependent, that is, of transient origin [10].

From materials, plastics belong to two major groups, the thermoplastics, and the thermostable. Thermoplastics when heated reach the softening stage and can be molded. The change of state does not alter its chemical composition, this allows that after a cold it can be heated again and reused. Thermostables also soften when heated allowing molding. However, the process causes an irreversible chemical transformation, preventing its reuse [11]. For this work, polypropylene (PP) is the thermoplastic used for testing in the other stages of the work.

The thermoplastic injection process is subject to a phenomenon called contraction, which occurs due to the characteristics of the plastic materials when they are cooled. As is an expected phenomenon, the design of the mold and the machine settings must take into account the contraction process [11]. In this sense, the sensorized mold will bring numerous advantages from the point of view of configuration.

The plastic injection machine is available in different configurations, such as vertical

or horizontal. The machine consists of the injection unit, mold assembly, and clamping unit [12]. Figure 2.1 illustrates an injection molding machine.

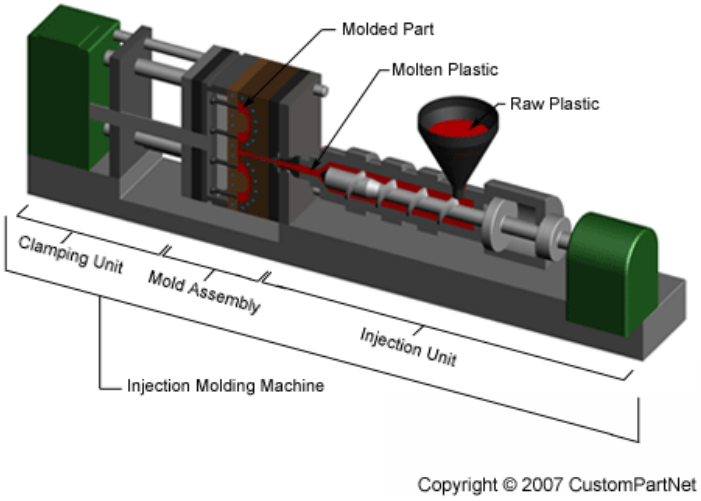


Figure 2.1: Injection mold machine [12].

The injection unit consists of a funnel-fed cylinder, an alternative thread that pushes the thermoplastic into the mold, and a heating assembly. It is responsible for heating the thermoplastic resin, and for compressing the thermoplastic in the injection mold. The alternative screw rotates and moves axially to push the molten thermoplastic through the injection channels of the mold, the thermoplastic is heated by both the resistances and the pressure of the system[12].

The clamping unit is responsible for joining and closing the two halves of the mold, the cavity and the core. During the injection, and until reaching the cooling temperature, the set is closed with the necessary force to keep the halves together. During the extraction the set is opened, removing the solidified injected part [12].

Assembly mold is the structure that contains the mold. The mold consists of several pieces, the main ones being the cavity and the core. The cavity is the part attached to the injection nozzle. The core is the part fixed to the movable set that slides to close and open the mechanism, there are also accessories for the extraction of the part aftercooled[12].

The space formed between the cavity and the core is called the part cavity, or molding

zone, it is in this space that the thermoplastic is injected to form the part [11]. To help in understanding, the figure 2.2 illustrates the side cut in an injection mold.

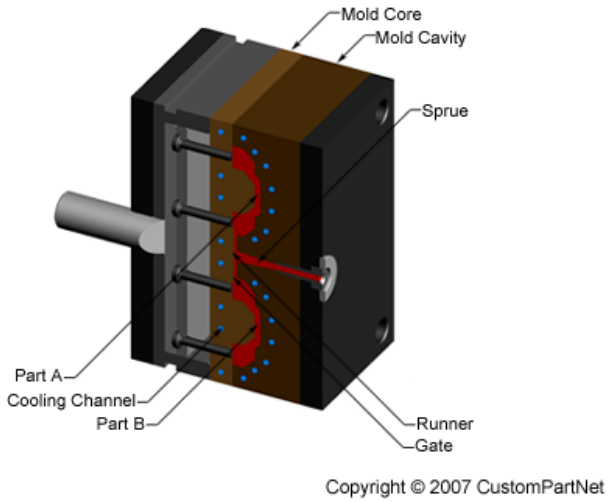


Figure 2.2: Side cut in an injection mold [12].

The injection channels of mold can be cold or hot, what determines the technique is the geometry of the part, both in shape and size. The injection mold is a part made of steel, aluminum, or composite materials [12]. The metal used in the test mold is steel 2738.

Regarding the temperature, the mold is always colder than the thermoplastic, and as the thermoplastic enters the chambers, it starts to cool. The mold can have natural cooling for smaller parts or forced cooling for larger parts or when the injection cycle is shorter. The recommended minimum temperature for the mold is 20 °C and the maximum for amorphous thermoplastics is 70 °C, the referred temperatures are that of the mold, not the liquid in the cooling system. As the injection cycle of the machine becomes continuous, the temperature of the mold tends to stabilize at some value within this range [11]. For the test mold the stabilization temperature is 30 °C.

The injection cycle consists in dosing, injection, packing, cooling and extraction [11]. Typically an injection cycle lasts between 2 seconds to 120 seconds, this value varies depending on the geometry of the part to be injected [12].

The graph of injection process can be seen in Figure 2.3.

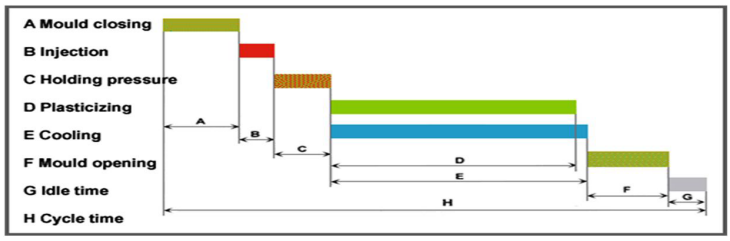


Figure 2.3: Injection cycle [13].

Dosing is the entire process of feeding the thermoplastic resin, heating, and pressurizing the injection cylinder. The thermoplastic resin is placed in the hopper, which is transferred into the injection cylinder by gravity and by the movement of the screw thread. The screw thread directs the material to the injection nozzle. Around the injection cylinder, there are electrical resistances positioned to heat the plastic gradually. As the thermoplastic is pushed into the injection nozzle it melts due to the temperature and pressure under which it is being pushed [14].

The injection occurs when the cylinder is loaded with molten plastic, the thread rotates and advances pushing the molten material through the nozzle, filling the mold [14].

In the packing time, the molten plastic inside the mold tends to return to the nozzle as it cools, at that moment the thread needs to maintain pressure on the nozzle to prevent the return of the molten material, this technique is known as repression. The repression pressure is lower than the injection pressure and serves to maintain the characteristics of the injected material [14].

The cooling step is responsible for the solidification of the plastic inside the mold, it can take a few seconds or more depending on the geometry and size of the part. It is usually the most time-consuming step of the process, during which time the dosage for the next cycle is prepared [14].

The extraction occurs after the cooling cycle of the part, the mold opens and the extractors remove the part. This step can be manual or automated, depending on the need [14].

2.2 In-Mold Sensors

To control the injection process and improve the index of damaged parts, the whole process needs to be very well developed. An important step towards controlling the thermoplastic injection process is to be able to measure the parameters in the mold cavity, where the plastic part is formed. Currently, mold sensing methods are gaining ground thanks to new technologies combined with the advent of Industry 4.0 [6].

The in-mold sensors technique consists in measure the parameters inside the mold chamber. There are different types of sensors that are used to measure the parameters of interest in an injection mold, and choosing the most suitable one is not an easy task [6]. Figure 2.4 illustrates the classification of in-mold sensors.

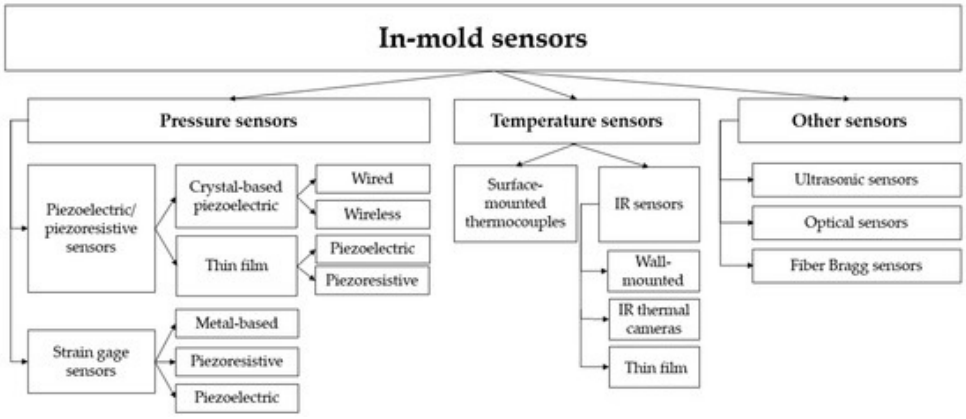


Figure 2.4: In-mold sensors type [6].

The values found by the sensors, serve as parameters to evaluate the materials and components as well as providing information to computational models to predict the behavior of the system [6].

In their work Karbasi and Reiser [15], use temperature (k type) and pressure sensors (piezoelectric) in the mold cavity to improve the process of controlling the thermoplastic injection with the aid of the LabView software. Such an application has proven to be reliable and has good reliability and repeatability in the results of pressure measurements. As the temperature sensor is mounted on the mold body, it is directly affected by the heat exchange, causing a delay between the expected value and the reading presented in

contact with the molten polymer. Another point to be consider is the environment where the sensors was positioned, there have movable parts and constant variation in temperature and pressure. Also, exposure to abrasive materials can change the measurement of parameters over time.

As the proposal of the On-Surf project is related to the study of surfaces, a focus will be given to in-mold thin-film sensors. The development of these sensors isn't a easy task since they are minimally invasive, the thickness parameters are minimum, around to 5-8 μm, and perform well in hostile environments with high temperatures. NASA has been developing and using thin-film thermocouples and strain gauges to measure metals, ceramics, and advanced ceramic-based composites. Initially, the sensors were developed separately, and later they were integrated on the same substrate forming a multifunctional temperature and strain gauge sensors. The sensors developed were tested in various material systems for jet aircraft and space-based engine applications at temperatures raged from 1000 - 1500 °C presenting. A good feature of a temperature thin-film sensor is the excellent ability to measure in steady-state as well as in transient response, it causes less thermal obstruction as well as physical flow and presents a good sensitivity[16]. Figure 2.5 illustrates an static strain gauge and a multifunctional temperature and strain gauge thin-film sensor.

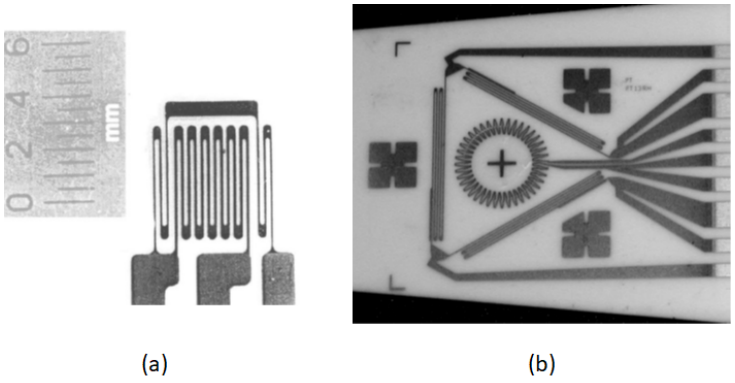


Figure 2.5: (a) Static strain gauge, (b) Multifunctional sensor prototype [16].

.

2.3 Virtual Sensors

Before discourse over virtual sensors, it is important to know what a physical sensor is and how it works.

Sensors are devices that are sensitive to some form of environmental energy, such as luminous, thermal, kinetic, that relate data about a quantity that needs to be measured, such as temperature, pressure, speed, among others. However, a sensor does not always have the necessary electrical characteristics to be used in a control system. Commonly, the output signal from a sensor needs to be handled before it is available to the control system [17].

A sensor can also be analog or digital. An analog sensor reports any value in the output signal over time within its operating range, an example of this type of sense is that of temperature. A digital sensor assumes only two values, zero or one. This type of sensor does not normally translate a quantity directly, but the answer can thus be presented by its control circuit or transducer. Digital sensors can be used as passage detection, an encoder to determine position, distance, or speed [17]. Figure 2.6 illustrates the block diagram of a generic sensor that depends on power to work.

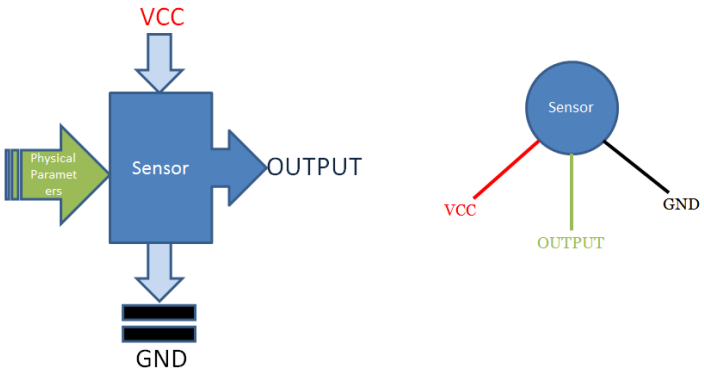


Figure 2.6: Generic sensor block diagram [18].

In a large number of applications, physical sensors are sufficient to extract the parameters of the system, but there are situations in which direct measurements are not viable, either due to the cost, the aggressiveness of the environment, the sampling rate involved,

the non-online measurement option, or the desired magnitude. One way to solve a large part of these problems is through virtual sensors [8].

Virtual sensors (also know as "soft sensors") are an powerful tool for measure quantities in many different industrial fields application, involving power plants, nuclear plants, refineries, chemical plants, urban and industrial pollution monitoring, as an example [8].

In fact, virtual sensors are a great instrument to provide indirect measure of quantities, in environments who the regular sensor cant be installed, by problems on the geometry, to no disturb the original system [19].

From an economic point of view, virtual sensors present a series of attractive elements, as they are a cheap alternative compared to expensive hardware, work in parallel with real sensors, provide useful information from fault detection making the system more reliable, can be easily implemented on existing hardware, estimate values in real-time, and fault tolerant in hostile environments [8].

A virtual sensor can be obtained by modeling systems, and they can be classified into 3 types, the white box, gray box, and black box [7], as can see in Figure 2.7.

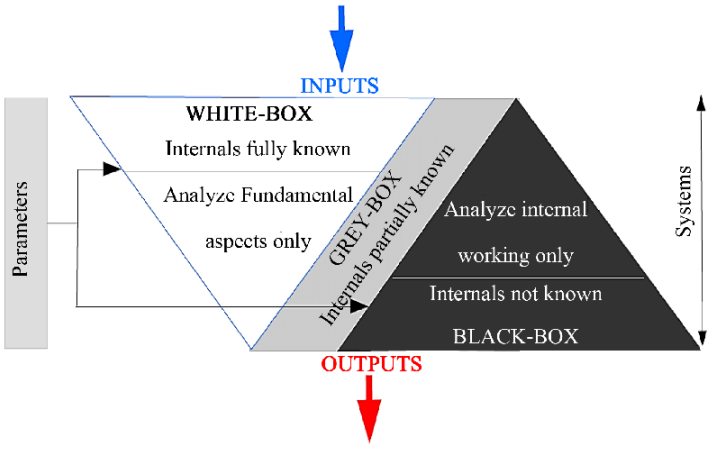


Figure 2.7: White-box, gray-box, and black-box diagram [20].

White-box, or model-driving modeling, represents the physical nature of the process and can be described mathematically, all physical phenomena involved in the system are known, the inputs and the outputs. The model is represented by mathematical equations, and it is the most difficult to be represented. For the approach to work, all the phenomena

involved must be clearly understood and taken into account [8][21].

The white-box approach is usually aided by software to find the plant model, the equation model, or a co-relation factor. Such an approach is found in the work of Monteiro et all [22], in which the research makes use of a mathematical model provide by Matlab software to infer the temperature value in a greenhouse that has the temperature and voltage correlation provided as an input parameter by a real sensor.

However, this approach has limitations, for a more complex system, a complete understanding of the fundamental mechanisms that permeate the process is practically impossible to obtain [7].

Gray box modeling is a relatively new approach and is generally used when the system is highly complex, so only partial system modeling is possible, however, there is a history with a lot of operational system data available. This model makes a hybrid approach possible, on the one hand, there are data from a mechanistic model and the complement comes from artificial intelligence techniques [21].

Black-box modeling, also called data-driven, is built using regression techniques with neural networks or neuro-diffuse networks, without any internal knowledge of the process [23].

In their work, Soares and Oliveira [24], use artificial intelligence to infer the temperature of an aluminum reduction furnace. The algorithm used is of the Multilayer Perceptron type, and a plant provides a lot of indirect information that is used to train the model and thus establish the working pattern of the virtual temperature sensor. The motivation for the work is the cost of the actual sensors and the availability of historical process data.

The most used techniques for the development of virtual sensors are the white-box and the black-box. The white-box is used for the clarity of the process, and the black-box for the ease of using collections of data to represent the system without the need to know it [25].

2.4 Simulation with CAE Software

The study in the field of engineering has for some time made use of software as development tools. Computer-Aided Design (CAD), Computer-Aided Manufacturing (CAM), and Computer-Aided Engineering (CAE) are software systems that assist in the product development and improvement process, allowing adjustments and corrections to be made before conception, reducing development costs [26].

With the increasing evolution of computational resources, associated with simulations based on numerical methods, simulating the conditions of use of a product became more accessible. In this sense, the CAE software is a tool that helps the development of a product through static, fluid, dynamic, thermal, electromagnetic, acoustic, and other analyzes [27].

The CAE software makes use of the geometry developed in CAD software. However, the integration of different tools can cause some compatibility conflicts, which are constantly being updated to mitigate such occurrences [26].

For a geometry to be simulated, it goes through three steps within the CAE software, pre-processing, processing, and post-processing. The pre-processing stage defines the materials involved, interactions with external elements, applied forces, temperatures, fixations, mesh creation, elements and nodes, among others [28]. The processing step is where the computer performs the calculations and generates the results available for the post-processing step. The post-processing stage displays the results through visualization and diagnostic tools so that engineering professionals can evaluate the model, checking whether it can be approved or if it is necessary to change the design [27].

CAE techniques are implemented with different mathematical methods, the most common of which are Finite Element Analysis (FEA), and Computer Fluid Dynamics (CFD) [27].

The strategy of this work is the thermal analysis without taking into account the fluid flow, therefore, it does not contemplate the use of CFD, which simulates processes that involve fluid flow.

The FEA analysis offers an approximate result using numerical resolution techniques. The method considers that a component can be described as a set of several small components, called elements, that have simple geometry and defined physical patterns. These elements constitute the mesh, which represents the geometry to be studied. The elements can be of two dimensions or three dimensions and are related to each other through the connection nodes [27]. Figure 2.8 represents the FEA mesh example.

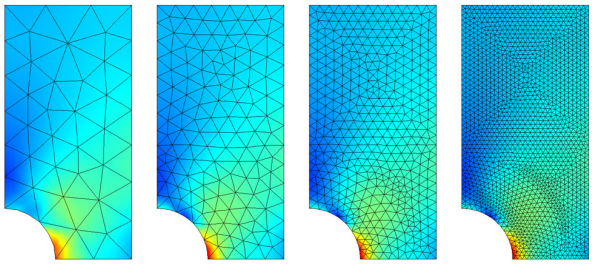


Figure 2.8: FEA mesh example [29].

The FEA method is used to simplify the resolution of problems known as boundary value problem [30], in the context of this work the differential equation of heat. As the exact resolution of the differential heat equation is very complex and is not the main focus of this work, the alternative found is the approximate solution using the FEA method.

In the context of mold development, it is no different, many studies have been carried out using CAE software to optimize the process through simulations. Thermal behavior is a focus of great interest in the researched bibliographies, in 1992 Himasekhar at al. [31], makes use of a CAE tool to dimension the cooling system of a mold. The study by Almoria et al. [32] contemplates the thermal behavior of an injection mold using CAE software. In his study Raposo [33] performs a numerical analysis of the temperature effect of the injection system of a mold using CAE tools, its study traces the thermal profiles involved during the injection process. The Stout and Billings [34] study details the heat transfer in a 2D and 3D model in electronic components, making use of transient analysis with ANSYS software.

2.5 Heat Transfer Phenomena

This section will show the physical phenomena that involve heat transfer, and that were fundamental for the development of the simulations. To understand the physical phenomena that guide the thermal analysis of the development of a virtual sensor, it is essential to know what are the heat transfer processes involved and how they behave. In the context of heat transfer, there are three main types, namely conduction, convection, and radiation. Such phenomena will be explained below.

2.5.1 Conduction

The conduction heat transfer process can be described as the energy transfer from the most energetic to the least energetic particles due to the interaction between the particles. Conducted heat transmission is usually more present in solid materials but also occurs in fluid (liquid and gaseous) [35].

The conduction heat transfer is illustrated in Figure 2.9.

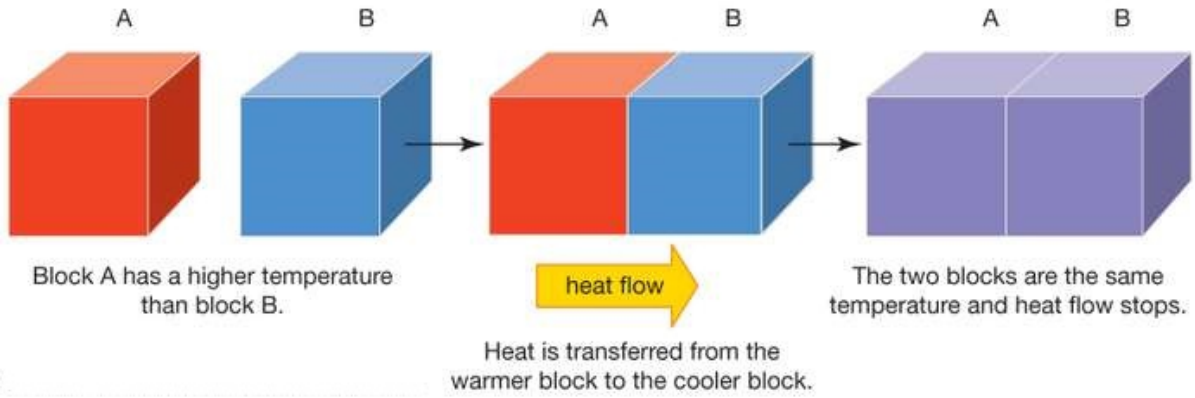


Figure 2.9: Conduction heat transfer [36].

Heat transfer between bodies is influenced by several factors such as:

- Body geometry;
- Section thickness;

- Materials involved;
- The thermal differential between the bodies;
- The thermal load applied;
- The thermal losses involved.

To quantify the heat exchange process, it is necessary to have the appropriate rate equations, through these equations it is possible to obtain the amount of energy transferred per unit of time. The equation that defines conduction is known as Fourier's law [35], and is presented in Equation 2.1.

$$q_x'' = -k\frac{dT}{dx} \tag{2.1}$$

Where $q_x''(W/m^2)$ is the thermal flow in the x direction per unit area perpendicular to the direction of the transfer, being proportional to the temperature gradient dT/dx in this direction. The parameter k is the material's thermal conductivity $(W/m \cdot K)$.

Considering the permanent regime, with the linear temperature distribution, the Equation 2.1 can be rewritten as follows Equation 2.2.

$$q_x'' = kA\frac{T_1 - T_2}{L} \tag{2.2}$$

where:

$q_x^{"}$ is Rate of heat transferred per unit time $[W/m^2]$

k is the thermal conductivity of the material $[W/mK$ or $W/m°\text{C}]$

A is the transfer heat area $[m^2]$

$T_1 - T_2$ is the temperature difference between the ends of the material [K or °C]

L is the thickness of material

However, the presented equations were developed considering a permanent regime. Therefore, the approach needs to be adjusted to understand the phenomenon associated with the study of the injection mold.

A transient regime is characterized by changing conditions over time. Typically a problem is of transient origin when some system boundary condition is changed. If the heat gradient inside the solid can be displaced, there is the global capacitance method. In conditions where the temperature gradients are negligible and the heat transfer inside the solid is one-dimensional, exact solutions can be used to describe the behavior of the temperature as a function of time. For more complex conditions, the finite difference and finite element method are used to find the dependence between temperature and time inside the solids, as well as the heat transfer rates in their contour [35].

The case study of the mold fits the approach of the finite difference method, due to the complexity of the analysis imposed by the geometry of the part.

For this solution, there is an explicit method and the implicit method. Both methods are solutions that can be solved numerically, taking into account the characteristics of each node involved in the analysis [35].

2.5.2 Convection

The phenomenon of heat transfer by convection occurs between the fluid in motion in contact with a surface, where both are at different temperatures. The heat transfer by convection covers 2 modes, the energy transfer by the random molecular movement (diffusion), and the energy transfer through the global or macroscopic movement of the fluid. The movement of the fluid is characterized by a large number of molecules being collectively or as an aggregate at any time, and in the presence of a temperature gradient, it promotes heat transfer. The term convection is commonly used to refer to cumulative transport and advection to transport by the global movement of the fluid [35].

Figure 2.10 illustrates the process of convection heat transfer.

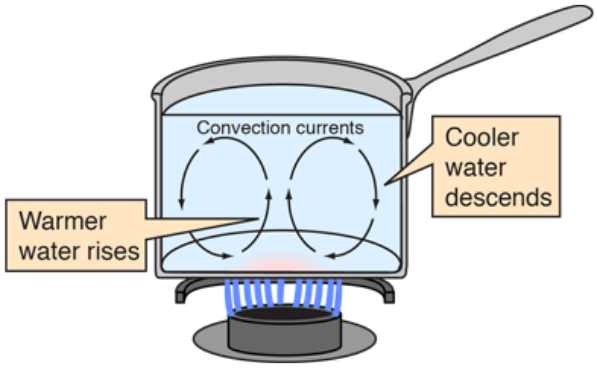


Figure 2.10: Convection heat transfer [37].

When heating the fluid, the portion located near the heat source absorbs energy. As the temperature tends to seek equilibrium from the warmest to the coldest source, convection flow is created, the molecules with the most energy go up and the ones with the least energy go down to stabilize the system, because of that, the fluid moves.

Convection can be natural or forced. Forced convection occurs when the flow occurs through an external force, such as a fan or a pump. In natural convection, the flow occurs by the induction of buoyant forces, originated from different densities and by variations in the temperature in the fluid, an example of this phenomenon is the heat dissipation in a printed circuit board arranged vertically and exposed to the air [35].

The equation that represents the heat transfer rate by convection is also known as Newton's law of cooling. Regardless of nature, whether it is forced or natural, the Equation 2.3 presents that of convection heat transfer.

$$q^{"} = hA(T_s - T_\infty) \tag{2.3}$$

Where:

$q^{"}$ is the convection heat flow $[W]$

A is the area of the transfer surface $[m^2]$

h is the convection heat transfer coefficient $[W/m^2K]$

$T_s - T_\infty$ is the temperature difference between the surface and the surrounding fluid [K]

2.5.3 Radiation

The transfer of heat by radiation happens by electromagnetic waves, or photons. This means that there is no need for a material form for the transfer phenomenon to happen, and it performs better in a vacuum. An example of this phenomenon is the radiation emitted by the Sun in the form of heat that reaches Earth [38]. Another example is the radiation emitted by the flame of a stove when heating a pan. Figure 2.11 illustrates the radiation heat transfer.

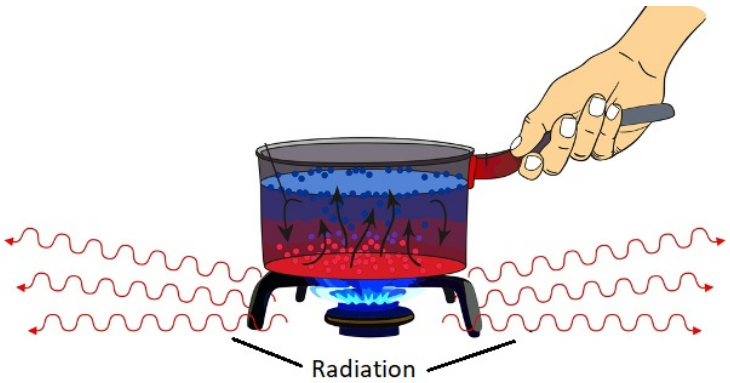


Figure 2.11: Radiation heat transfer [39].

A characteristic of this phenomenon is that every body with a temperature higher than 0 K, emits thermal radiation. Although the focus is on solids, radiation is also emitted by liquids and gases. The radiation rate of a body depends on the geometry of the surface and the type of material that constitutes it [35]. Thermal radiation is governed by Stefans - Boltzmann law , the Equation 2.4 describes the behavior for a blackbody, which is a perfect radiator of thermal radiation.

$$E_n = \sigma A T_s^4 \tag{2.4}$$

Were:

E_n is the emissive power [W]

σ is the Stefan-Boltzmann constant $[5,67.10^{-8} W m^2 K^{-4}]$

A is the surface area $[m^2]$

T_s is the absolute temperature of surface [K]

For real bodies (gray bodies), the Equation 2.5 describes the behavior.

$$E_n = \varepsilon \sigma A T_s^4 \tag{2.5}$$

Where ε is the emissivity of the material normally with values $\leq$1.

2.5.4 Heat Equation

The heat equation, or thermal diffusion equation, is widely studied in mathematics and physics, and describes the process of conducting heat in isotropic and homogeneous solids in a two-dimensional or n-dimensional space that does not have internal heat sources. The heat equation is a partial differential equation (PDE) that represents the temperature value in a steady or transient solid body. The heat equation is derived from the Fourier Law of thermal conduction [40].

Equation 2.6 represents the heat equation .

$$\frac{\partial u}{\partial t} = \alpha \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} \right) \tag{2.6}$$

For its resolution, some boundary conditions need to be defined, as the Dirichlet, Neumann, Robin, Cauchy, and mixed condition, according to the desired analysis.

With the advance in computing methods, a numerical solution to the heat equation is an valuable option. However, if the initial boundary conditions not obey an infinite set of compatibility conditions, singularities will be present in the corners of the initial space domain. For 1-D heat equations, the impacts of these singularities are short-lived, but they can cause severe losses of numerical precision in some transient cases. For long transient times, the error becomes irrelevant, but if the instant to be observed is very small, the error can severely alter the measurement. One way to get around this problem is to choose the method that meets the instant of time desired or apply a correction technique [41].

Chapter 3

Methodology do Develop Virtual Sensors

This chapter describes the methodology chosen for the development of virtual temperature sensors and the programs that were used to obtain the models. The elements that guided the research for this approach will also be described.

3.1 Strategy Approach

The approach taken to solve the problem was driven by the nature of the problem. The behavior of the temperature in an injection mold is of transient origin, since the heat involved in the process is always varying over time, through a heating and cooling cycle.

Another point to be considered is that the analysis is also spatial, since one of the interests is to be able to measure the temperature in any part of the mold geometry. So the problem can be characterized as dependent on time and position.

One of the several challenges encountered was the fact that there is no historical data on the use of the equipment. To use a black-box or a gray-box model, it is necessary to have a significant data set, in which case it is not available. This means that the topology chosen for the development of the virtual sensor was the white box, or model-driving modeling. Thus, the phenomenon associated with the virtual sensor is represented by

employing a mathematical equation.

After understanding the heat equations, starting with the stationary regime, and progressing to the transient regime, it was found that it would be extremely complex to describe the surrounding phenomena in the mold mathematically, due to the geometry and the characteristic of the thermoplastic injection process.

So, to obtain the referred equations, four stages of analysis are necessary,the sequence used to develop the virtual sensors can be seen in the figure 3.1.

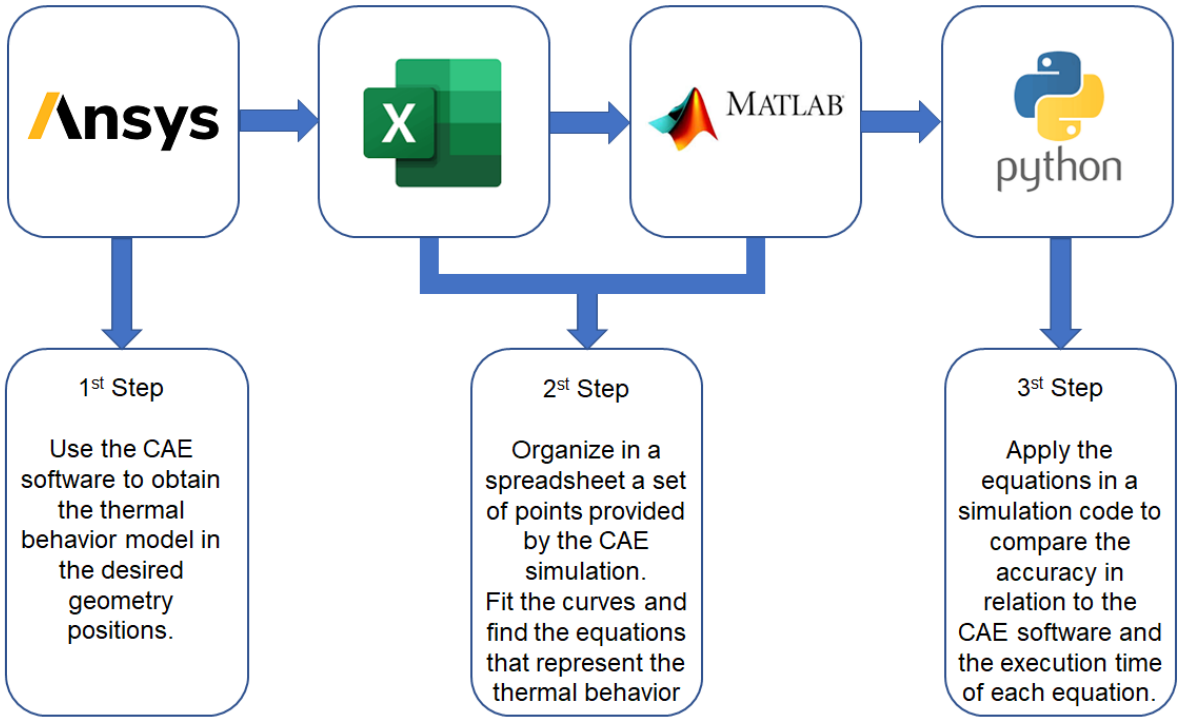


Figure 3.1: Design steps of virtual sensors.

For industrial purposes, it is very important that a process or technology can be replicated and allows scalability. In this sense, it is understood that it is necessary that virtual sensors can be applied to any geometry in an easy way, with few adjustments to the model and in the shortest possible analysis time, even though understand that each mold developed has unique and individual characteristics. Because of these characteristics, was opted to perform simulations of thermal processes with the aid of CAE software.

The choice of CAE software was a delicate step in the process, at least 3 software were

considered to perform the task, however, the software chosen to perform the analysis was ANSYS, as it is a recognized engineering software, being present in several consulted bibliographies, as well as IPB has the software license.

With the defined software, it was necessary to choose the type of analysis to perform the simulation, which was also done based on bibliographic criteria, and the Transient Thermal tool was defined.

To verify the hypothesis, the approach considers the following 5 case studies:

1. A metal plate with one heat gradient;
2. A metal plate with two heat gradient;
3. A metal cylinder;
4. A metal pipe;
5. A injection plastic mold.

The first four case studies consist of surface analysis of simple geometries. The fifth case study is the application of the best techniques learned from the first case studies to a plastic injection mold.

For code level tests, two approaches have been proposed, the first being a real sensor providing the input parameter for inferring values from multiple virtual sensors. The second approach is the input of multiple real sensors and the output with multiple virtual sensors. Figure 3.2 illustrates the test software approach.

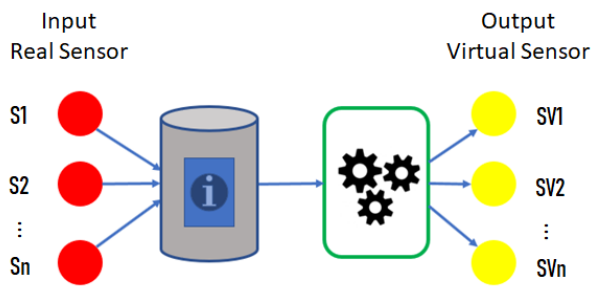


Figure 3.2: Software approach for virtual sensors.

The first approach is the most simplified, as there is no need to worry about the relationship between other real sensors and their positions. The implementation is not so simple, in this case, 1 real sensor providing data for the inference of n virtual sensors.

For the second approach of virtual sensors, multiple inputs, and multiple outputs, the problem presents a certain degree of complexity. Establishing a correlation factor between the positions of the actual sensors is not a trivial task. To clarify the idea Figure 3.3 show a plate with a distribution with real sensors and virtual sensors.

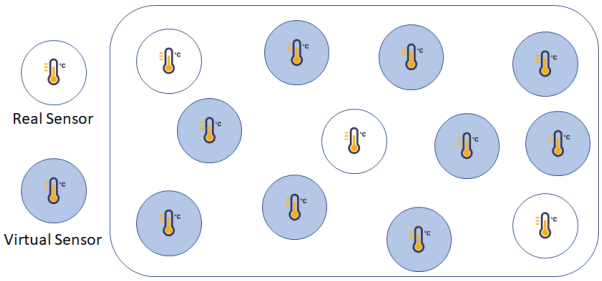


Figure 3.3: Example of real and virtual sensors in a plate.

In this sense, the case studies of the plate, cylinder, and tube were ways to observe the behavior of the sensors from the point of view of temperature, time, and position.

The approach of the plate with a heat source brings some ways to solve the problem, while the cylinder and the pipe others. To solve the problem next to the mold, what was done is an overview of the methods used in the simple geometries, resulting in a code that solves the algorithm for multiple entries under restrict initial conditions.

One of the initial conditions verified is that every sensor present on the surface must be modeled, including the real sensors. The strategy for implementing the code depends on the position of the actual sensors. Such details will be explained with a wealth of details in the development chapter.

3.2 CAE Simulation Sequence

The CAE software was used to obtain productivity associated with reliability. In this project, the tool should play an important role in modeling the case studies, as it has

considerably facilitated the number of manual calculations that would need to be made. The ANSYS software was responsible for the simulation of the 5 case studies and provided valuable information on the behavior of the temperature in different specifications. The results obtained through these simulations were a basis to find as equations that represent the virtual sensors.

To facilitate the understanding of the steps taken to create the simulation of the models within ANSYS, the Figure 3.4 illustrates the workflow diagram.

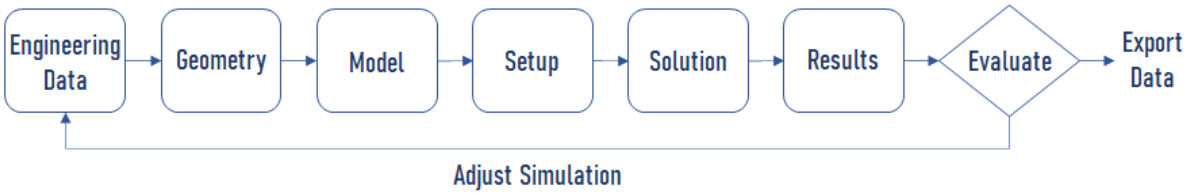


Figure 3.4: ANSYS work flux.

For a good result in ANSYS, it is important to know the physical phenomena involved, so that the boundary conditions are correctly configured. This analysis will provide the temperature result at predetermined points.

The thermal analysis could have been performed in more than one way within the program, the most traditional, and the one adopted is through the Workbench. The Workbench is a console that groups together the analysis contained in ANSYS. The Transient Thermal tool allows the simulation of heat sources and losses due to contact, conduction and radiation, as well as the analysis at a specific point.

The first step is the *Engineering Data* configuration, this option is used to configure the material associated with the geometry. *Geometry* is a step that deserves attention, as it contains the characteristics that will be studied. For the first 3 case studies, the geometries were simple and were developed within the CAD software that comes with the ANSYS software.

The geometry of the mold used corresponds to a mold developed for the tests in the On-Surf project. The geometry of the mold in question was complex and had many details that were simplified so that the analysis could be done.

For ANSYS to be able to perform the calculations, the geometry must be represented by a mesh, this is configures in the *Model* option. The complexity of the mesh generation is directly linked to the reliability of the response, as well as the use of the computational resource. In this phase, a good mesh is a mesh that represents the phenomenon at a compatible computational cost. In this sense, complex simulations such as mold geometry take hours, and for this reason there needs to be a balance between precision and simulation time.

For the simulation to be reliable it is important to understand the boundary conditions, configured in the *Setup* option. The boundary conditions for the case studies involve the material and geometry of the parts, the sources of heat present, the sources of heat loss, the simulation time, the initial temperature of the part, and the duration of the simulation. It is important at this stage to adjust the parameters so that they represent the phenomenon that you want to reproduce through simulation.

The results are obtained by configuring the *Solution* option, adding a component that represents the desired results, in this case, the temperature.

After these steps, the simulation can be run, and the points sampled on each temperature probe was showed in the *Results* options. For the temperatures to be measured at the desired points, it is important to create the coordinates for the points, then these coordinates must be associated with the result of each probe. It is common during the simulation setup process to revisit the previous steps to obtain a more refined result.

After running the final version of the simulation, the results can be exported. The available option is in a text file or a table.

Full details of the case study settings are available in the implementation chapter.

3.3 Fitting Curves Approach

For case studies, plate with 1 temperature gradient, plate with 2 temperature gradients and cylinder, with the data provided from the ANSYS simulations, 3 different types of equations were obtained for each sensor. Two polynomial equations, one of degree 8 and

one of degree 6, and a logarithmic equation.

For case studies of tubes and molds, the approach was broader, involving polynomial equations of different degrees and the logarithmic function for each sensor in the studies cited. This technique was used to compare the performance of each equation when submitted to testing developed code. These equations were chosen according to the characteristic of the temperature curves obtained.

To find the curves associated with each sensor, two programs were used, Microsoft Excel and Matlab. Through Microsoft Excel, the data exported by ANSYS was correctly organized for later use in Matlab.

The spreadsheet with the correct tabulated values was imported into Matlab, and through the Curving Fitting Toll Utility, the equations of the curves were determined. With this utility, polynomial equations were obtained. The Figure 3.5 illustrate the fitting toll.

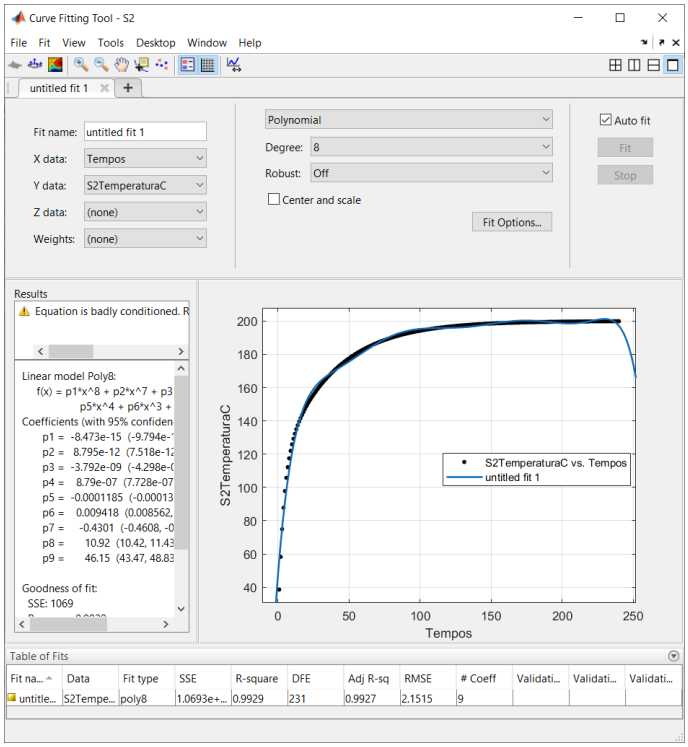


Figure 3.5: Matlab fitting toll utility example.

The Curving Fitting Utility provides information on the determination coefficient (R^2).

Through R^2 it is possible to verify how much the equation represents the original curve, the greater the degree of the polynomial function, the greater the representativeness through the equation.

From the point of view of the representativeness of the curve, a higher degree function is optimal. On the other hand, it is important to take into account the computational cost of the solution, as well as the precision of the method.

To obtain the logarithmic equation, Excel was used to fitting the curves. The Excel provide the R^2 parameter too, that was used to compare the curve representation. The Excel fitting code technique's can be seen in the Figure 3.6.

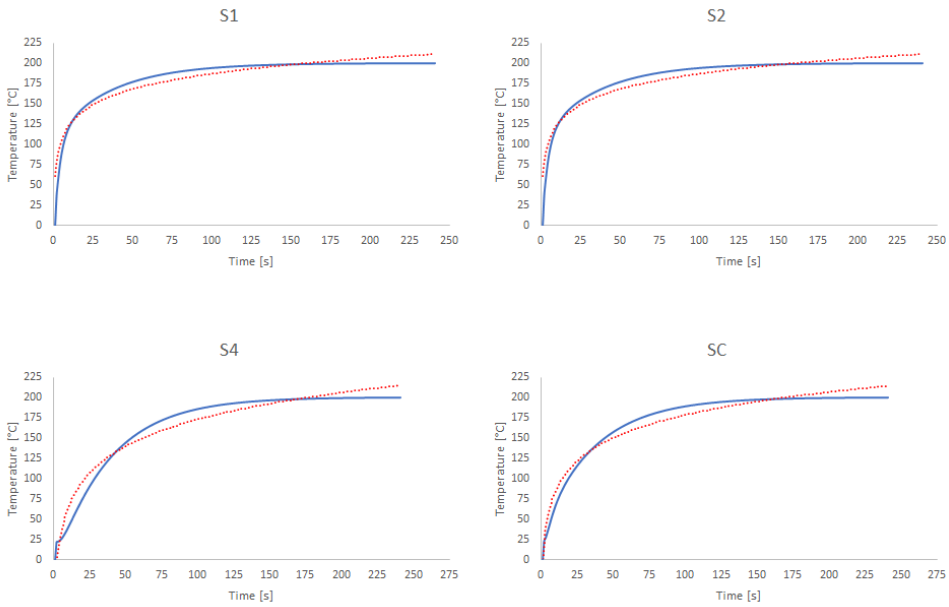


Figure 3.6: Logarithmic equation Excel fitting example.

3.4 Simulation Code for Performance Tests

To evaluate the equations obtained in the curve fitting process, it was necessary to develop a code to run these equations and provide performance information.

The requirements to be achieved with the simulation code involve running an application cycle within the range of the acquisition rate of a physical sensor, that is, achieving an

update rate for the virtual sensor close to or below the real sensor. Another requirement to be met is to calculate several points of a virtual sensor during the parts injection cycle.

For the code simulation, some implementation techniques were tested to try to reach these objectives.

The diagram presented in Figure 3.7 illustrates the flux of data in the algorithm.

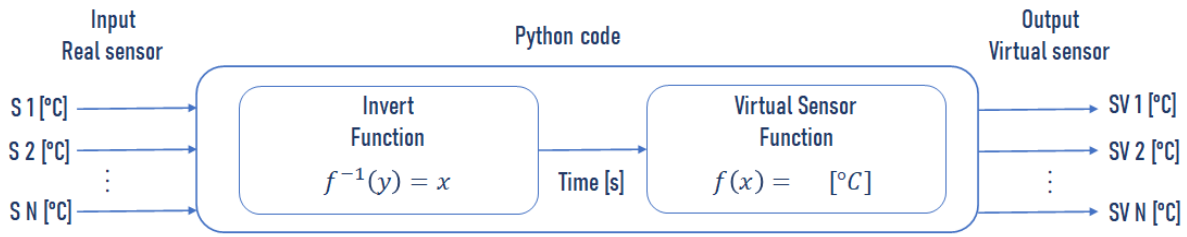


Figure 3.7: Virtual sensor code diagram.

The diagram is a simplified representation of the code, the main idea being that through the input of the temperature of a real sensor, the algorithm finds the time referring to that temperature through an inverse function. With the time obtained by the inverse function, the values of the virtual sensors are calculated at that time. As for the operation of the algorithm, codes were developed to represent some situations. Code variations are derived from two main codes, which can be seen in the following list of topics:

- Single real sensor temperature as input;
- Multiple real sensors temperature as input.

For the techniques to work properly, the physical sensors must be also modeled, that is, it is necessary to obtain the equation in the positions of the real sensors.

The initial version started from the simplest premise, being 1 real sensor providing the time parameter for the calculation of one or multiple virtual sensors. This topology will be identified as 1 for n.

Figure 3.8 illustrates the diagram of a real sensor as input and multiple virtual sensors as outputs.

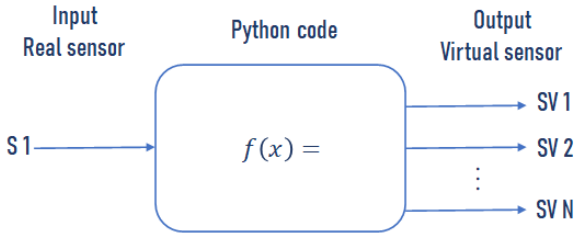


Figure 3.8: Virtual sensor diagram for 1 real input and multiple inferred outputs.

Considering the temperature as input, it is possible to find the input time value by applying the corresponding inverse function. The inverse function must be the function of the position that is making the actual reading since the position of the sensor directly interferes with the distribution of heat on the surface.

The version of the code for multiple entries presents a challenge for the integration of real sensors in different coordinates in the part geometry. The geometries studied in the case studies brought more clarity on how to carry out the implementation through the software. Figure 3.9 illustrates the diagram of a multiple real sensor as input and multiple virtual sensors as outputs.

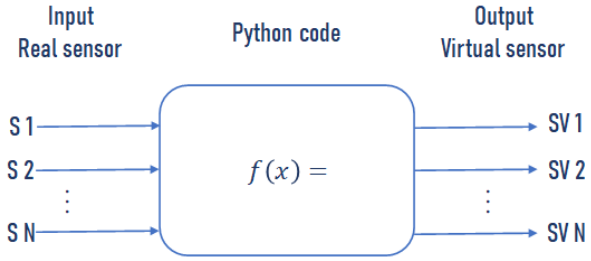


Figure 3.9: Virtual sensor diagram for multiple real inputs and multiple inferred outputs.

The relationship of physical and virtual sensors according to their neighborhood occurs by approaching the modeling of all sensors present in geometry, in other words, all the points to be monitored must necessarily be represented by an equation.

The approach proposed in the case study of the plate is to position 2 real sensors parallel to the heat source. In this approach, it is enough to average the temperatures and perform an inverse operation to find the time, and then solve the set of equations in

the model.

In the approach of the cylinder and the pipe, the geometry does not have parallel sensors but rather arranged along with the bodies, which implies a perfect synchronization in the process of solving the inverse roots.

For the mold, the considerations made for the previous models were taken into account. In this case study, the sensors were distributed throughout the injection chamber, which allows using some variations of the models found in simple geometries.

Chapter 4

Implementation

This chapter describes all the steps necessary to obtain the equations that represent the virtual sensors in their respective geometries, the difficulties encountered, and the proposed solutions.

For each study case, the workflow begins with the creation and simulation of the models using the ANSYS software, then the results obtained are formatted and organized in Excel spreadsheets, next to these data are used in Matlab to adjust curves and results of the equations, and finally the development of the code for the tests with the customized equations for each model.

The entire sequence of procedures and software required to obtain the virtual sensors is extensive and complex. As there are many similar procedures for all case studies, the procedures that are repeated will be described once and will be referenced when necessary.

For the ANSYS models, all materials used in the simulations were extracted from the ANSYS database or catalogs of manufacturers or suppliers. The dimensions used in the case studies, plate, cylinder and tube, were extracted from supplier catalogs.

4.1 ANSYS General Configuration

The first step to begin the simulation is start the Workbench an add the Transient Thermal analysis, just like in Figure 4.1.

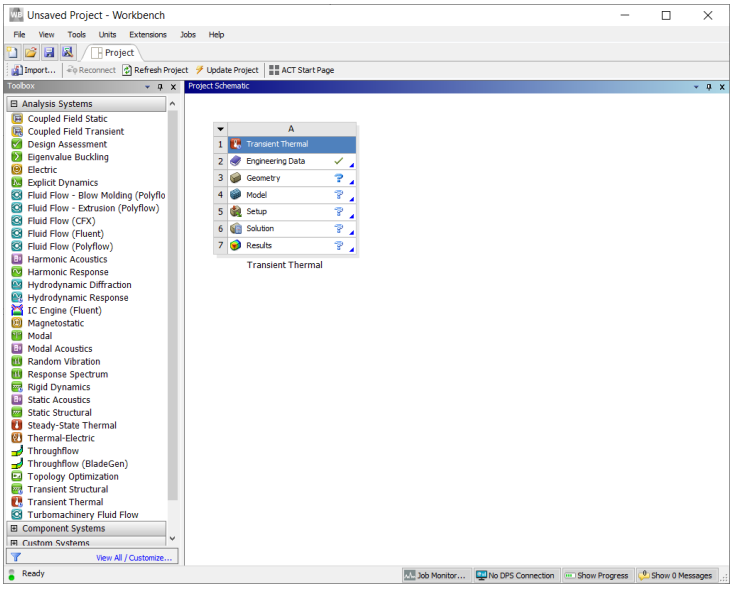


Figure 4.1: Workbench with transient thermal analysis loaded.

The second step is select the materials involved in analysis. That option was made by clicking in Engineering Data, then Edit, as show in the Figure 4.2.

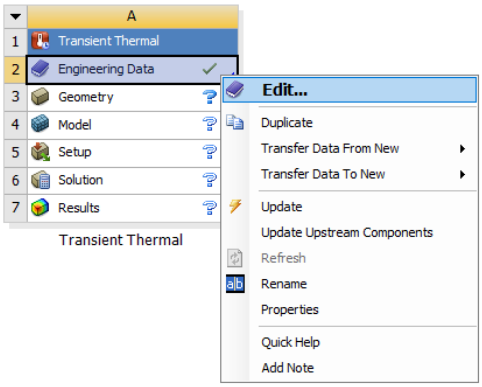


Figure 4.2: Engineering data.

For an analysis to be reliable, an important step is to choose which materials are involved in the simulated. In this sense, ANSYS has a very large material database,

which covers materials in most states of matter, solids, liquids, and gases, as well as composite materials and chemical substances.

The Engineering Data option can be seen in Figure 4.3.

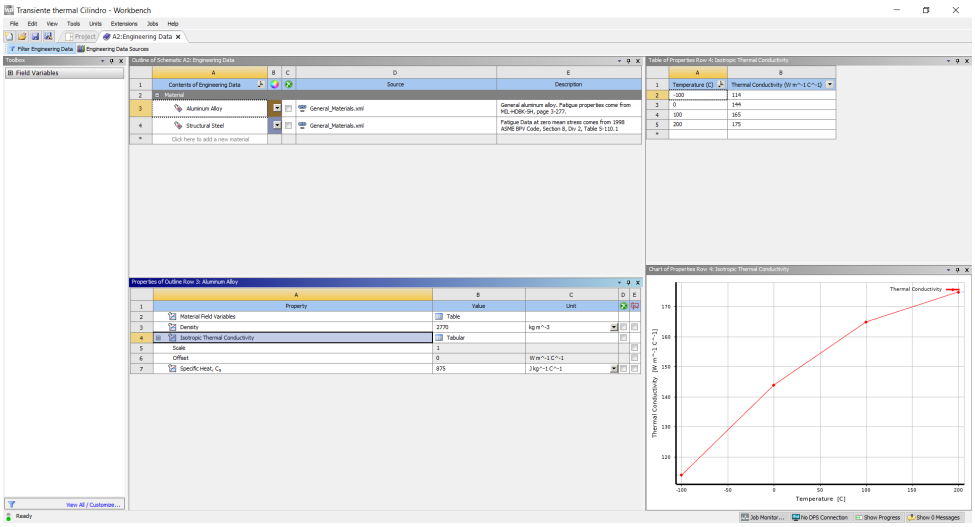


Figure 4.3: Engineering data sources.

If the necessary material does not exist in the database, it is possible to create a material and add the necessary properties.

This procedure can be seen in Figure 4.4.

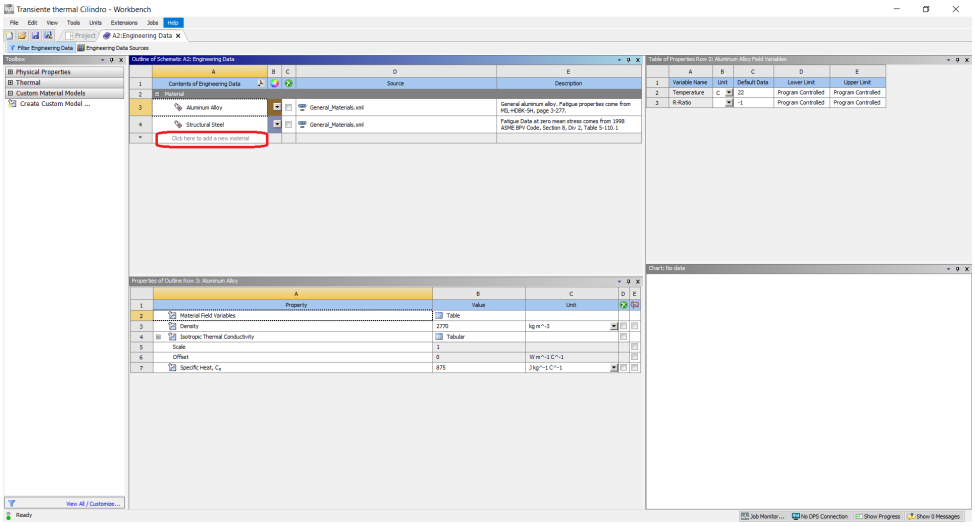


Figure 4.4: New material.

The third step is import or create a geometry. The geometry can be imported from others CAD softwares, but the ANSYS has 2 CAD environments available, the SpaceClaim and the DesignModeler. The Geometry options can be seen in Figure 4.5.

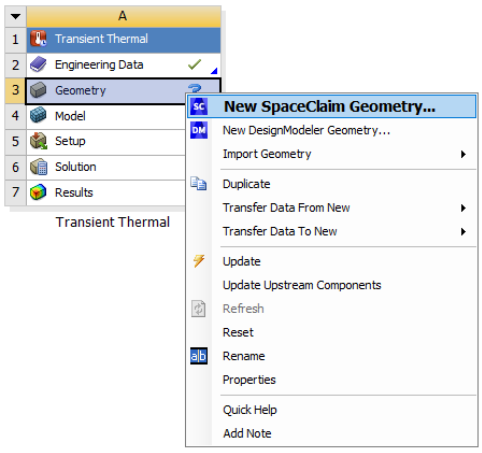


Figure 4.5: Geometry options.

The fourth step is the Model configuration, using the Mechanical environment. The simulation is organized in a project tree, and all simulation settings can be made from there. To associate the material to a geometry, the option can be executed by clicking in Material, than Insert, and Material Assignment. The options allow to associate a material in the geometry is by vertices, faces or bodies. The procedure can be see in Figure 4.6.

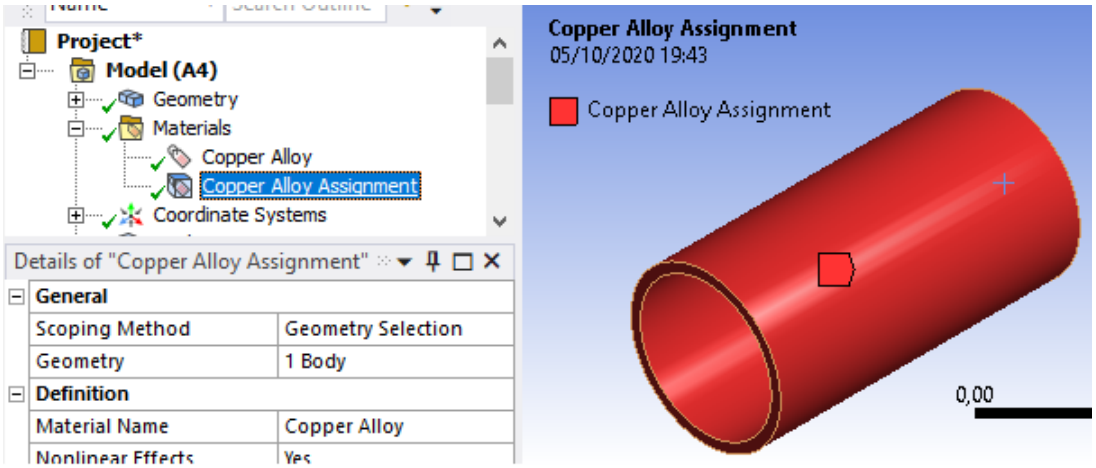


Figure 4.6: Material assignment.

After adding a Material Assignment, is necessary to associate the geometry to the material assignment.

The fifth step is create the coordinates who represents the sensors in the model. To do that, in the project tree just click in the Coordinate Systems, than Insert, and select Coordinate System, as can be seen in Figure 4.7.

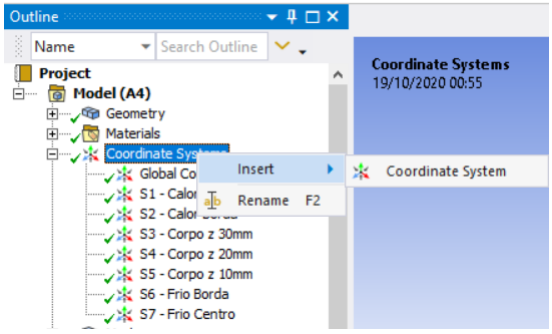


Figure 4.7: Add coordinate system.

After adding the Coordinate System, its necessary to configure the coordinates for the geometry.

The sixth step is generate the mesh, for that just right-click in Mesh, than generate. First generate with the standard configuration, than evaluate then if necessary improve the mesh adjusts add mesh controls. The mesh generate option can be seen in Figure 4.8.

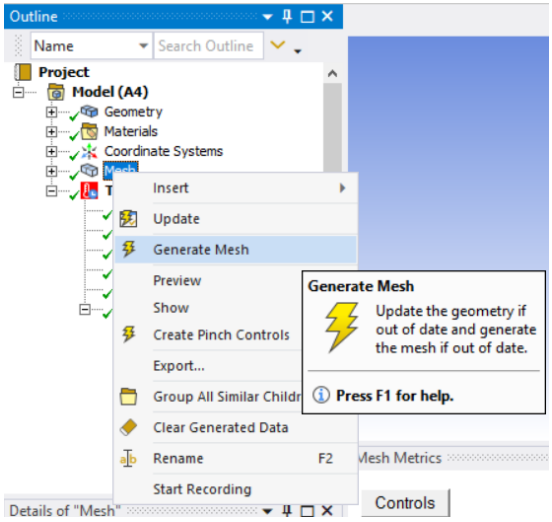


Figure 4.8: Mesh generate.

The seventh step is configure the Setup, the boundary conditions. In these option was configured the Initial Temperature, in the option Analysis Settings are configured the time of the simulation, the steps increments. To adding an analysis just right-click in Transient Thermal, than Insert, and chose a analysis.

Figure 4.9 illustrates the analysis option.

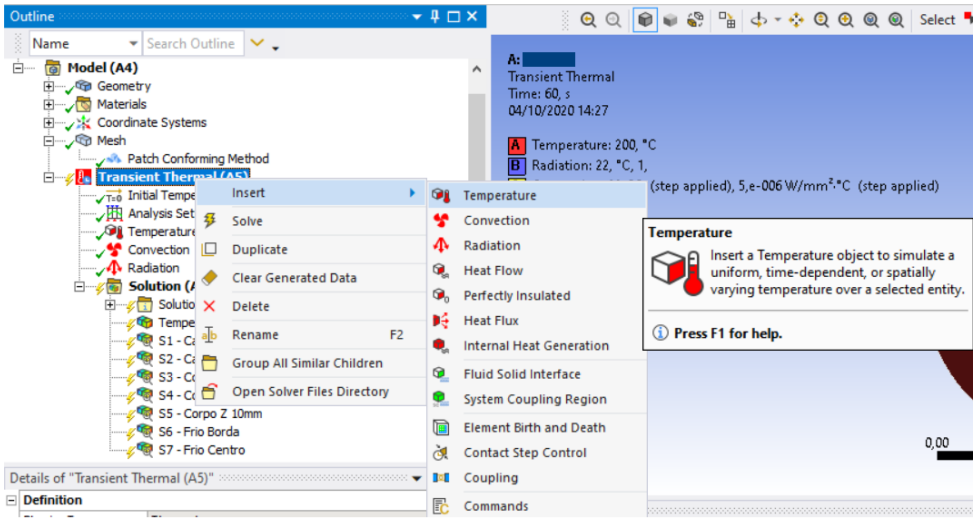


Figure 4.9: Insert analysis.

After adding an analysis is necessary to configure it. The heat components adding in the analysis can be configured to be an heat source or a heat loss. The simulation accepts more than non heat sources or losses.

The eighth step is add a solution, to do that, just right-click in Solution option, than Insert, next Thermal, and add a Temperature. The procedure can be seen in the Figure 4.10.

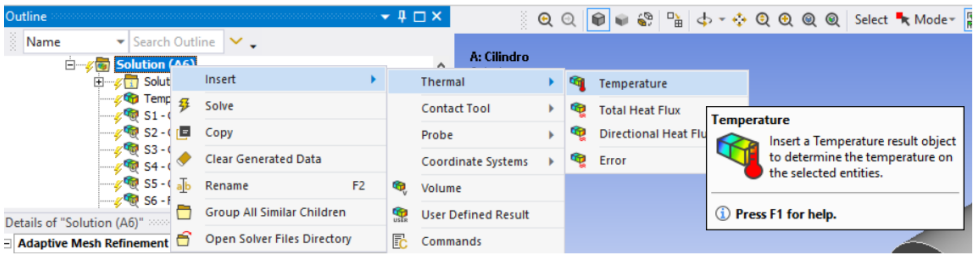


Figure 4.10: Solution configuration.

The ninth step is adding a solution to each sensor. To do that just right-click in Temperature, than Insert, next Probe, and add a Temperature. Figure 4.11 illustrates the configuration

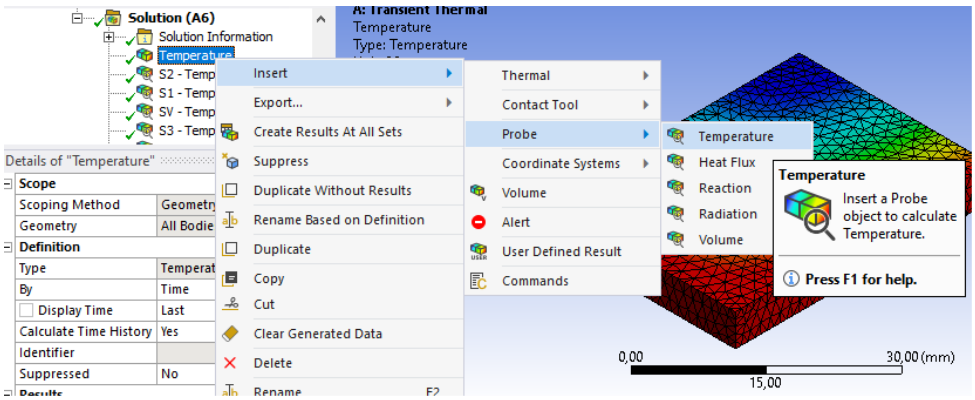


Figure 4.11: Temperature probe.

The tenth step is export the data, to do that just right-click in the results table and select the option to export. There are 2 formats to expost, in text file (.txt) and Excel file (.xls). Figure 4.12 ilustrates the precedure.

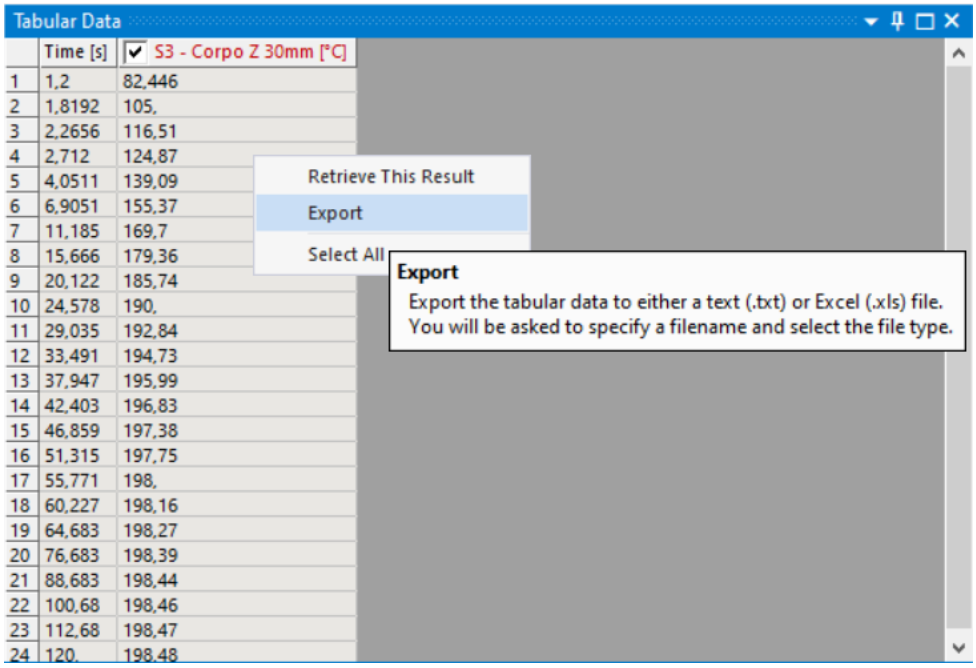


Figure 4.12: Export data.

4.2 Plate ANSYS Configuration

For the analysis of the metal plate, two simulations were developed, the first with 1 temperature gradient, and the second with 2 temperatures gradient. The configuration settings for the simulations are similar, except for the Setup option, which adds another heating region to the plate. The adding of one more heat source changes all the curves for the second simulation compare whit the first. This procedure will be duly explained in due course.

To beginning the simulation, Transient Thermal module was loaded into the Workbench. For this simulation, in the option Engineering Data it was decided to keep the default material,Steel Structure, to verify the methodology hypothesis.

The plate geometry was created through DesignModeler. At that moment it was noticed some restrictions of the program and some difficulty with simple tasks, for the other geometries, it was decided to develop them in SpaceClaim. Having overcome the initial difficulties, a solid three-dimensional piece with measures of 30 mm (length) X 40mm (width) X 5mm (thickness) was developed, as can be seen in Figure 4.13.

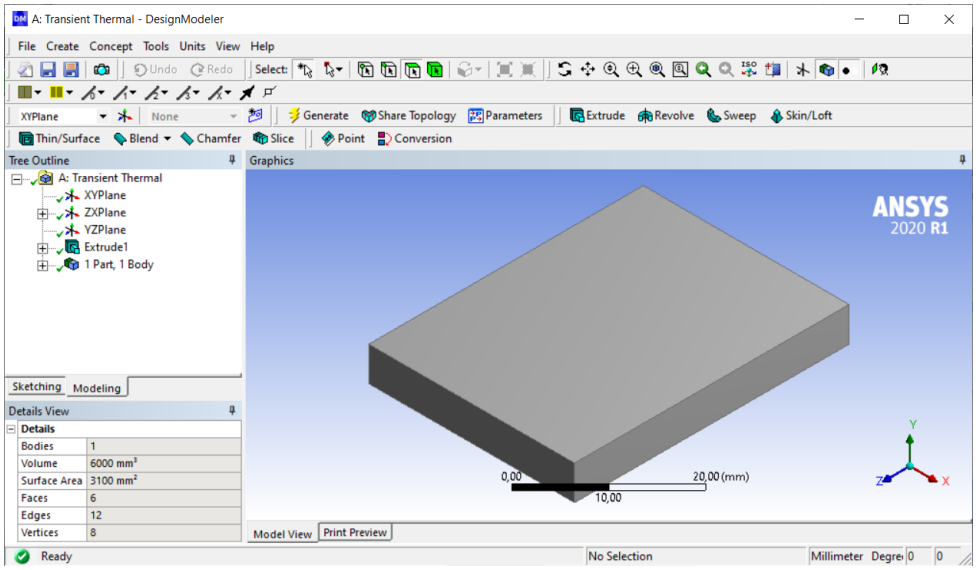


Figure 4.13: Plate geometry.

Through the Mechanical environment, the coordinates of the sensors were inserted,

for this simulation, the plate was divided into four quadrants, and in the center of the quadrant a sensor was positioned, as well in the center of the plate, totaling 5 sensors for this simulation.

The physical coordinates of the sensors on the plate can be seen according to the Figure 4.14.

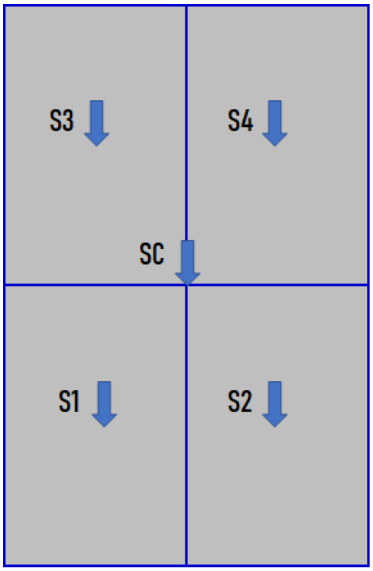


Figure 4.14: Plate sensor position.

The value of the coordinates of the sensors can be seen in Table 4.1.

Table 4.1: Plate sensors coordinates.

Sensors	**Coordinates [mm]**		
	X	Y	Z
S1	22	5	30
S2	8	5	30
S3	8	5	10
S4	22	5	10
SC	15	5	20

For the generation of the mesh, the procedure followed was first to generate the default

mesh, check the result, and then make the necessary improvements through the mesh control options.

Regarding the mesh quality parameters, the literature and forums consulted suggest the use of the graphic analysis of the *Quality* option, in this criterion, it is recommended that the more elements tend to 1, the better the mesh quality will be.

In this sense, it is important to make it clear that adjusting the quality of the mesh to acceptable levels is a task that can take time, as one must not forget another point, the computational cost, therefore, it is necessary to maintain balance.

Thus, the necessary adjustments were made so that most of the elements tended to 1, the settings made through the Details of Mesh option can be seen in the Table 4.2.

Table 4.2: Plate mesh configuration.

Sizing		**Quality**	
Option	Value	Option	Value
Use Adaptive Sizing	Yes	Smoothing	Medium
Resolution	4	Mesh Metric	Element Quality
Defeature Size	1e-003m		
Transition	Slow		

After configuring the above parameters, the quality of the mesh obtained a significant improvement but could improve a little more according to the quality criteria. Thus, a mesh control element called Method was added. The Patch Conforming Method settings can be seen in the Table 4.3.

Table 4.3: Plate mesh method configuration.

Option	Value
Geometry	1 Body
Method	Tetrahedrons

To Setup configure, the initial temperature of the plate in the simulation the value

was set at 22 °C. After, the Step Controls contained in Analysis Settings options were configured as can be seen in the Table 4.4.

Table 4.4: Plate analysis settings.

Option	Value
Number of Steps	1
Current Step Number	1
Step End Time	240 s
Define By	Time
Initial Time Step	1s

The Temperature component represents heat by conduction. To configure this component, it is necessary to select which part of the object will be exposed to heat and then configure the temperature, or the corresponding temperature curve.

For this simulation it was considered constant temperature source at 200 °C, and that the boundary conditions are ideal, that is, there are no losses for the environment. Another characteristic of this simulation is that the heat linearly propagates in the object. Figure 4.15 illustrates the heat region selected for 1 gradient.

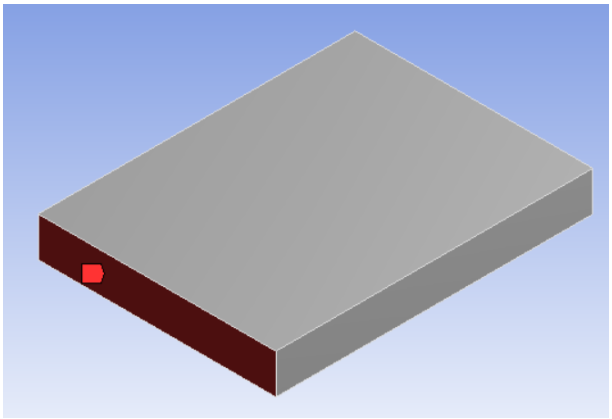


Figure 4.15: Plate with 1 temperature gradient.

The difference between a simulation for a plate with 1 gradient and another with 2 gradients, are the regions selected as a heat source.

Figure 4.16 illustrates the heating region on the plate for 2 gradients.

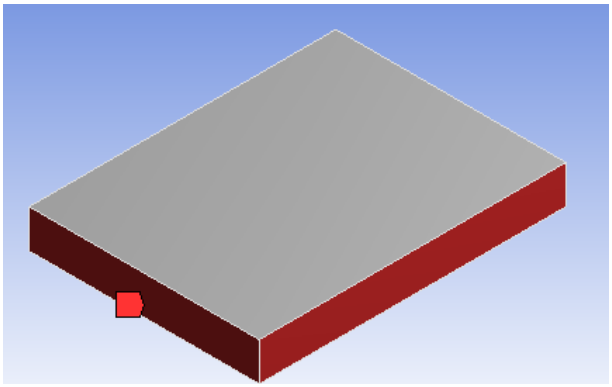


Figure 4.16: Plate with 2 temperature gradient.

After this point the Solution component was configured just like describe in section ANSYS General Configuration.

A good practice is to rename the probe for the respective sensor. After these steps, the result must be interpreted, and if it makes sense, the data can be exported.

4.3 Cylinder ANSYS Simulation

For the cylinder simulation, the material was added just like described in the section ANSYS General Configuration. Figure 4.17 illustrate the aluminium properties.

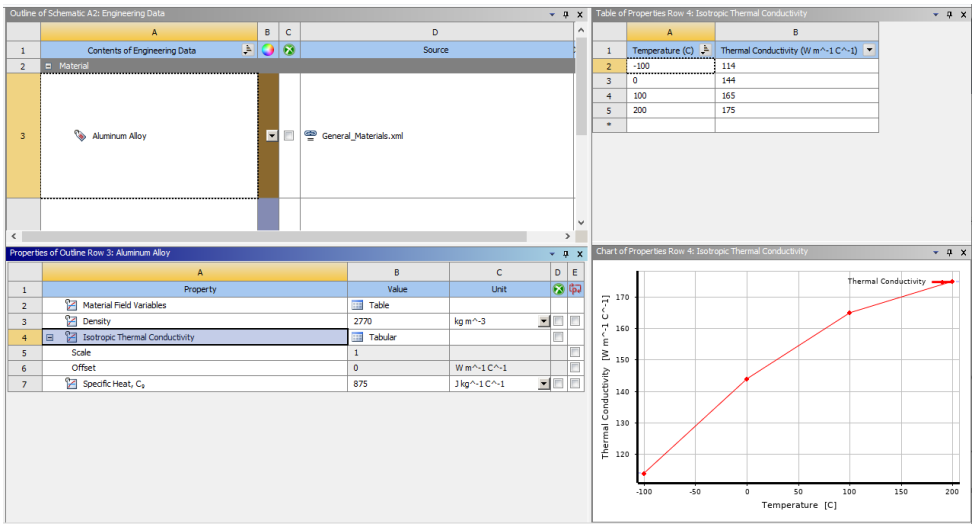


Figure 4.17: Aluminium properties.

The cylinder design was developed in the SpaceClaim CAD, as can be seen in Figure 4.18.

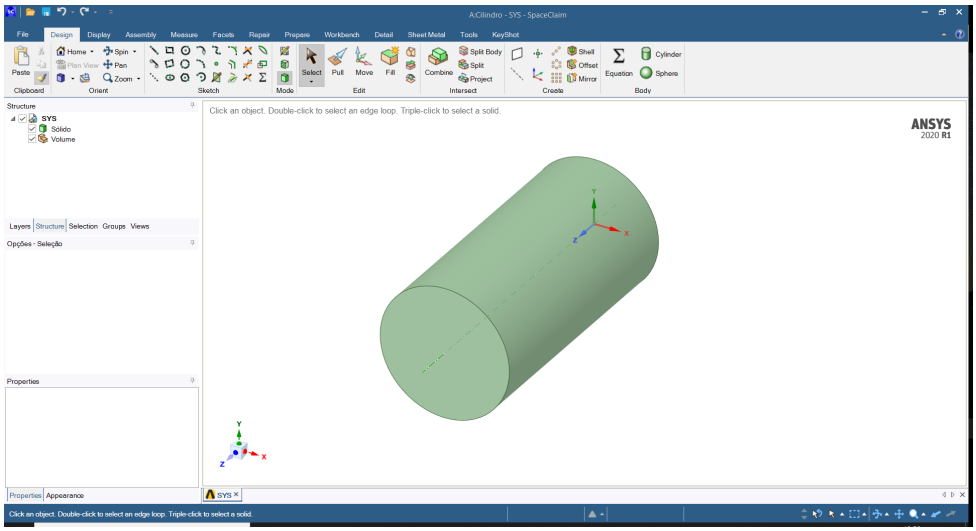


Figure 4.18: Cylinder developed in the SpaceClaim.

For the geometry design, initially, the coordinate system was adjusted to the X Y

plane, then the scale was adjusted to millimeters (mm).

For the construction of the cylinder, a circle with a radius of 10 mm was made, in sequence, the circle was extruded with 40 mm in length. The radius measurement was chosen through a catalog of materials.

In the configuration of the model, through the Mechanical console, the geometry was associated with the material, in this case aluminium alloy, just like described in the section ANSYS General Configuration.

For the positioning of the probes, 7 coordinates were created, distributed on the surface of the cylinder. The coordinates on the cylinder body are 10 mm apart from each other on the Z-axis, and the coordinates at the ends of the cylinder have been centered.

Figure 4.19 represents the distribution of the sensors along the cylinder geometry.

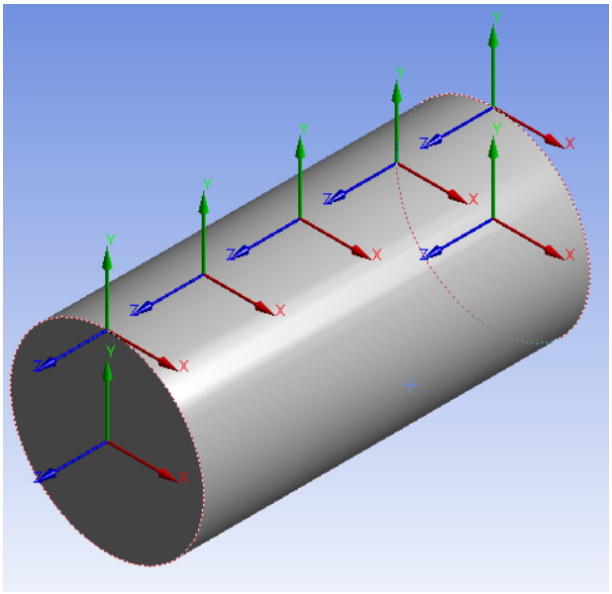


Figure 4.19: Cylinder sensor position.

As the distribution of heat in the geometry results linearly, for the experiment the positioning of the sensors along the radius of the cylinder circumference does not bring different readings. For this reason, the sensors were positioned sequentially. This phenomenon was verified by positioning the central sensors positioned at the boundaries of the geometry compared with the sensors positioned at the ends of the wall in the same

position to the Z-axis. Such observation from a simulation point of view is important to avoid redundancy in the model. Sensors 1 and 2 were evaluated to confirm the linear temperature distribution, as the value read is equal to that of the heat source, that is, continuous, these were not considered for the results.

The coordinate system positions of each sensor can be seen in the table 4.5.

Table 4.5: Cylinder sensors coordinates.

Sensors	**Coordinates [mm]**		
	X	Y	Z
S1	0	0	40
S2	0	10	40
S3	0	10	30
S4	0	10	20
S5	0	10	10
S6	0	10	0
S7	0	0	0

For the generation of the mesh, the initial procedure was to generate the standard mesh, then improve the adjustments, and finally add the mesh control, always meeting the criteria indicated through the Quality option. The Details of Mesh configuration options can be seen in the Table 4.6.

Table 4.6: Cylinder mesh configuration.

Defaults		**Sizing**		**Quality**	
Option	Value	Option	Value	Option	Value
Element Size	1 mm	Use Adaptive Sizing	Yes	Smoothing	Medium
		Resolution	4	Mesh Metric	Element Quality
		Transition	Slow		

The other settings remained at the default values. The settings used to improve the mesh control were made by adding the *Patch Conforming Method* component.

The Patch Conforming Method adjustments made can be seen in the Table 4.7.

Table 4.7: Cylinder mesh method configuration.

Option	Value
Geometry	1 Body
Method	Tetrahedrons

For the Setup configuration the Initial Temperature parameter was set to 22 °C. Then the Analysis Settings parameter was configured. The Step Controls parameters can be seen in the Table 4.8.

Table 4.8: Cylinder analysis settings.

Option	Value
Number of Steps	1
Current Step Number	1
Step End Time	60 s
Initial Time Step	0.6 s
Minimum Time Step	6e-002 s
Maximum Time Step	6 s

The values generated in Initial, Minimum, and Maximum Time Step are multiples of the time scale set to 60 s in Step End Time, these values are generated automatically by the program, by choice the values were accepted, as the analysis is not critical in the initial moments.

The cylinder setup was more elaborate compared to the plate, for this simulation, the losses due to convection and radiation to the environment were considered, the configuration of the simulation boundary conditions will be discussed below. Three components were added for thermal analysis, the Temperature component configured as a heat source,

and the Convection and Radiation components were configured to simulate the losses to the environment.

The Temperature component was configured with a magnitude of 200 °C, and the heating face of the cylinder was selected according to the Figure 4.20.

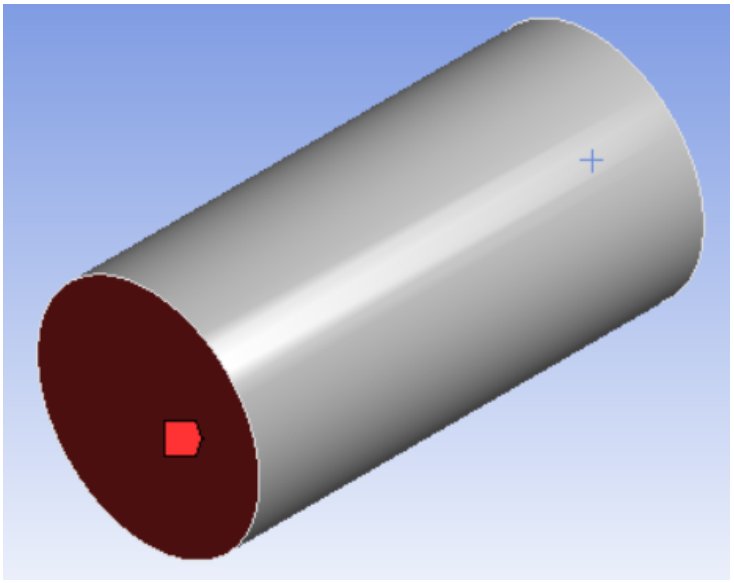


Figure 4.20: Cylinder heat selected face.

The configuration of the Convection component was performed through the Details of Convection section, the selected geometry involves the entire cylinder body except for the heating region, as can be seen on the Figure 4.21.

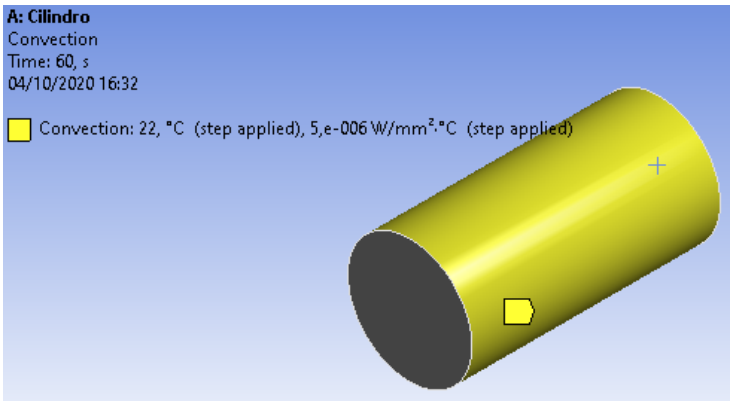


Figure 4.21: Cylinder convection loss selected face.

The Details of Convection configured parameters can be seen in the Table 4.9.

Table 4.9: Cylinder convection configuration.

Scope		**Definition**	
Option	Value	Option	Value
Geometry	2 Faces	Film Coefficient	5e-006 W/mm^2.°C
		Ambiente Temperature	22 °C

The value of the Film Coefficient is obtained through the simulator library, there is the possibility to configure this parameter manually, however, for this simulation the value provided was the one that corresponded to the scenario.

The configuration of the Radiation component was performed through the Details of Radiation, the region corresponding to radiation losses is illustrated in Figure 4.10.

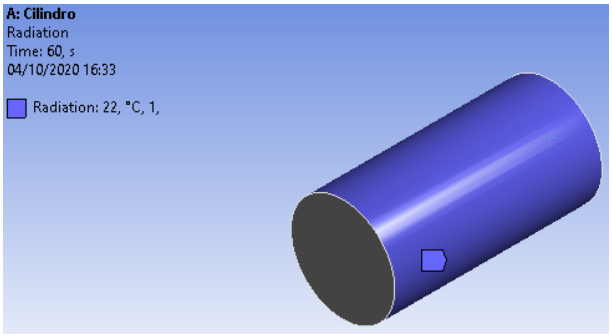


Figure 4.22: Cylinder radiation loss selected face.

The Details of Radiation parameters adjusted for the loss by radiation can be seen in the Table 4.10.

Table 4.10: Cylinder radiation configuration.

Scope		**Definition**	
Option	Value	Option	Value
Geometry	2 Faces	Correlation	To Ambient
		Ambient Temperature	22 °C

With the Setup steps configured, the solution needs to be added to the model, just like described in the section ANSYS General Configuration. Than the model must be evaluated for exports the data from the sensors.

4.4 Pipe ANSYS Simulation

For the pipe simulation, the Transient Thermal module was initially loaded into the Workbench. Then the material was selected, for this simulation a copper alloy was chosen. The material properties can be seen in the Figure 4.23.

Outline of Schematic A2: Engineering Data

	A	B	C	D	E
1	Contents of Engineering Data			Source	Description
2	Material				
3	Copper Alloy			General_Materials.xml	
*	Click here to add a new material				

Properties of Outline Row 3: Copper Alloy

	A	B	C	D	E
1	Property	Value	Unit		
2	Material Field Variables	Table			
3	Density	8300	kg m^-3		
4	Isotropic Thermal Conductivity	401	W m^-1 C^-1		
5	Specific Heat, C_p	385	J kg^-1 C^-1		

Figure 4.23: Copper alloy properties.

The geometry was built within the SpaceClaim development environment. First, the coordinate system was adjusted to the X Y plane, then two concentric circles were created, the smallest diameter with 17.5 mm and the largest 20 mm, finally the part was extruded with a length of 40 mm. The design of the part can be seen in Figure 4.24.

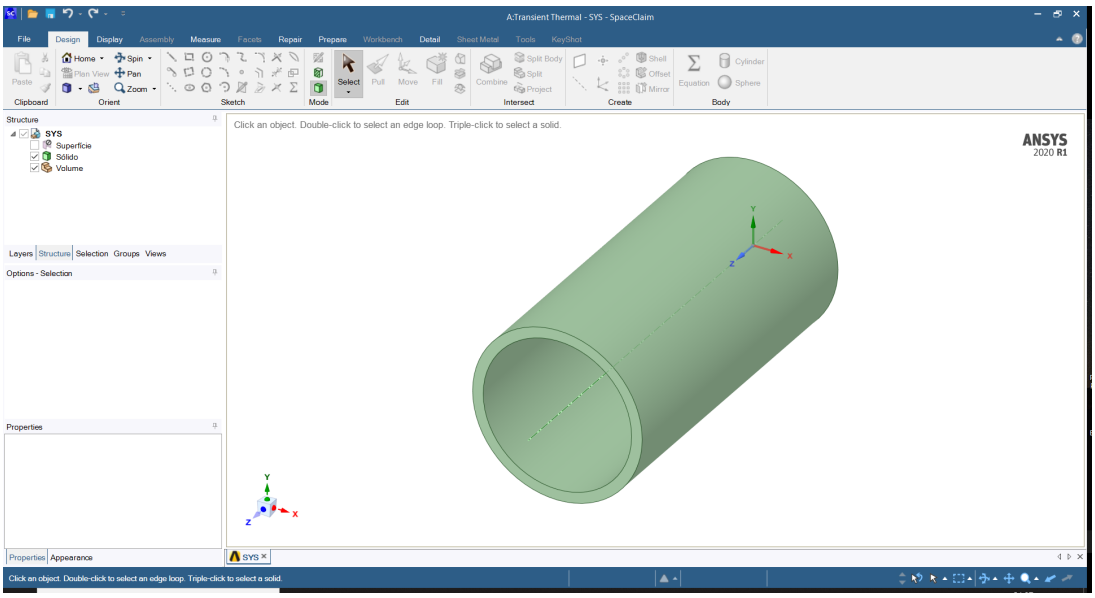


Figure 4.24: Pipe developed in the SpaceClaim.

After loading the Mechanical module, the geometry was associated with the material.

For the positioning of the probes, 5 coordinates were created. They were positioned on the outer surface of the pipe, 10 mm apart along the Z-axis. Figure 4.25 illustrates the position of the sensors.

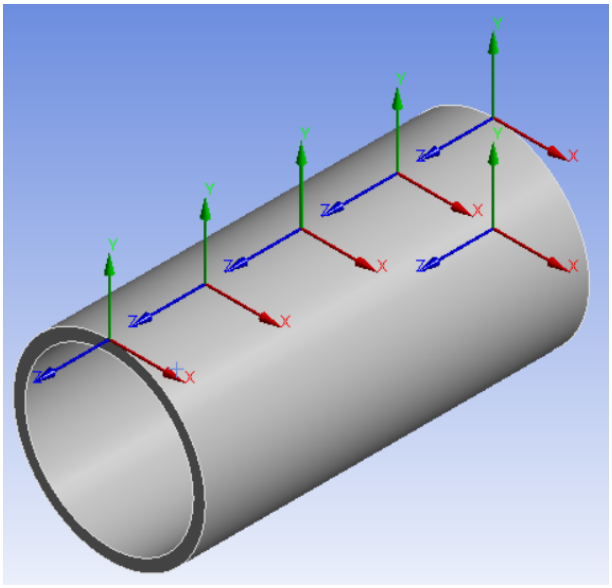


Figure 4.25: Pipe sensor position.

The coordinates of each sensor can be seen in the Table 4.11.

Table 4.11: Pipe sensors coordinates.

Sensors	**Coordinates [mm]**		
	X	Y	Z
S1	0	10	40
S2	0	10	30
S3	0	10	20
S4	0	10	10
S5	0	10	0

For the generation of the mesh, the standard procedure was followed, first to generate the default mesh, then to improve the adjustments, and finally to perform a refinement.

The Details of Mesh adjustments made can be seen in the Table 4.12.

Table 4.12: Pipe mesh configuration.

Defaults		**Sizing**		**Quality**	
Option	Value	Option	Value	Option	Value
Element Size	1 mm	Use Adaptive Sizing	Yes	Smoothing	Medium
		Defeature Size	1 mm	Mesh Metric	Element Quality
		Resolution	4		
		Transition	Slow		

As it is a pipe, and has a 2.5 mm wall, it was necessary to refine the mesh generation in the extremity regions. To improve the quality of the mesh, the Patch Conforming Method and Refinement components were added.

The module configurations can be seen in the Table 4.13.

Table 4.13: Pipe mesh method configuration.

Patch Conforming Method		**Refinement**	
Option	Value	Option	Value
Geometry	1 Body	Geometry	2 Faces
Method	Tetrahedrons		

For the refinement method, Patch Conforming Method, other parameters such as Automatic and Hex Dominant were tested in the Method option, however, for this geometry, the best performance was the option chosen as described in Table 4.13.

For the Setup the Initial Temperature of the pipe was set at 22 °C.

Next, the Step Controls parameters of the Analysis Settings option were configured according to the Table 4.14.

Table 4.14: Pipe analysis settings.

Option	Value
Number of Steps	1
Current Step Number	1
Step End Time	100 s
Initial Time Step	1 s
Minimum Time Step	0,1 s
Maximum Time Step	10 s

The configuration of the pipe Setup starts with a different arrangement from the ones adopted so far. The heat generation component is now HeatFlow. The difference of the HeatFlow component for Temperature, is that the configuration of the HeatFlow component is the heat transfer rate in watts (W), whereas the Temperature component the parameter is the temperature in °C.

For Heatflow and Temperature components, it is possible to associate an equation that describes the phenomenon, or even describe the behavior in tabular data form as a function of the step.

For the HeatFlow component, the Magnitude was set to 100 W, and the selected heating region can be seen in Figure 4.26.

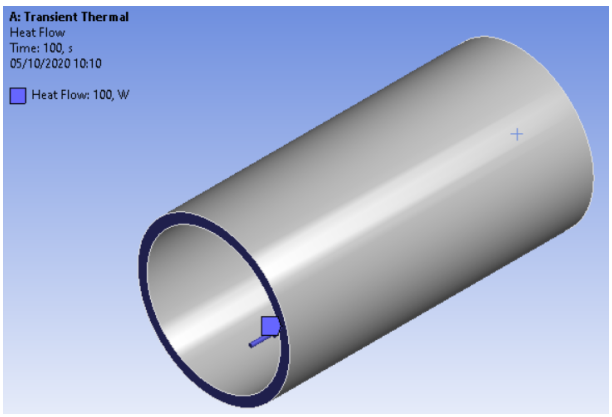


Figure 4.26: Pipe heat selected face.

The configurations of losses by convection and radiation are similar to those used in the cylinder, except for the different geometry. The convection loss settings for this geometry also comprise the inner surface of the cylinder. As a result, a larger area influenced by losses is expected.

Figure 4.27 illustrates the regions selected to simulate convection losses.

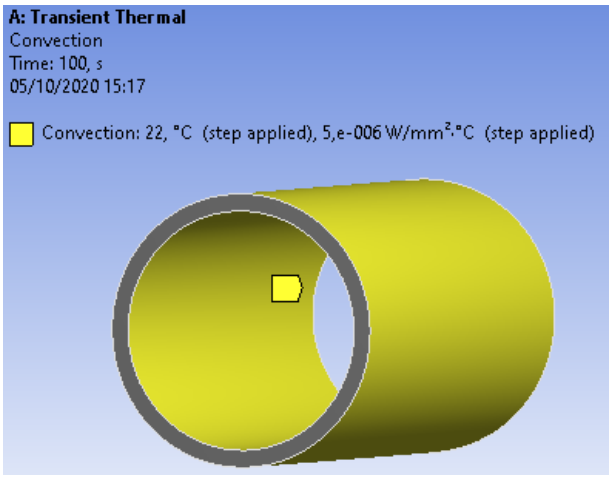


Figure 4.27: Pipe convection loss selected faces.

To obtain the convection loss, the Film Coefficient is a value from the ANSYS data base. The other parameters that are part of the component have been kept at their default values.

The Details of Convection parameters configured to simulate the convection losses can be seen in the Table 4.15.

Table 4.15: Pipe convection configuration.

Scope		**Definition**	
Option	Value	Option	Value
Geometry	3 Faces	Film Coefficient	5e-006 W/mm^2.°C
		Ambient Temperature	22 °C

For the configuration of radiation losses, the selected geometry can be seen through

Figure 4.28.

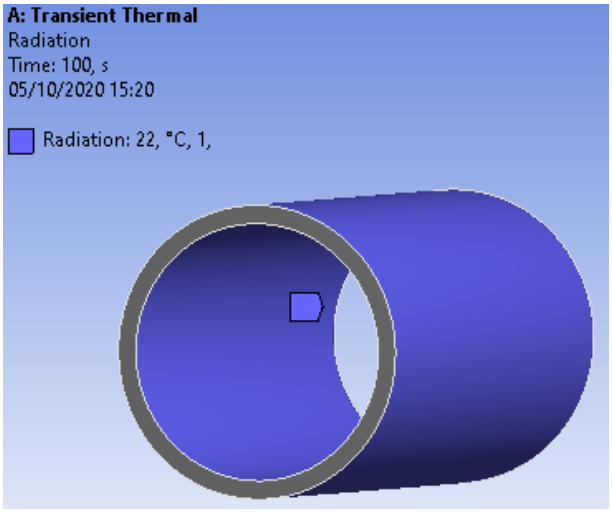


Figure 4.28: Pipe radiation loss selected face.

To simulate radiation losses, the Details of Radiation parameters was defined according to the Table4.16.

Table 4.16: Pipe radiation configuration.

Scope		**Definition**	
Option	Value	Option	Value
Geometry	2 Faces	Correlation	To Ambient
		Ambient Temperature	22 °C

With the parameters of the Setup step configured, it was necessary to configure the Solution component, just like explained in the section ANSYS General Configuration, than evaluated the results and export the sensors data.

4.5 Mold ANSYS Simulation

All the case studies described above, served to learn, understand and collect information about specific characteristics of each simulation. The geometries in different formats, as well as the materials, served to understand which configurations would be more assertive to apply the technique to the injection mold.

The injection mold file was made available in .STP format, and with all the parts and components that serve for its operation, as can be seen in the Figure 4.29.

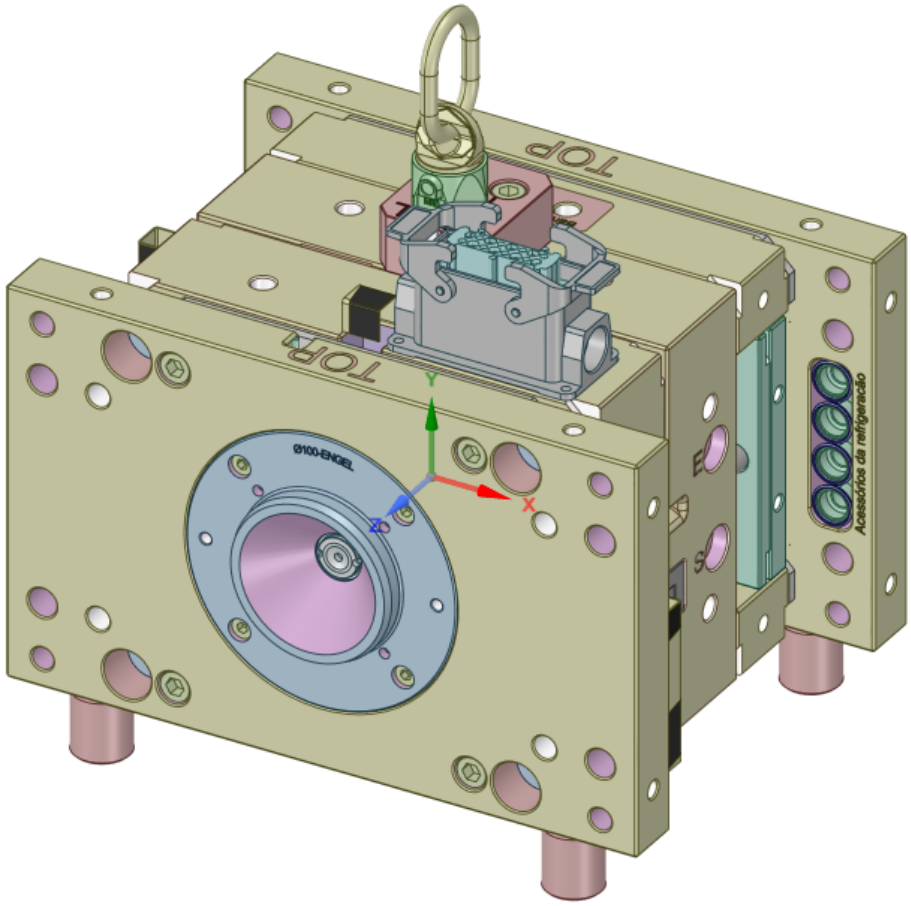


Figure 4.29: Complet mold geometry.

However, for simulation purposes, it is impossible to simulate the original file due to the quantity and complexity of the elements involved. To solve this limitation, the simulation literature says that the geometry to be analyzed needs to contain the main

elements that will be observed in the simulation, but in a simplified way.

In this sense, there is no arbitrary rule, but a critical sense of who is performing the simulation of perceiving which elements interfere, and therefore need to be maintained, and which elements do not interfere, or interfere very little, and thus become dispensable for the simulation result.

The first step in analyzing the mold was to simplify the geometry, keeping the main elements, and discarding the elements that don't have relevance for simulation. The first attempt at simplification was carried out using the Solidworks software, however, when opening the file inside the program, the project tree was very messed up, and it was not possible to easily identify the elements of the mold.

Due to the difficulty encountered with the Solidworks software, as an alternative, SpaceClaim was used, which surprised in this sense, having opened the file tree in an organized way. The project tree can be seen in the Figure 4.30.

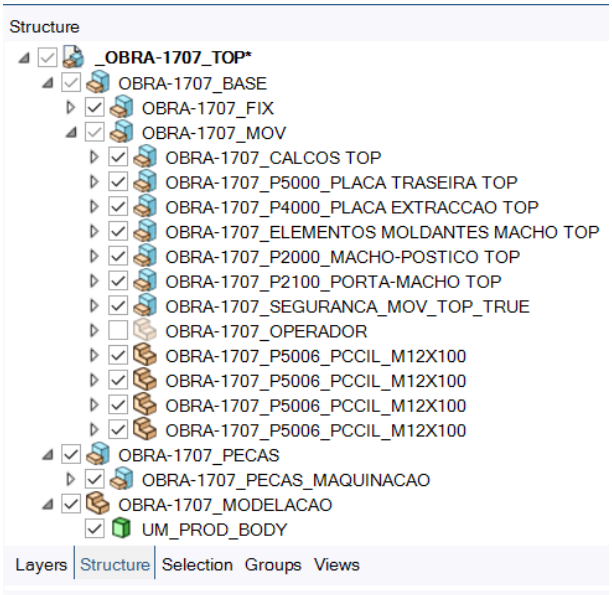


Figure 4.30: SpaceClaim project tree.

This facilitated the identification of the parts of the mold, as well as the subsequent edition for the simplification of the geometry.

Simplifying the mold geometry was a difficult task that took considerable time to

reach the desired result. Initially, it was defined that the object of study is the injection chamber, composed of the core and the insert. Separating the core and the insert from the rest of the parts was a cautious task, which required attention and care.

After separating the core and the insert from the rest of the mold parts, it was necessary to simplify the referred parts, since they contained screws, extraction guides, sealing rings, among other elements that were not important for the analysis. After removing these elements, the next step was to fill unnecessary holes, all the holes where screw guides and other elements would have previously been closed.

External finishing elements were also removed from the piece, such as rounded corners and chamfers. It is important to make it clear that the internal geometry of the injection chamber has been kept intact, as in this case, it represents the actual physical parameter, and cannot be modified. The modifications carried out, such as the elimination of rounded corners, the elimination of holes and chamfers, do not interfere in the dynamics of the analysis, as they were made to preserve the original characteristics and provide a more simplified mesh for the analysis. Figure 4.31 illustrates the core and the insert, still in the process of simplification.

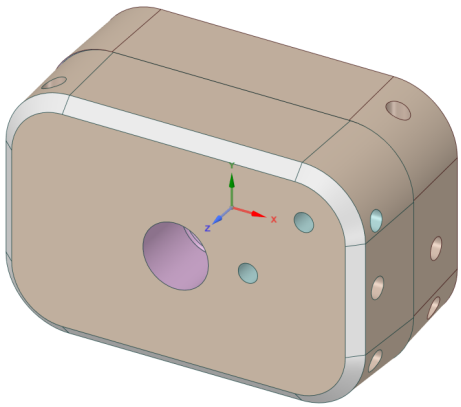


Figure 4.31: Injection chamber in simplification process.

With the part carefully simplified, leaving only the elements of interest for analysis, such as the injection chamber and the cooling ducts, the simulation can proceed to the next phase.

At this time, it was necessary to add the simulation material, but the ANSYS library does not include the mold metal. Therefore, the metal was added manually, like described in the section ANSYS General Configuration. The mold was made with steel 2738, the parameters were extracted from the manufacturers' catalogs. For that simulation the polypropylene (PP) was add to represent the plastic heat domain. Figure 4.32 illustrates the material properties.

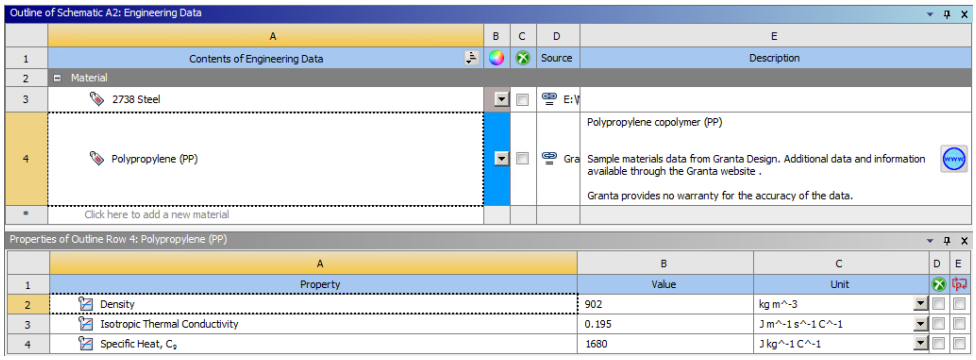


Figure 4.32: Mold materials.

The parameters configured in the steel was that who is needed to represent the thermal behavior. For other type of analysis, other parameters need to be adding.

For Setup configurations, the first parameter to be configured is to associate the material with geometry. In that case, the steel was associated with the mold's core and the cavity, and the PP was associated with the volume injection. To facilitate the understanding, the Figure 4.33 will show the plastic domain and the mold core.

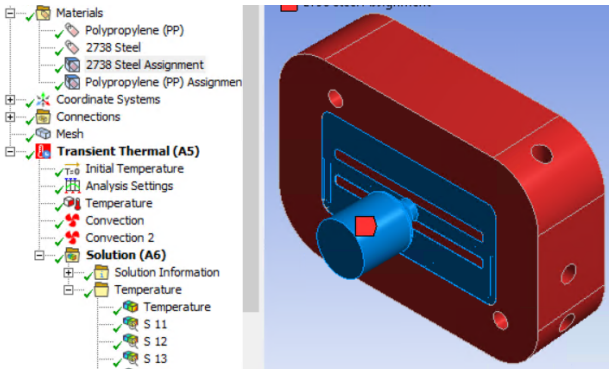


Figure 4.33: Mold materials assignment.

After associating the material with the geometry, it is time to determine the position of the simulation sensors. As the geometry of the mold has complex surfaces, with ridges and recesses, the techniques of positioning the probes learned previously, played a role of great importance. For the mold, 8 sensors were defined in locations with different geometry bounces, the position of the sensors can be seen in Figure 4.34.

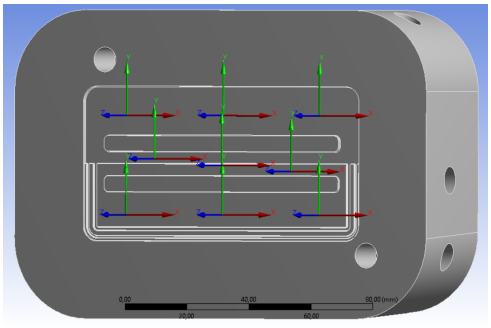


Figure 4.34: Mold sensor position.

The coordinates of the sensors were organized as the positions of a matrix, so the names of the sensors represent the respective position, except for the sensor that corresponds to the position S22 because at this point there is a hole in the geometry. The positions of the sensors can be seen in Table 4.17.

Table 4.17: Mold sensors coordinates.

Sensors	**Coordinates [mm]**		
	X	Y	Z
S11	-35	16	-1
S12	0	16	-1
S13	35	16	-1
S21	-25	2	1
S23	25	-2	-0,5
S31	-35	-16	-0,5
S32	0	-16	-0,5
S33	35	-16	-0,5

The generation of the mold mesh was a step that required a lot of work. Because it is complex geometry, with two bodies, a volume domain, many curves, cylindrical elements, small edges, chamfers, and ledges, it was necessary to test various configurations until a representative mesh was obtained at a viable computational cost.

Sometimes, during the mesh generation, the computational resource demanded was so great that the simulator's notification interface recommended reducing the mesh refinement, since there was no more memory available in the system. The recommendation was reported during development on both the laptop and the virtual machine created to run the simulation.

After many tries the mesh configurations was made with success. The Details of Mesh configuration can be seen in the Table 4.18.

Table 4.18: Mold mesh configuration.

Defaults		**Sizing**		**Quality**	
Option	Value	Option	Value	Option	Value
Element Size	3 mm	Use Adaptive Sizing	Yes	Smoothing	Medium
		Resolution	3	Mesh Metric	Element Quality
		Transition	Slow		

For the mold, it was decided not to use any mesh refinement method, as the tested methods did not support much difference and added a considerable computational cost. As the values found are within the quality metrics, it was considered that the settings were sufficient.

In the Setup configuration the Initial Temperature was set in 30 °C. The initial temperature was set at this value according to data provided by consulted bibliography.

The next step was to configure the Analysis Settings option, here is important to pay attention to representing the times involved in the injection cycle. The data provided by the Mold engineering inform that the injection cycle for the test piece, lasts around 30 s to 50 s. For the simulations settings, the fast case was assume, 30 s. The Step Controls

in Analysis Settings can be seen in Table 4.19.

Table 4.19: Mold analysis settings.

Option	Value
Number of Steps	30
Current Step Number	1
Step End Time	1 s
Auto Time Step	Program Controlled
Initial Time Step	1e-002 s
Minimum Time Step	1e-003 s
Maximum Time Step	0,1 s

For the configuration of the heat source, the temperature component was used. The heat source temperature was adjusted according to the test mold injection cycle, as can be seen in Table 4.20.

Table 4.20: Mold injection parameters.

Option	Value
Mold closed	30 °C
Polymer injection	240 °C
Polymer cooling with closed mold	240 °C to 30 °C
Mold opening	30 °C
part extraction	30 °C

Before configure the heat source and losses parameters, is necessary to understand the heat exchange inside a mold. For this case, the heat exchange inside the mold is forced, through an internal water cooling circuit. The injection chamber is positioned inside a metal set in such a way that there is no contact with air during the injection, heating, and cooling operation, only in the extraction stage, as one of the halves (core) moves. By

making a side cut of the injection chamber to identify which heat exchange phenomena are present, heat exchange by contact and convection would be identified. Figure 4.35 illustrates a representation of the layers seen in a side section of the mold.

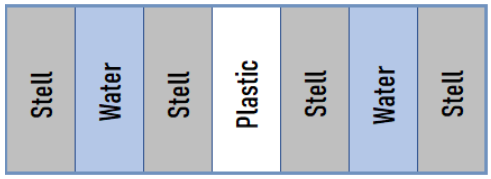


Figure 4.35: Mold cross section.

The melted plastic transfers heat to the mold wall by convection and conduction, considering that the flow is laminar and the flow speed on the sides tends to zero. The metal walls transfer heat by conduction to the walls of the refrigeration tubes, where the heat is exchanged by convection.

To simulate the heat source the component Temperature was used. The temperature value was tabulated according to the information provided through the Table 4.20, beginning at 240 °C and ending at 30 °C. The heat inside the mold comes from the melted polymer, thus, a representation of the phenomenon for a thermal analysis is made through the volume occupied by the polymer.

The heat area can be seen in Figure 4.36.

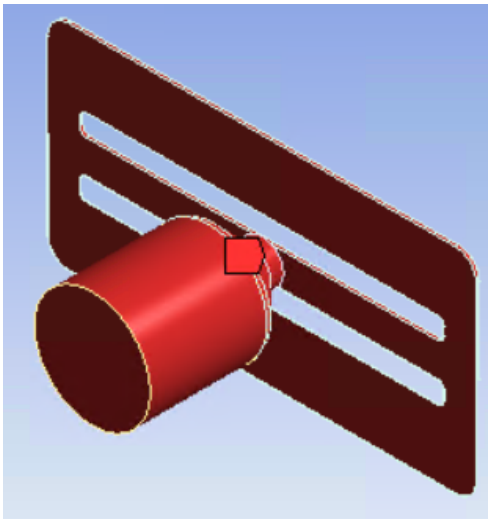


Figure 4.36: Mold heat regions.

For the losses the component Convection was used, the water channels was the section area and heat exchange in the mold. This configuration involved a lot of work because the sections of the refrigeration tubes are developed by parts. So selecting all the parts requires a lot of attention since the work had to be done independently for the cavity and the core. Figure 4.37 show the convection areas selected.

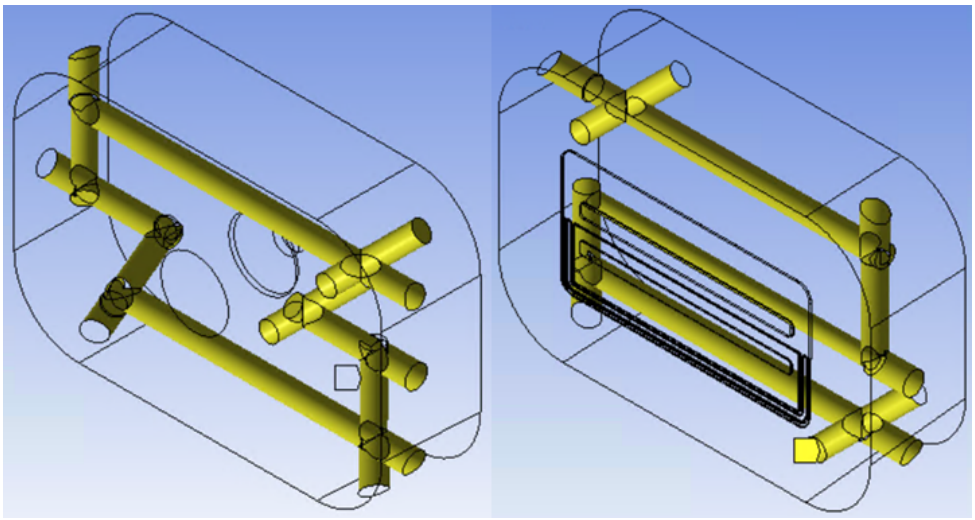


Figure 4.37: Mold convection regions.

Table 4.21 shows the Details of Convection parameters for the mold core.

Table 4.21: Mold core convection configuration.

Scope		**Definition**	
Option	Value	Option	Value
Geometry	15 Faces	Film Coefficient	5e-006 W/mm^2.°C
		Ambient Temperature	30 °C

Table 4.22 shows the Details of Convection parameters for the mold cavity.

Table 4.22: Mold cavity convection configuration.

Scope		**Definition**	
Option	Value	Option	Value
Geometry	28 Faces	Film Coefficient	5e-006 $W/mm^2.°C$
		Ambient Temperature	30 °C

To configure the solution, it is necessary to add the Temperature component as seen in the section ANSYS General Configurations.

One of the main difficulties encountered up to this point is the fact that the simulation takes more than 4 hours to process. In fact, this makes it difficult to refine the adjustments because the time interval for checking the results is very long.

After many attempts and different configurations, the closest result produced from past requirements was found, for transient thermal analysis. At this point the results of each sensor can be export, to start the fitting curves process and find the mathematical expressions that represents the temperature behavior in the selected points.

4.6 Fitting Curves

This section describes the step by step how to obtain the data exported by ANSYS simulation, the pre-treatment with the aid of Microsoft Excel, and the equations obtained through the Matlab software.

As it is a repetitive procedure in all study cases, it will be approached as a single topic. The variations produced in each analysis refer to the number of equations produced, according to the number of sensors added to the study geometry. What changes from one analysis to another is the amount of data that needs to be processed to produce the necessary sensors.

The first step to implement this analysis is to organize the information exported by ANSYS. The data needs to be organized and tabulated because during the export process, due to some incompatibility with the current version of Microsoft Office, many data lose the decimal point, and in others, the digit is changed in position. Figure 4.38 shows an example of the situation explained.

B	C	D	E	F
	Temperature °C			
Time [s]	S3 Z 30mm	S4 Z 20mm	S5 Z 10mm	S6 Z 0mm
0.600	60.038	29.537	23.509	22,56
0.909	78.830	35.384	24.897	23112.00
11.259	89.888	40.186	26.223	23675.00
13.429	98.788	45.328	27.909	24456.00
19.347	114.130	58.544	34.390	28,42
2.930	129.180	76.277	47.370	38751.00
48.237	144.860	99.785	71.459	61928.00
84.328	161.320	128.530	106.760	99137.00
12.876	173.810	151.530	136.550	131,22
17.351	182.070	166.900	156.700	153,03
21.827	187.560	177.130	170.130	167,59

Figure 4.38: Disordered imported data.

Regardless of the decimal and thousands separator settings used in Microsoft Excel, none fixed the reported problem. Thus, the data was manually adjusted. With the

tabulated and organized data, it was possible to obtain a preview of the polynomial and logarithmic curves using the Excel graphics wizard, which also provides the characteristic curve and the determination coefficient, as can be seen in the Figure 4.39.

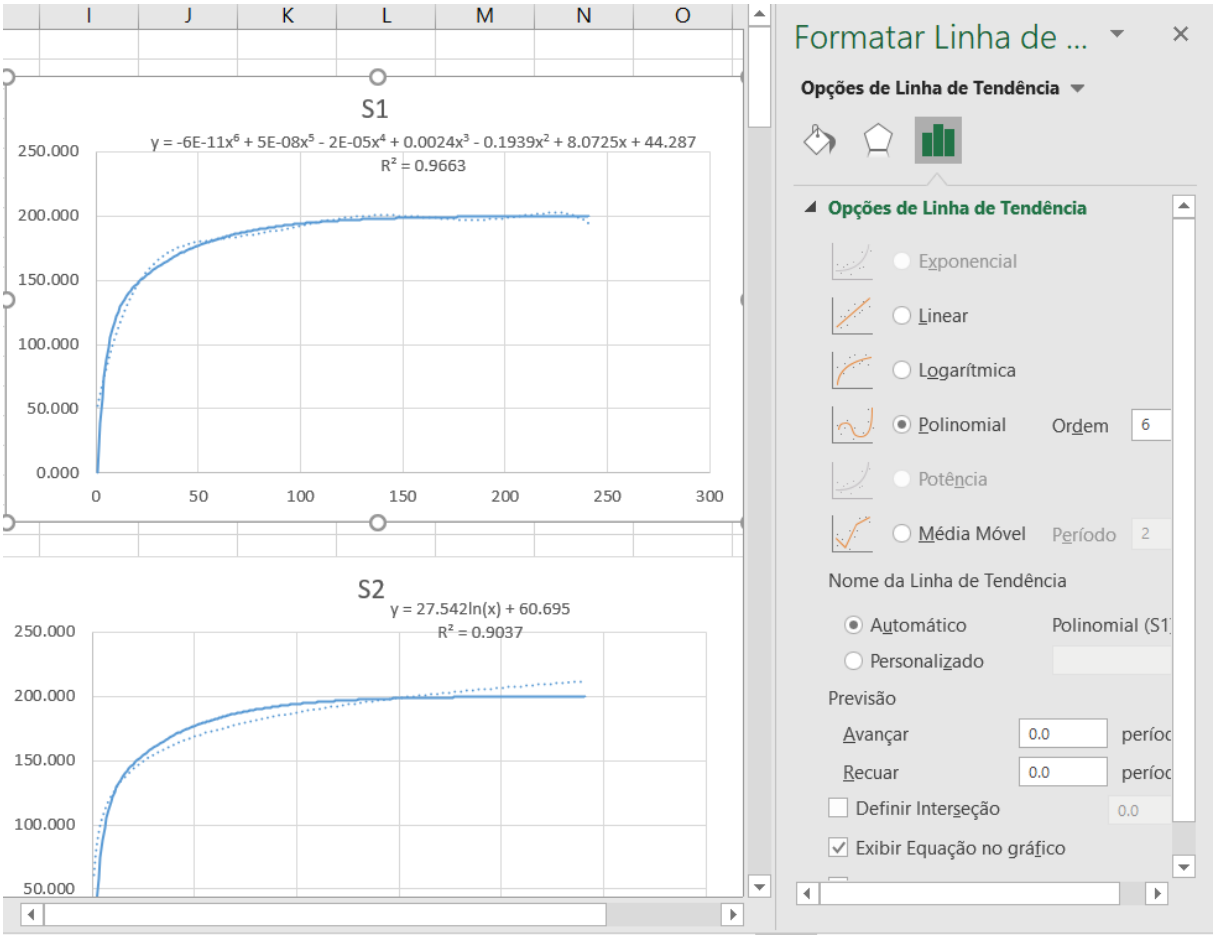


Figure 4.39: Microsoft Excel fitting example.

The file organized in Excel needs to follow some parameters so that it can be imported correctly by Matlab. To save time, it was necessary to group the values in columns. To facilitate organization, the first column is the time column and the subsequent ones are the temperature values of the sensors grouped in ascending order. Another feature is that a spreadsheet needs to be saved as tab-separated text to be imported in Matlab.

Matlab is a software with many resources, for this work it was very important to find and adjust the equations of the points of each sensor. Each curve represents the unique

behavior of the heat interaction at the configured point. To perform this procedure the first step is to import the table with data organized in Excel. This procedure is done by clicking on the *Import Data* button located on the *Home* tab, according to Figure 4.40.

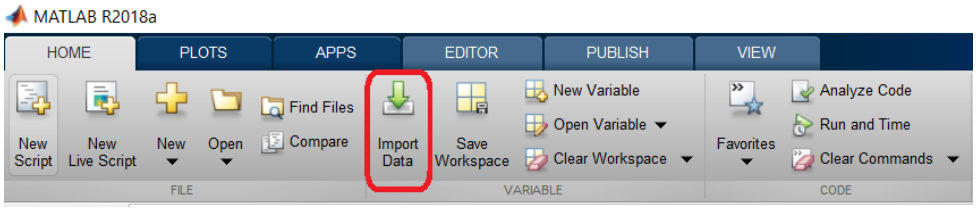


Figure 4.40: Matlab import data button.

Then, the import utility opens automatically and the *Output Type* parameter needs to be configured as column vectors, and then press the *Import Selection* button. The operation can be seen in Figure 4.41.

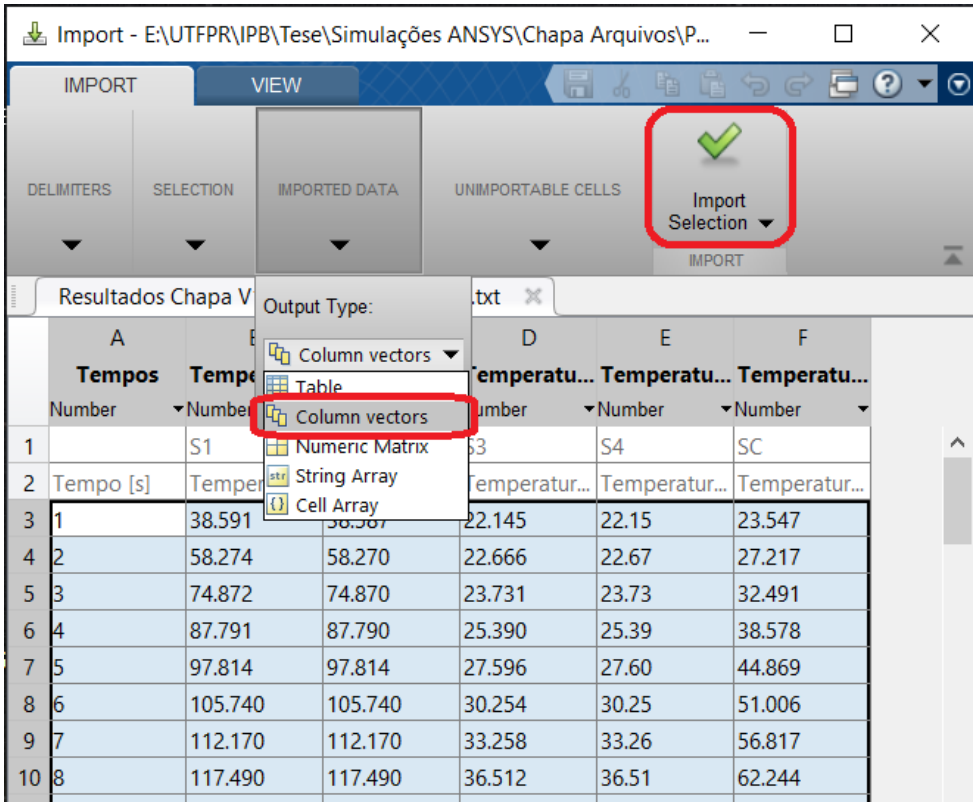


Figure 4.41: Matlab import utility.

It is important to check if the data was actually loaded in the Matlab environment,

this can be identified by checking the *Wokspace*, which is the field where all the variables used in the program are presented. In the *Workspace*, all values related to temperature and time must be present. By double-clicking on the variable it is possible to see its contents. Workspace can be seen in the Figure 4.42 .

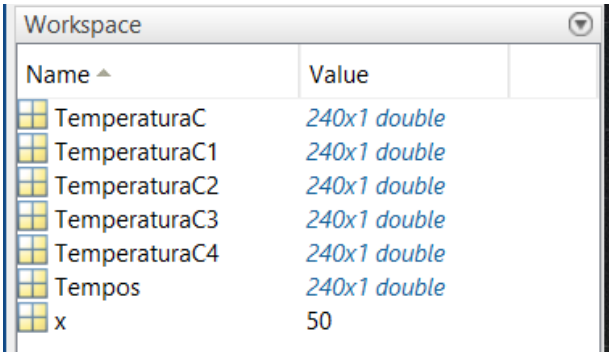


Figure 4.42: Workspace example.

To open the *Curving Fitting Tool*, just type the command *cftool* in the *Command Window* and wait for the utility to load. The steps taken to find an equation that represents a curve are illustrated in Figure 4.43 and the procedure is explained below.

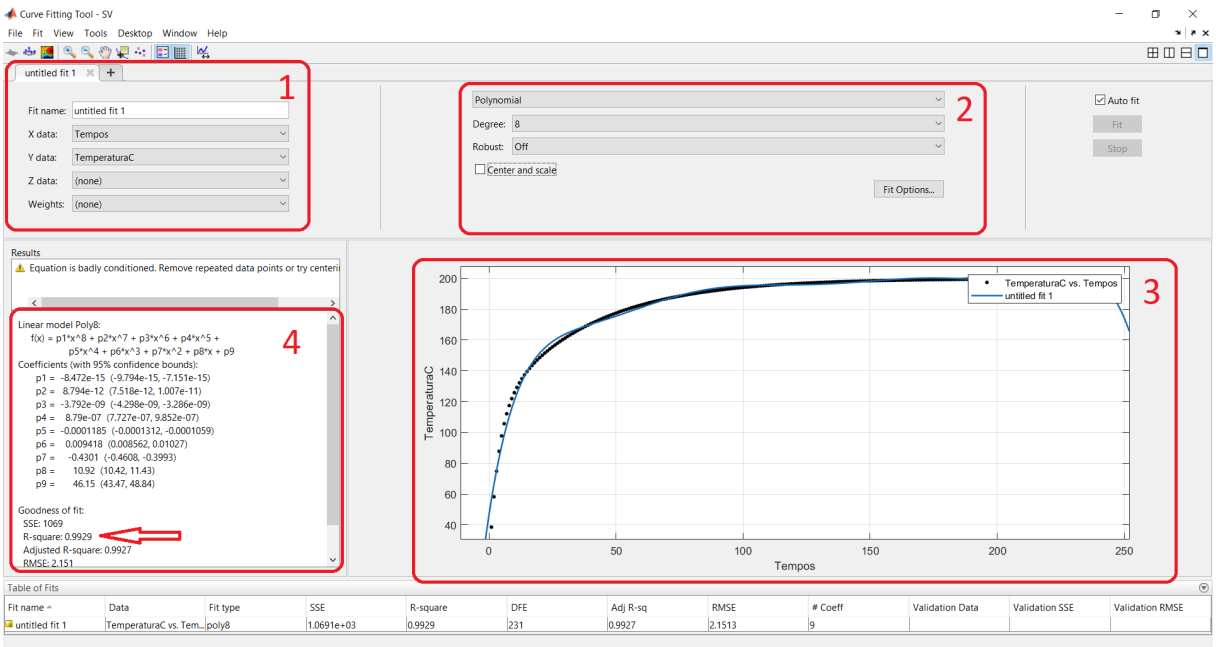


Figure 4.43: Curve Fitting Toll steps.

With the utility open, the first step is to associate the variables with the coordinate system, according to the selected area 1 shown in the Figure, for this study the time variable is on the X axis and the temperature variables are on the Y axis. The second step is to choose the type of equation to be associated with the curve, according to the area 2 selected, for this analysis the polynomial equations were chosen. The result of the approximation of the curve can be seen graphically in the selected area 3, this visual analysis helps in choosing the degree of the polynomial. The resulting equation, as well as the coefficients, can be seen in the selected area 4. It is also possible to observe the value of R^2, which increases or decreases according to the chosen equation.

Since the equations are a function of time, to find the temperature value at a given time, just replace that value in the equation variable. In that point the equations are ready to be tested in a test code.

4.7 Simulation Code for Performance Tests Implementation

For the tests performed, the computer performance used interferes directly with the code performance. The configuration of the computer used in the tests have a Intel Core I5 2450M 2,5 GHz processor, 250 GB SSD storage, 6GB of RAM memory, a Win 10 Home Edition, and Google Chrome version 86.0.4240.65 web browser. That information is relevant from the point of view of experiment reproduction.

A Python code was developed in Jupyter Notebook using the Anaconda Navigator program, the choice was made due to the versatility and organization of the files. The choice of language is because the other parts of the project were developed in Python.

The simulation developed in Python consists of a code to check the reliability and the execution time of each equation found in the models. As the acquisition rate of the physical sensor is approximately 130 ms, one of the objectives of the simulation is to verify which equation meets this requirement. Other requirement is the injection cycle, which lasts between 30 s and 50 s, and depends on the configuration of the injection machine. For the simulations 30 s was considered.

The Python code is a generalist code and was developed to test all simulated case studies, what changes in the implementation of one case study to another are the equations found through the curve fitting process.

The code was developed to be as fast as possible and interfere as little as possible in the overall performance. Thus, the code uses only the *sympy* library and the *time* library.

In this version of the software, the input parameter is simulated and can be entered manually or operated incrementally through a repetition loop.

The implementation make uses the time library to measure code execution time. The time execution is triggered at the beginning of the code and is measured at two different times, during the execution of the inverse function, and at the end of the code, providing the total execution time.

Particularly in this measurement, the time of execution of the inverse function is of

great interest, since it is the one that determines the time for the calculation of the virtual sensors. The analysis of the total time reveals characteristics of the implementation of each equation, it is possible to perceive the speed of resolution through the difference between the final time and the time of the inverse function.

For the final results, statistical measurements were made regarding the execution time, the precision of the time returned by the inverse function, and precision measurements of the equations in inferring the value of each sensor.

The Python language has some interesting features such as handling variable types. The type of variable is automatically recognized, so there is no need to deal with this situation. In this sense, the code takes advantage of this technology with a feature implemented through the function *Symbol*. An symbol was configured to return only real values.

To find the resolution of an inverse function, for example in a polynomial of degree 8 and degree 6, there are complex roots, which do not matter for the result. Thus, through the use of the *Symbol* function, the imaginary value is ignored because the argument of the *Symbol* function was defined to return the real value of the function. This result is desired because having fewer results to evaluate means less time to choose the roots.

The function *Symbol* is required for type conversion when using the function *solve*. The resolution of the inverse function was performed with the aid of the *solve* function. The function *solve* isolates the variable to be operated and returns the value found from the roots. The real roots are stored in a float vector.

For a higher degree polynomial function, it is important to know which real root is the one that represents the valid time instant. For the tested grade 8 and 6 polynomials, two real roots were found.

Filters were created to choose which of the resulting roots is most suitable for the code in question. The negative time is invalid, and time greater than the total value of the simulation does not represent the process. So only the significant value will be selected.

At this point, the analysis is the same for topologies 1 for n and n for n. The big difference is that for topology 1 for n there is only one inverse function, while in topology

n for n there are n inverse functions.

For topology 1 for n, with the result of the time obtained through the inverse function, it is already possible to infer the temperature value at the determined points. An example of the code with 1 real input sensor and multiple virtual output sensors can be seen in Figure 4.44.

```
#Sensor Virtual com 1 Gradiente de Temperatura, com Polinômio de Grau 8, 1 Sensor Entrada, 4 SV, Tempo
#Versão 1.0
from sympy import *
x=Symbol('x', real=True)

#Mede o tempo de execução
import time
start_time = time.time()

#Simula imput de temperatura do sensor real
T=184.980



print('Temperatura Real:',T,'ºC')
print('-------------------------------------\n')

#Encontra o tempo através da equação do sensor Real
v=solve((2.001e-10)*x**8 + (-5.145e-08)*x**7 + (5.453e-06)*x**6 + (-0.0003063)*x**5 + (0.009708)*x**4 +
print('Raizes:',v)
if(v[0]>0):
    x=v[0]
if(v[1]>0):
    x=v[1]

print('Raiz selecionada:',x)
print('-------------------------------------\n')

#Informa o tempo usado para encontrar a raiz (função inversa, encontra o tempo em segundos do sistema)
print("---Tempo Raiz --- %s segundos ---" % (time.time() - start_time))
print('-------------------------------------\n')
#Calcula os sensores Virtuais a partir do tempo encontrado na equação do sensor Real
S1=(-7.469e-10)*x**8 + (1.943e-07)*x**7 + (-2.094e-05)*x**6 + (0.001208)*x**5 + (-0.04024)*x**4 + (0.7
print('Sensor virtual 1:',S1)
S2=(-1.536e-10)*x**8 + (4.142e-08)*x**7 + (-4.67e-06)*x**6 + (0.0002858)*x**5 + (-0.01034)*x**4 + (0.2
print('Sensor virtual 2:', S2)
```

Figure 4.44: Python code 1 input real to n output virtual.

For the situation of multiple inputs, it is necessary to initially solve all inverse functions. The results of all inverse function must be the same time instant, or very close. After this step the algorithm calculates the average of the times. This time is the input parameter to infer the temperature values. Such approach has limitations, by the computational cost and time instant roots perspectives.

Chapter 5

Results

In this chapter, the case studies results, the curves found, the equations derived from the curves, and the virtual sensors performance will be presented. The results will be presented according to the sequence of the software used, starting with the simulation in ANSYS, going through the adjustments of the curves and obtaining the equations with Matlab and Excel, and obtaining the performance analysis of the equations by the code developed in Python. The sections with the respective results can be seen in Figure 5.1.

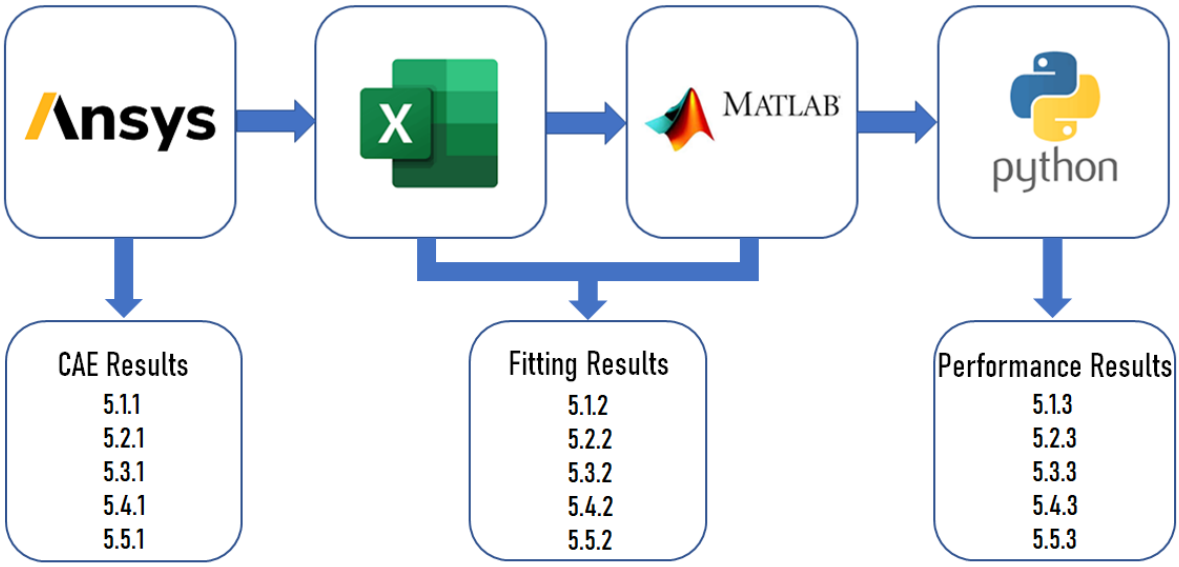


Figure 5.1: Software flux results.

5.1 Plate Case Study Results

In this section the main results for the plate case study with 1 heat gradient will be presented.

5.1.1 Plate ANSYS Simulation Results

First the mesh results will be shown. For this case study, the geometry resultant with the mesh generated an the mesh quality graph can be seen in the Figure 5.2.

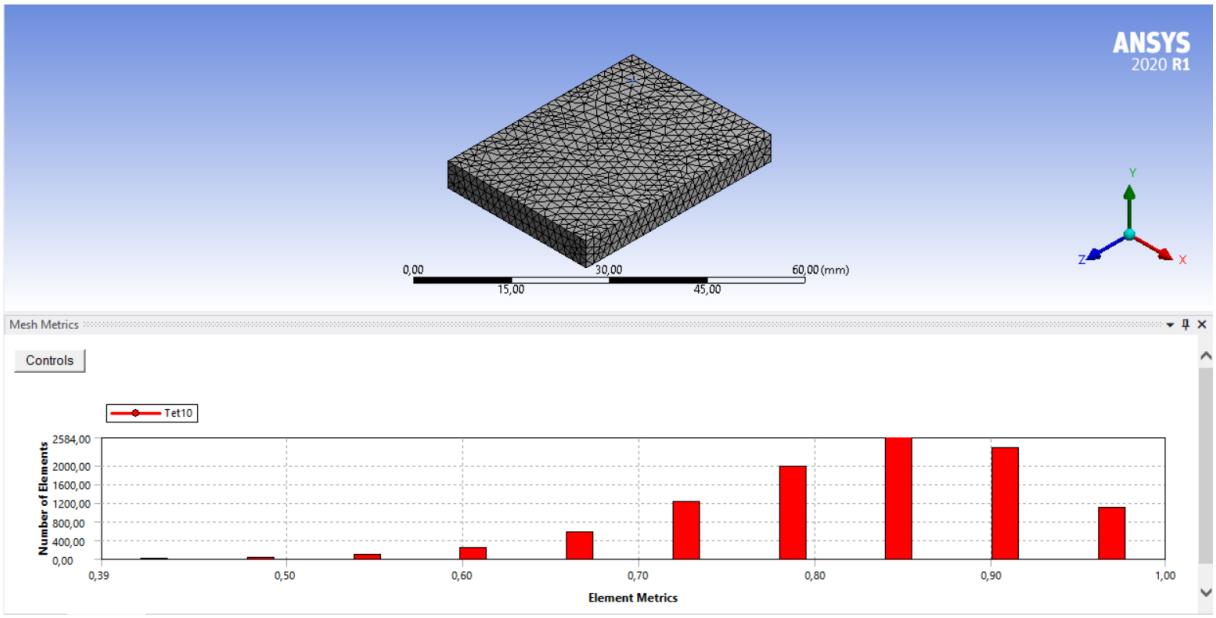


Figure 5.2: Plate mesh graphical results.

Through the Figure 5.2, it is possible to see the final version of the generated mesh, with the tetrahedral elements. The graph shows that most elements are concentrated above 0.7, which is great since 1 is the ideal value. According to the presented, the mesh presents a good result, and meets the requirements of a good simulation at a balanced computational cost.

The results shown in the Table 5.1 contribute to the quality information of the mesh.

Table 5.1: Plate mesh results.

Quality		**Statistics**	
Mesh Metric	Element Quality	Nodes	Elements
Min	0,3947	16004	10113
Max	0,99857		
Average	0,82876		
Standard Deviation	9,2903e-002		

The results shown in the table confirm the values presented in the graph, with respect to the quality of the elements, the average of the elements is of good quality and is above 0,8. The low value found in the standard deviation corroborates with the good result. The values found are the result of several attempts and configuration changes that culminated in this good result. Mesh generation is a step that requires a lot of dedication because the results of the simulation depend on it.

The general solution presents an image with the direction of the heat distribution on the surface of the geometry, and a graph that shows the average temperature of the heat distribution of the entire plate. The general solution can be seen in Figure 5.3

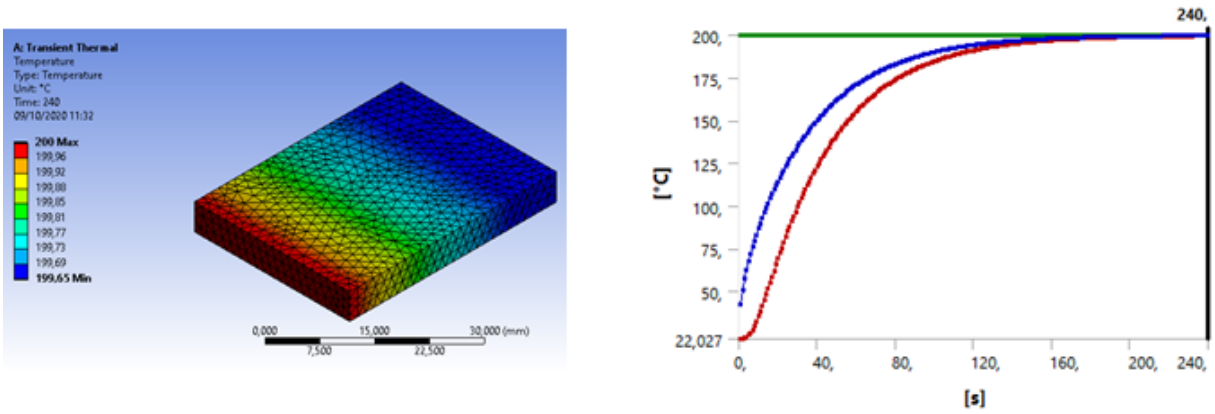


Figure 5.3: Plate average heat distribution results.

The green line of the graph represents the heat source that is constant at 200 °C, the red line represents the minimum heat zones, and the blue line represents the average heat

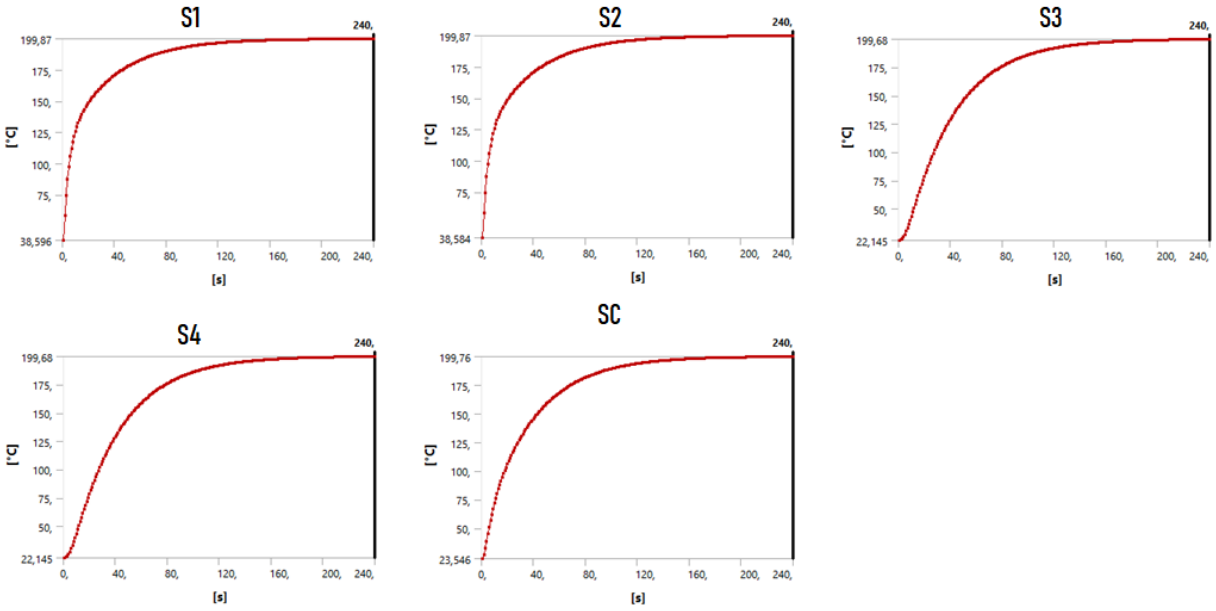
distribution.

For the dimensions and properties of the geometry considered, the metallic plate reaches the maximum heating temperature at approximately 200 s of simulation. The ANSYS software also provides, as a result, a very didactic animation that shows the heat flow along with the geometry of the plate, and from it, it is possible to visually check the behavior of the simulation, which corroborates with the evaluation of the final result.

Through the general result, it is possible to have a good idea of the principle of the problem and if the solution makes sense from a graphical point of view.

To corroborate the results, a spreadsheet containing the points of the graph is also generated. Then it is possible to see through this set of information if the result found is the expected one.

The graphs in Figure 5.12 corresponding to the results of each probe.



p

Figure 5.4: ANSYS plate sensors graph results.

It is possible to observe that the sensors S1 and S2 have similar curves, as the sensors S3 and S4. This is because the sensors S1 and S2 were positioned in the same Z coordinates, in the same way, the sensors S3 and S4.

Another point to note is the difference in slope in the curves. In S3, S4, and SC the slope of the curve is less than S1 and S2, this is because these sensors are more distant from the heat source. So it is expected that there is a delay at the beginning of the response of the curve.

5.1.2 Plate Fitting Curves Results

Before showing the results of the adjustment curves, it is important to know where the virtual sensors are positioned on the metal plate.

Figure 5.5 shows the virtual sensors position in the metallic plate.

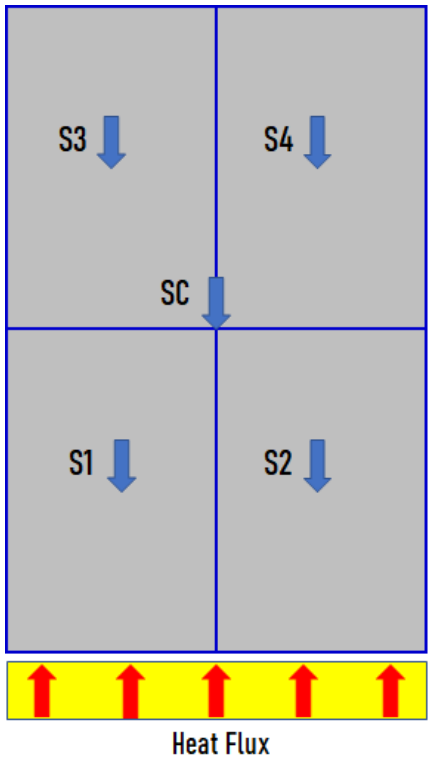


Figure 5.5: Plate virtual sensors position.

The fitting curves process for the plate result in three curve adjustments, two polynomial and one logarithmic.

The results begin with the approximations made by the 8th degree polynomial equations, as can be seen in Figure 5.6.

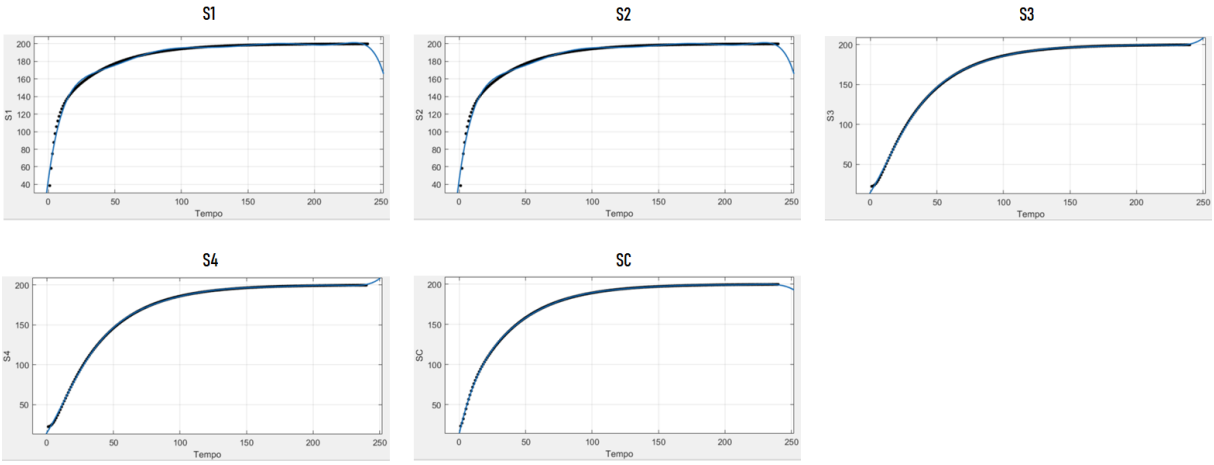


Figure 5.6: Plate 8th-degree polynomial curves fitting results.

The blue dots correspond to the coordinates for the original curves, and the blue line represents the virtual sensor expression that most closely matches the coordinates.

It is possible to notice that the equations approach the curves of the sensors S1 and S2 have a certain oscillation. This effect can lead to errors in inferring the value of these sensors. Another important point to note is that the polynomial of 8th-degree describes the original curve very well, but it is not possible to extrapolate beyond the points of analysis. The error is expected to be greater at the curve boundaries.

The 8th-degree polynomials determination coefficients obtained through Matlab for can be seen in Table 5.2.

Table 5.2: Plate 8th-degree polynomials determination coefficients results [R^2].

S1	S2	S3	S4	SC
0.9929	0.9929	0.9998	0.9998	0.9999

The polynomial of degree 8 presents an excellent factor of determination for the referred curves, for on hand its a god result.

But, in the other hand, have a high coefficient of determination don't guarantee errors in the value inferred. The error impact depends the instant who is observed, and what

is the importance of this moment for the system. In that case, the error is acceptable, because it doesn't dramatically interfere in the final inferences.

The polynomial 8th-degree equations that represent the curves can be seen in the Table 5.3.

Table 5.3: Plate 8th degree polynomial equations resulting.

Sensors	Equations
S1	$(-8.472e-15)*x^8+(8.794e-12)*x^7+(-3.792e-09)*x^6+(8.79e-07)*$ $x^5+(-0.0001185)*x^4+(0.009418)*x^3+(-0.4301)*x^2+(10.92)*x+46.15$
S2	$(-8.473e-15)*x^8+(8.795e-12)*x^7+(-3.792e-09)*x^6+(8.79e-07)*$ $x^5+(-0.0001185)*x^4+(0.009418)*x^3+(-0.4301)*x^2+(10.92)*x+46.15$
S3	$(2.744e-15)*x^8+(-2.836e-12)*x^7+(1.212e-09)*x^6+(-2.756e-07)*$ $x^5+(3.545e-05)*x^4+(-0.002454)*x^3+(0.06152)*x^2+(2.647)*x+14.72$
S4	$(2.745e-15)*x^8+(-2.836e-12)*x^7+(1.212e-09)*x^6+(-2.756e-07)*$ $x^5+(3.545e-05)*x^4+(-0.002454)*x^3+(0.06153)*x^2+(2.647)*x+14.72$
SC	$(-2.035e-15)*x^8+(2.162e-12)*x^7+(-9.616e-10)*x^6+(2.329e-07)*$ $x^5+(-3.362e-05)*x^4+(0.003013)*x^3+(-0.1749)*x^2+(7.1)*x+14.61$

These equations are the virtual sensors that represent the temperature behavior in the selected points for the plate. The process that defined and found the equations up to that point guarantees precision according to the boundary conditions defined in the physical model configured in the CAE software.

To determine the precision and performance of the equations, they were submitted to software that simulates the use of virtual sensors. This procedure will be described below, in the section *Plate Tests Results*.

In the process of adjusting these curves, the Curve Fitting Tool provides 3 values for each coefficient, within a range with minimum, average and maximum values, within a confidence interval of 95 %.

The coefficients chosen for the polynomial equations of 8th-degree are contained in

the average value.

For modeling using 6th-degree polynomials, the result through the fitting tool can be seen in Figure 5.7.

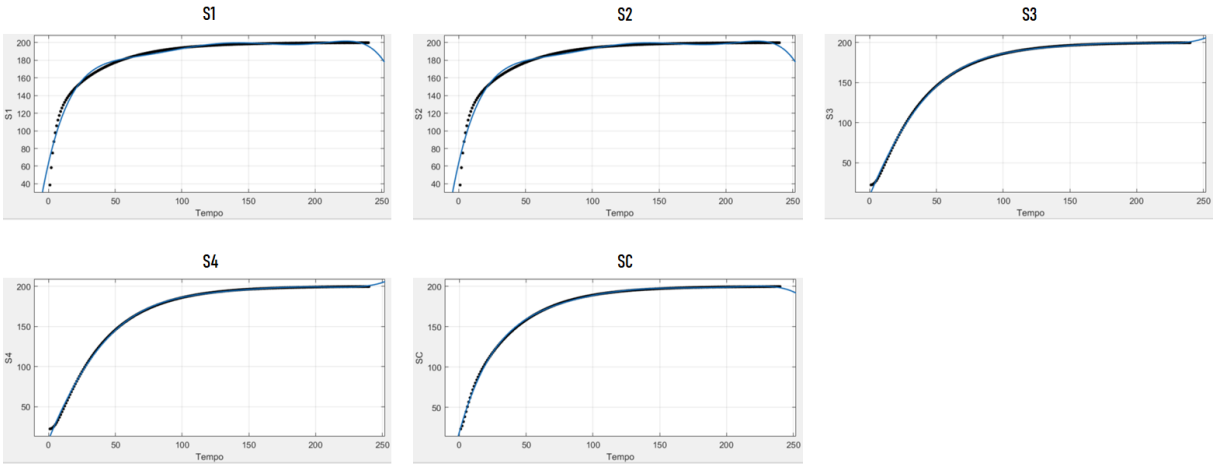


Figure 5.7: Plate 6th-degree polynomial curves fitting results.

Where the blue line represents the original curve provided by the CAE software and the blue dots represent the approximation by the expression of the virtual sensors. The representation of the curves through a polynomial of 6th-degree is less accurate compared to the curves represented by polynomials of 8th-degree. However, it is a minimal loss of precision as can be verified through the Table of the termination coefficient. Looking at the graphs, it is almost imperceptible to detect the variation of the curve. A closer look can see that the oscillation frequency of the function in S1 and S2 is lower when compared to functions approximated by polynomials of 8th degree. In the same way, as was observed in 8th-degree curves, for 6th-degree functions the process cannot be extrapolated because beyond the limits of the function, it no longer represents the phenomenon. This result is experienced because it is characteristic of an approximation by a polynomial function.

Table 5.4: Plate 6th-degree polynomials determination coefficients results [R^2].

S1	S2	S3	S4	SC
0.9807	0.9807	0.9995	0.9995	0.9995

Even so, the loss of precision can mean a big error in the inference of the values depending on which region of the equation was most affected. Therefore, at this stage, it is relative to avoid loss of precision, as it depends on the temperature range of interest. The 6th-degree polynomial expressions that represent the curves can be seen in the Table 5.5.

Table 5.5: Plate 6th degree polynomial equations resulting.

Sensors	Equations
S1	$(-4.611e-11)*x^6+(3.688e-08)*x^5+(-1.158e-05)*x^4+(0.001815)*x^3+(-0.1502)*x^2+(6.516)*x+63.96$
S2	$(-4.611e-11)*x^6+(3.688e-08)*x^5+(-1.158e-05)*x^4+(0.001815)*x^3+(-0.1502)*x^2+(6.516)*x+63.96$
S3	$(9.29e-12)*x^6+(-6.347e-09)*x^5+(1.404e-06)*x^4+(-4.41e-05)*x^3+(-0.02684)*x^2+(4.034)*x+9.137$
S4	$(9.291e-12)*x^6+(-6.348e-09)*x^5+(1.404e-06)*x^4+(-4.414e-05)*x^3+(-0.02683)*x^2+(4.033)*x+9.139$
SC	$(-1.98e-11)*x^6+(1.656e-08)*x^5+(-5.572e-06)*x^4+(0.000979)*x^3+(-0.09861)*x^2+(5.877)*x+19.64$

These equations are the virtual sensors expressions, and are represented by the approximation of the 6th degree polynomial equations. As occurred during the fitting of the 8th-degree polynomial equations, for 6th-degree polynomial equations the fitting tool provides 3 coefficients for the equation within 95% confidence interval. The average values was chosen. To evaluate the performance of these equations, they will be tested in software that simulates the functioning of the virtual sensors, they will be tested in software that simulates the functioning of the virtual sensors, this will be described further ahead in the section *Plate Tests Results*.

The logarithmic curve adjustment process is performed using Excel software. The resulting graphics for the process can be seen in Figure 5.8.

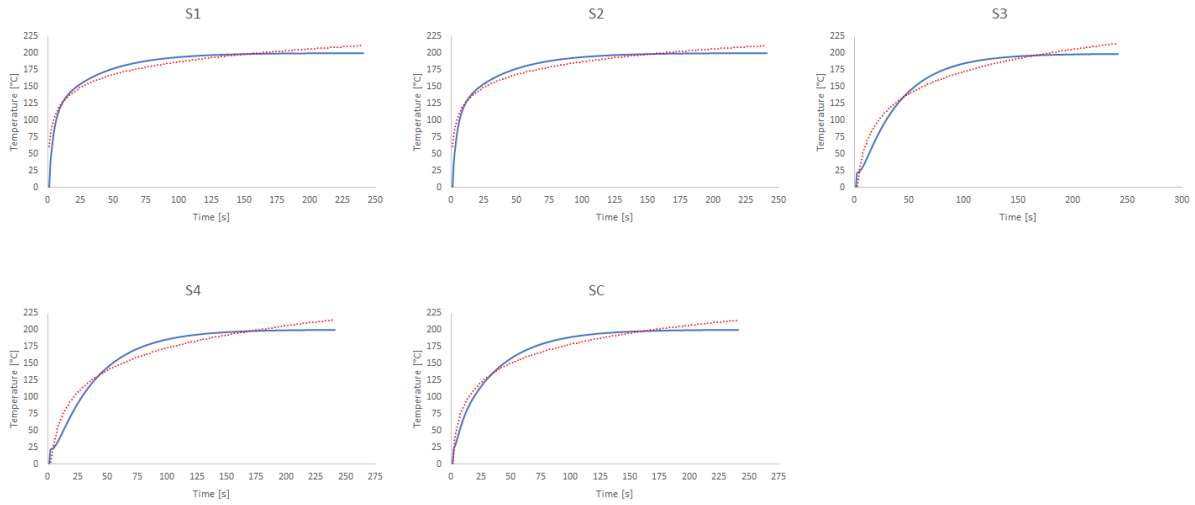


Figure 5.8: Plate logarithmic curves fitting results.

The original curve is represented by the blue line, and the virtual sensors represented by the logarithmic curve is represented by the red line. In this adjustment process, it is verified by the graph that the function has a certain error depending on the region analyzed. This function also presents distortions close to the limits, but with a bigger problem at the origin of the system, due to the characteristic of the logarithmic function. For values very close to zero, the precision is distorted. However, as the function distances itself from the origin of the cartesian system, the behavior of the function is more acceptable.

The determination coefficients can be seen in Table 5.6

Table 5.6: Plate logarithmic equation determination coefficients results [R^2].

S1	S2	S3	S4	SC
0.9037	0.9037	0.9424	0.9424	0.9509

The determination coefficients still up 0.9, but the result can be seen with caution. The logarithmic expression has a good performance when it moves away from the extremes. But even in the middle region, errors are presents.

The logarithmic equations can be seen in Table 5.7.

Table 5.7: Plate logarithmic equations resulting.

Sensors	Equations
S1	$27.542ln(x) + 60.696$
S2	$27.542ln(x) + 60.6956$
S3	$47.873ln(x) - 47.405$
S4	$47.873ln(x) - 47.405$
SC	$40.935ln(x) - 9.9564$

These expressions are the virtual sensors represented by logarithmic functions. The performance of this set of expressions will be explained in the *Plate Performance Tests Results* section using software that simulates the use of virtual sensors.

5.1.3 Plate Performance Tests Results

In this step, tests were performed to determine the performance and the error in each equation. The tests was divided in two parts, the first is the speed test, and the second is the precision tests.

To test the speed parameter was measures the root time, the total time. The time error found by the inverse function was also calculated concerning the real-time. To perform the simulation, 15 input parameters points of temperature was selected for each tested equation, and the points were distributed equidistant along the curve.

For these tests, the sensor SC was used to simulate the real temperature input, the tested topology for this case study is 1 for n. The sensors S1, S2, S3, and S4 are the virtual sensors for these tests. To illustrate the test configuration, Figure 5.9 shows an example of the selected sensor to be the input sensor.

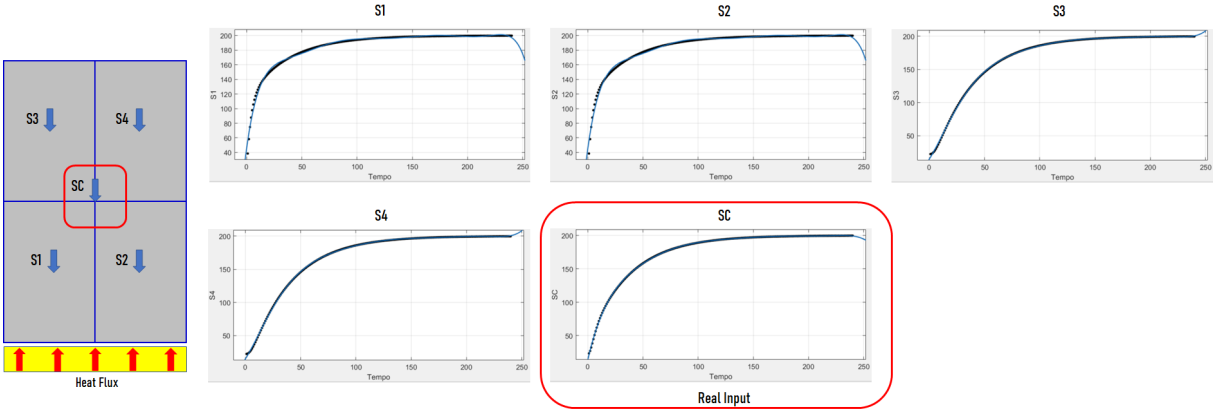


Figure 5.9: Plate position and graphic example.

The results of the first test stage, called speed tests, can be seen in the Table 5.8.

Table 5.8: Virtual sensors speed tests for the plate.

Equation Type	Root Time [s]	Total Time [s]	Time Error [%]
8th-degree polynomial	27,93071	27,9349	0,07496291
6th-degree polynomial	18,16396	10,9888	0,04210004
Logarithmic	1.154776812	1.160384941	0.010827735

The Root Time are the time is necessary to program solve the invert function, to find a time instant. The Total Time is all the time to solve the invert function and the virtual sensors equations. The Time Error is the error between a real time and the time found by the invert function.

As seen earlier, the performance of the 8th-degree polynomial function is reduced by finding the value of the inverse function (Root Time), by the amount of roots calculated in the process. Although the time calculated using the inverse function presents a very small error. The 8th-degree polynomial function would not be a good choice from the point of view of speed analysis.

Compared to the 8th-degree polynomial equation, the 6th-degree polynomial equation is almost 10 s faster to find the value of the inverse function (Root Time), which is a very

good gain in terms of performance. However, it is still not ideal for real-time processing, since one of the desired requirements is to meet the sampling rate of 120 ms from the real sensor. A positive point is an accuracy in the result of the inverse function as can be seen in the Time Error column.

In terms of performance, the logarithmic function achieved much better results compared to polynomial functions. Regarding Root Time, the best performance of the logarithmic function is justified due to the simplicity of solving its inverse function compared to a polynomial function. The inverse logarithmic function is the exponential function, and it is a function that has 1 real root as a result. In the case of polynomials, the number of inverse roots is directly proportional to the degree of the polynomial, and these roots are real and imaginary. At code level, is necessary to treat invalid time values, which occur more in polynomial functions, which means more instructions and more execution time

The other analysis was made is the accuracy of the virtual sensors in relation with CAE results. That measures is important to understand the precision of each equation tested.

Table 5.9 shows the second tests, the virtual sensors percent error.

Table 5.9: Virtual sensors percent error for the plate.

Euqation Type	S1	S2	S3	S4
8th-degree polynomial	3,52105	3,94811	1,040043	0,896274
6th-degree polynomial	1,14611	1,14668	1,661788	1,728106
Logarithmic	2.259309381	2.259322432	6.516826368	6.516826368

When comparing the curves of the CAE model, the 8th-degree polynomial equations have the largest error around 3.5%. Judging whether the accuracy is good depends on the point of analysis in the function and its relevance to the process.

From a numerical point of view, the errors found with the 6th-degree polynomial function are small. However, the result needs to be evaluated carefully, especially when

referring to the values at the ends of the function. Due to the reason for speed and precision, the 6th-degree polynomial equation is the best balanced if compared to the 8th-degree polynomial equation.

Regarding the sensor error, the performance of the logarithmic function was slightly worse than the other sensors. However, when analyzing global aspects, the set developed with logarithmic equations presents a better balance between computational cost and precision factor, for this set of studied curves.

5.2 Plate Case Study Results for 2 Heat Gradient

In this section the main results for the plate case study with 2 heat gradient will be presented.

5.2.1 Plate with 2 Heat Gradients Ansys Simulation Results

Initially, the mesh result can be seen in Figure 5.10.

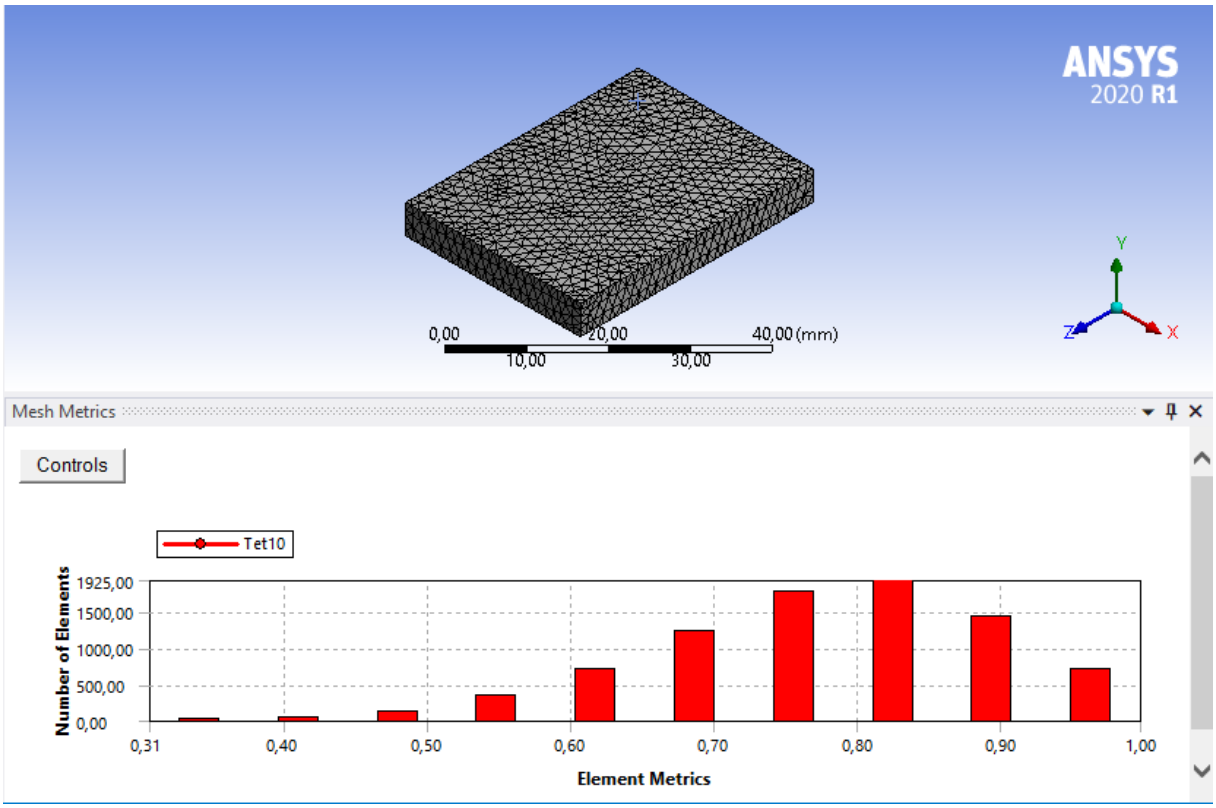


Figure 5.10: Plate with 2 heat gradient, mesh graphical results.

The result obtained for the plate with 2 temperature gradients shows good results regarding the mesh quality. The tetrahedron is the element that composes the mesh, this element has 10 connection nodes, and shown good performance in geometries with flat surfaces, as is the case of the plate.

The graph shows that most elements are contained between 0.7 and 0.9, which is desired for the mesh to provide reliability for the simulation.

The result of the mesh obtained for the plate with 2 temperature gradients is very similar to the result of the plate with 1 gradient, since they share the same geometry and the mesh control mechanism is configured similarly.

The mesh quality results are presents in Table 5.10.

Table 5.10: Plate with 2 heat gradient, mesh results.

Quality		**Statistics**	
Mesh Metric	Element Quality	Nodes	Elements
Min	0,3052	14264	8262
Max	0,99763		
Average	0,77596		
Standard Deviation	0,11601		

The average quality of the elements of 0,7 is one of the indicators of the good quality of the mesh. Another highlight is the low standard deviation.

From the mesh quality point of view, the results confirm a good numerical representation of the simulation.

The 2 gradient heat flux result in the plate and the average heat is represented in Figure 5.11.

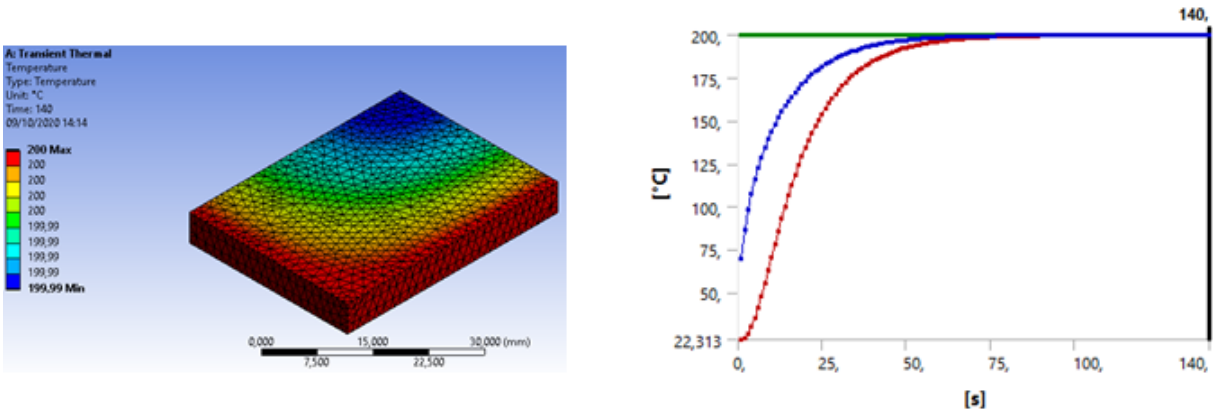


Figure 5.11: Plate with 2 heat gradient, average heat distribution results.

As can be seen in Figure 5.11, the 2 heat gradient flows ins the plate, the red regions represents the heat sources.

In the graph, the green line represents the heat source, the blue line the heat average, and the red line represents the minimum heat in the geometry.

The graphical solution was of great help to understand if the results made sense, and in this way, it was fundamental for the adjustment of the model.

The ANSYS curve results of each sensor can be seen in Table 5.12.

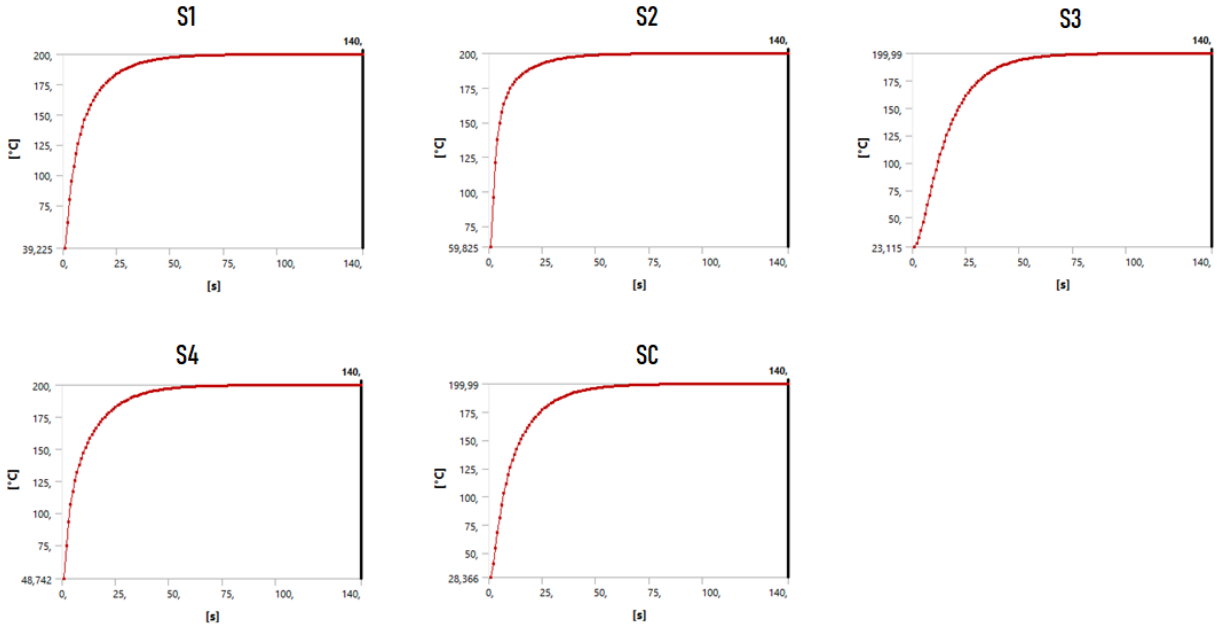


Figure 5.12: ANSYS plate with 2 heat gradient, sensors graph results.

For this simulation, all the sensors has curves with unique characteristics, because the heat source. This behavior will be verified when presenting the equations that represent them.

5.2.2 Plate with 2 Heat Gradients Fitting Curves Results

Before presenting the graphical results, to facilitate visualization of the process, the Figure 5.13 illustrates the position of the virtual sensors on the metal plate with 2 heat gradients.

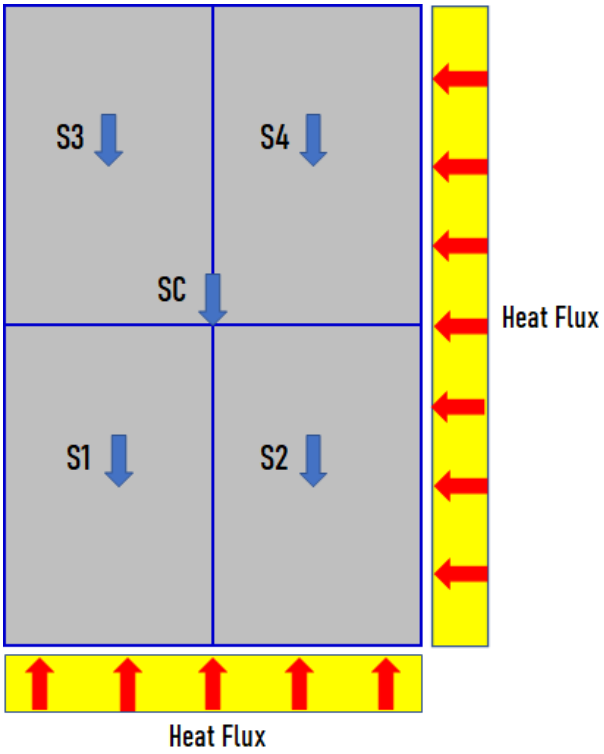


Figure 5.13: Plate with 2 heat gradient virtual sensors position.

For these curves, a polynomial equation of 8th degree, a polynomial equation of 6th degree and a logarithmic equation were tested.

The addition of a second temperature gradient changed the heat distribution on the plate. Even having the same physical characteristics as the first simulation, it was expected that the addition of another heat source would noticeably alter the simulation response, adding more heat to the part.

In this sense, it was possible to notice that the convergence to the value of continuous heating, 200 °C, was reached in approximately 60 s, that is, less than half the time required for the gradient plate, which was approximately 160 s.

The results for fitting a 8th-degree polynomial equation for each sensor can be seen in Figure 5.14.

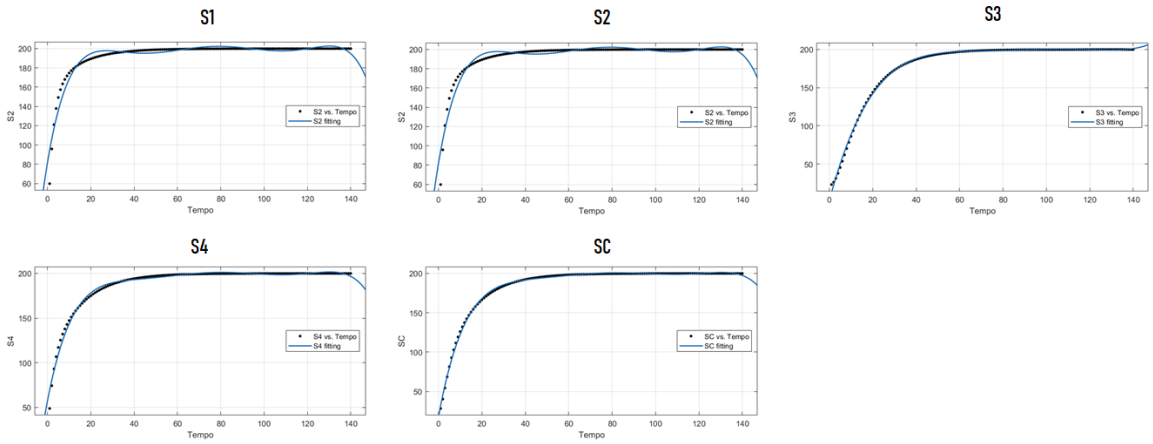


Figure 5.14: Plate with 2 heat gradient, 8th-degree polynomial curves fitting results.

The black dots represent the original curve provided by the CAE simulation, while the blue line represents the approximation by the 8th-degree polynomial function, what are the virtual sensors.

It is important to note that the 8th-degree polynomial function presents an excellent approximation despite showing a small oscillation for S1 and S2.

Another point to note is that at the limits of the graph, a function that does not represent the data with high precision.

To corroborate with the graphics information, Table 5.11 shows the determinant coefficient results for 8th-degree polynomial equations.

Table 5.11: Plate with 2 heat gradient, 8th-degree polynomial determination coefficient results.

S1	S2	S3	S4	SC
0.9985	0.9789	0.9995	0.9961	0.9998

The approximation through the 8th-degree polynomial functions has an excellent determining factor.

It is also possible to observe that in the limits of the graph, the function has distortions and that they need to be considered in order to avoid such situations.

For this case study, the precision obtained through this group of equations is quite high.

The 8th-degree polynomial equations resulting can be seen in Table 5.12.

Table 5.12: Plate with 2 heat gradient, 8th-degree polynomial equation resulting.

Sensors	Equations
S1	$(-4.638e-13)*x^8+(2.864e-10)*x^7+(-7.389e-08)*x^6+(1.034e-05)*x^5+(-0.0008533)*x^4+(0.04245)*x^3+(-1.256)*x^2+(21.17)*x+24.6$
S2	$(-8.423e-13)*x^8+(5.126e-10)*x^7+(-1.297e-07)*x^6+(1.767e-05)*x^5+(-0.001402)*x^4+(0.06553)*x^3+(-1.743)*x^2+(24.22)*x+55.24$
S3	$(3.085e-13)*x^8+(-1.847e-10)*x^7+(4.55e-08)*x^6+(-5.904e-06)*x^5+(0.0004248)*x^4+(-0.01563)*x^3+(0.1554)*x^2+(7.198)*x+10.35$
S4	$(-4.919e-13)*x^8+(3.002e-10)*x^7+(-7.63e-08)*x^6+(1.048e-05)*x^5+(-0.0008437)*x^4+(0.04067)*x^3+(-1.159)*x^2+(18.95)*x+41.42$
SC	$(-2.229e-13)*x^8+(1.415e-10)*x^7+(-3.778e-08)*x^6+(5.536e-06)*x^5+(-0.0004869)*x^4+(0.02655)*x^3+(-0.9001)*x^2+(18.42)*x+8.753$

These equations are the virtual sensors represented by 8th-degree polynomial expressions.

To evaluate the performance of these expressions, they were submitted to software that simulates the use of virtual sensors.

The results of these simulations can be seen in the section *Plate with 2 Heat Gradient Performance Tests Results*.

The fitting curves for 6th-degree polynomial equation can be seen in Figure 5.15.

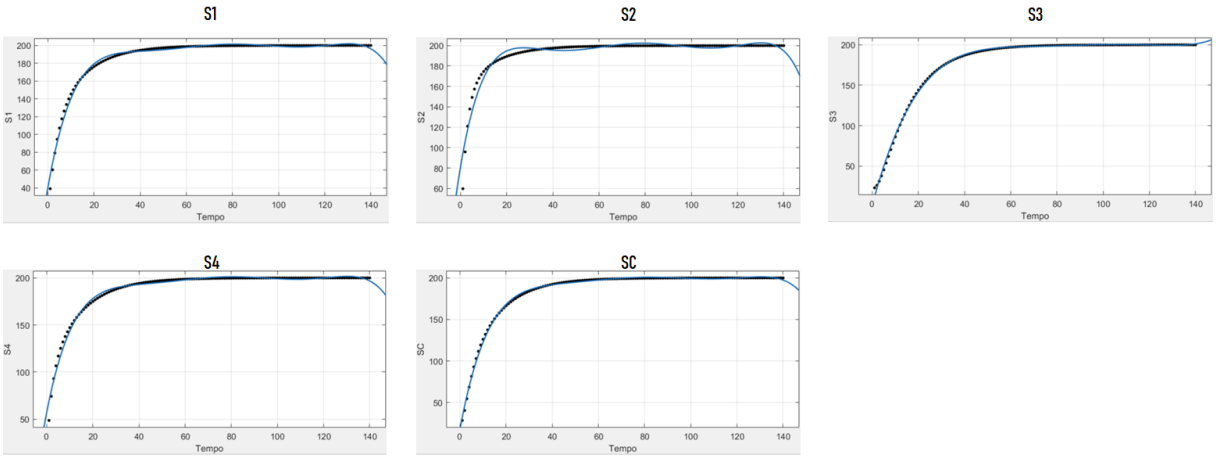


Figure 5.15: Plate with 2 heat gradient, 6th-degree polynomial curves fitting results.

Where the blue line represents the original curves provide by the simulation in CAE software and the blue dots are the virtual sensors approximated by a 6th-degree polynomial expression.

Comparing this result with presented in Figure 5.14, the difference is very small Carefully, it is possible to notice that for the S1 sensor, the oscillations decreased.

In general, there is a small displacement in the approximation in all graphics, which can mean loss of precision.

For this simulations the precision loss is very small in comparison with the 8th-degree polynomial function, that difference can be seen in Table 5.13.

Table 5.13: Plate with 2 heat gradient, 6th-degree polynomial determination coefficient results.

S1	S2	S3	S4	SC
0.9909	0.936	0.9987	0.9878	0.998

The table 5.13. shows the decrease in the representativeness of the 6th-degree polynomial function compared to the 8th-degree polynomial, in particular the S2 sensor, which decreased from 0.9789 to 0.936.

The 6th-degree polynomial equations can be seen in Table 5.14

Table 5.14: Plate with 2 heat gradient, 6th-degree polynomial equation resulting.

Sensors	Equations
S1	$(-1.331e-09)*x^6+6.359e-07*x^5+(-0.0001201)*x^4+0.0114*x^3+(-0.5731)*x^2+14.7*x+40.73$
S2	$(-1.657e-09)*x^6+7.747e-07*x^5+(-0.0001418)*x^4+0.01284*x^3+(-0.5984)*x^2+13.5*x+81.69$
S3	$9.145e-11*x^6+(-8.712e-09)*x^5+(-8.044e-06)*x^4+0.002184*x^3+(-0.2257)*x^2+10.71*x+1.79$
S4	$(-1.113e-09)*x^6+5.304e-07*x^5+(-9.996e-05)*x^4+0.009494*x^3+(-0.4805)*x^2+12.58*x+57.18$
SC	$(-1.038e-09)*x^6+5.068e-07*x^5+(-9.859e-05)*x^4+0.009774*x^3+(-0.5244)*x^2+14.8*x+17.92$

These expressions are the virtual sensors for a 6th-degree polynomial approximation. Each sensor has a characteristic equation that represents the temperature behavior in the select point. Performance tests was made by a software that simulates the virtual sensors in section *Plate with 2 Heat Gradient Performance Tests Results*.

The logarithmic fitting curves resulting can be seen in Figure 5.16

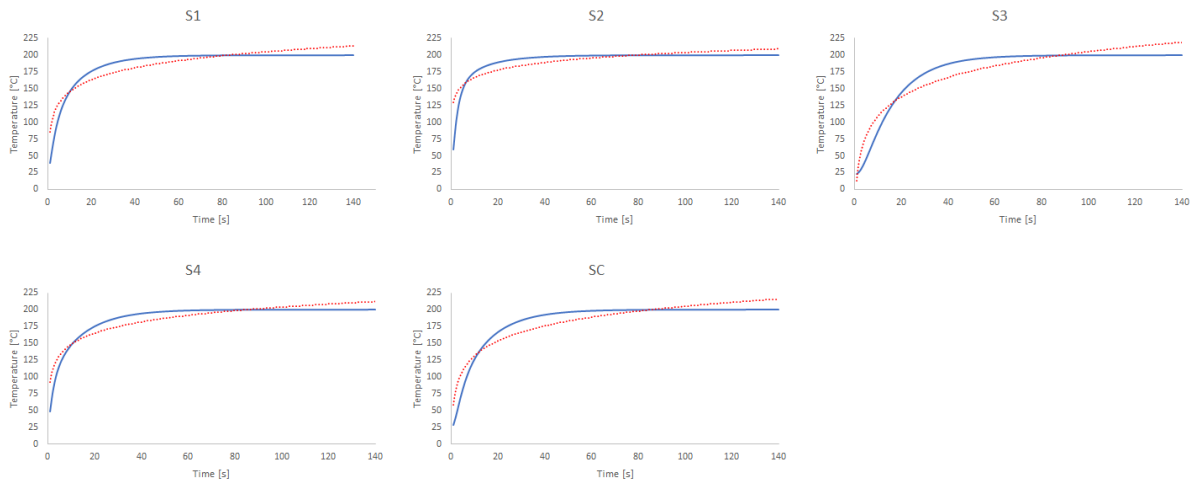


Figure 5.16: Plate with 2 heat gradient, logarithmic curves fitting results.

The blue line represents the original function provide by the CAE simulation and the red line represents the approximation using the logarithmic function for the virtual sensors. It is evident that for this case study, the logarithmic function presents a greater error when compared to polynomial functions.

However, it is important to emphasize that even with a lower precision, this does not mean that the function cannot be used, since it is a matter of determining which error is accepted by the inference.

Table 5.15 presents the results for determination coefficients for logarithmic equations.

Table 5.15: Plate with 2 heat gradient, logarithmic determination coefficient results.

S1	S2	S3	S4	SC
0.817	0.6946	0.8805	0.8404	0.8404

As verified at the graphical level, the data presented confirm that the precision decreased through the approximation of the logarithmic function.

The S2 sensor, by the curve characteristic, presents the greatest loss of approach, around 20% .

In general, the others sensors for logarithmic functions, the curves lost almost 10% of the ability to approximate the original function.

For the other hand, the expression resultants are shorter when compared with the polynomial, as can be seen in the Table 5.16

Table 5.16: Plate with 2 heat gradient, logarithmic equation resulting.

Sensors	Equations
S1	$25.801ln(x) + 86.154$
S2	$16.159ln(x) + 129.52$
S3	$41.835ln(x) + 12.559$
S4	$24.149ln(x) + 93.09$
SC	$31.971ln(x) + 58.208$

These expressions are the virtual sensors, and each equation represents the temperature behavior in the selected points. The precision of these equations is rationed with the curve type, that curve is provided by the CAE simulation and depends of the boundary conditions programmed.

The performance of each equation is directly linked to the type of curve resulting from the CAE simulation process, other details regarding the performance of the equations will be shown in the section *late with 2 Heat Gradient Performance Tests Results* where they are tested by software that simulates the use of virtual sensors.

5.2.3 Plate with 2 Heat Gradient Performance Tests Results

For the simulation of the code, the SC sensor was selected to provide input parameters through 15 points chosen along the graph curve. The topology used was the 1 for n and S1, S2, S3, and S4 was the virtual sensors for these tests. Figure 5.17 illustrates the sensor used to input the real temperature and they respective curve.

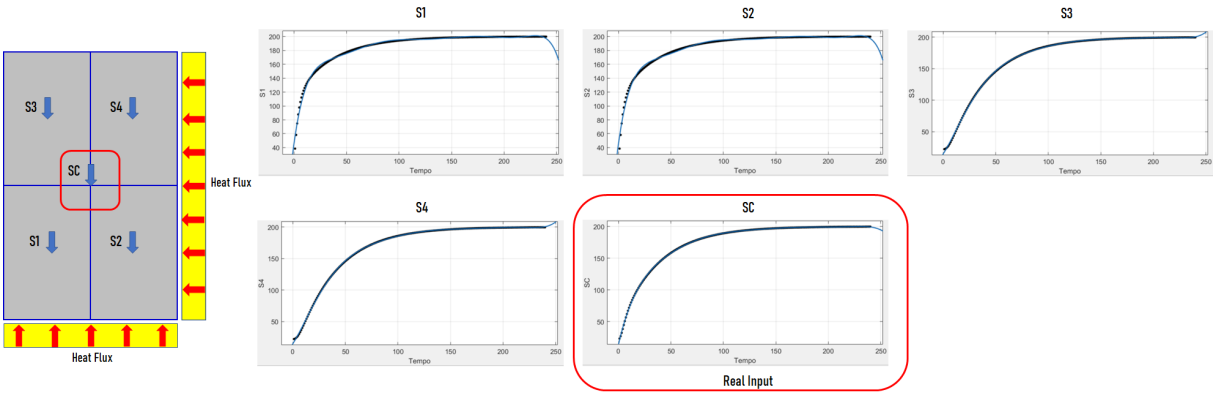


Figure 5.17: Plate with 2 heat gradient, sensor position and graphic example.

The performance test was separated into two analyzes, the first referring to the code execution speed and the second referring to the precision found for each sensor.

The first analysis consists of measuring the root time, the total time, and the error between the instant of real-time and the time calculated by the inverse function. The second analysis is the measure of error accuracy between the values inferred by the expressions

of the virtual sensors and the CAE curves.

Table 5.17, presents the 8th-degree polynomial speed test.

Table 5.17: Virtual sensors speed tests for the plate with 2 heat gradient.

Equation Type	Root Time [s]	Total Time [s]	Time Error [%]
8th-degree polynomial	23.12857768	23.1344364	0.19197058
6th-degree polynomial	9.008682221	9.008682221	0.137486003
Logarithmic	0.277428508	0.281203032	0.007978269

In the parameters of the speed test, it is noticed that 8th-degree polynomial function has a poor result, than the 6th-degree polynomial function has a intermediate results. It is possible to verify that polynomial functions take much longer to find the inverse function. This is expected as there are many more roots to be found. From the point of view of the code, in addition to the delay caused by the resolution of the inverse function, there is the treatment of the appropriate root, which does not take much time to be solved, but it is still an extra time.

Concerning speed tests, the logarithmic equation has a better performance than a polynomial functions. In terms of Root Time and Total Time, compared to the 8th-degree polynomial equation, the 6th-degree polynomial equation is almost 10 s faster to find the value of the inverse function. The Time Error present a better performance when compares to the 8th-degree polynomial equation.

In relation of the Time Error, the best performance was the logarithmic equations, with an high accuracy.

The accuracy of the virtual sensors can be seen in the Table 5.18.

Table 5.18: Virtual sensors percent error for the plate with 2 heat gradient.

Equation Type	S1	S2	S3	S4
8th-degree polynomial	0.001406492	0.650373099	0.763804816	1.266911105
6th-degree polynomial	0.209840765	0.414101556	0.885982012	0.389002935
Logarithmic	0.326061111	0.691224472	0.27468896	0.632606974

For this case study, with respect to precision, the results of the polynomial functions

were very close to each other, except for the S1 sensor represented by the 8th-degree polynomial equation, which presented an excellent result.

For the logarithmic equation, the results were average, but with the worst result for S1 and S2.

It is important to realize that the margins involved are very small, in the sense of a more balanced function of the aspect of speed and precision, the logarithmic function has a small advantage.

5.3 Cylinder Case Study Results

In this section the main results for the cylinder case study with 1 heat gradient will be presented.

5.3.1 Cylinder ANSYS Simulation Results

The results for a cylinder mesh control techniques can be seen in Figure 5.18.

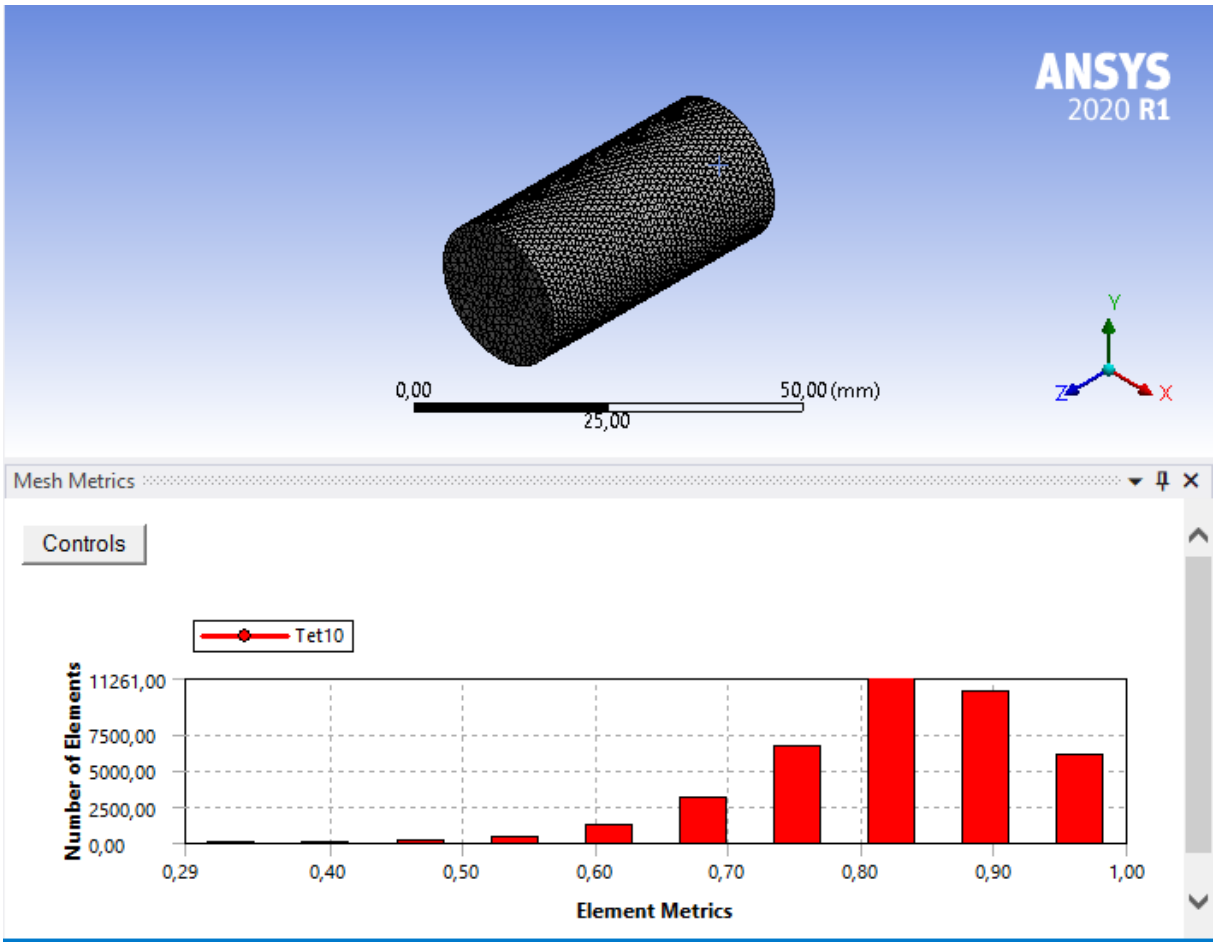


Figure 5.18: Cylinder mesh graphical results.

The image of the cylinder mesh shows the achieved density. Through the graph it is possible to understand the good performance of the mesh obtained. Most elements are above 0.6, and the mesh is composed of tetrahedral elements with 10 connection nodes.

Through the mesh results table it is possible to see that the average elements have 0.83 quality, which is desired, for improve the reliability of the simulation results. These results can be confirmed in the Table 5.19.

Table 5.19: Cilynder mesh results.

Quality		**Statistics**	
Mesh Metric	Element Quality	Nodes	Elements
Min	0,28991	58542	38793
Max	0,99996		
Average	0,8309		
Standard Deviation	9,6241e-002		

The average elements and the standard deviations show the good mesh quality. The number of nodes and elements is directly linked to the representation capacity of the phenomenon. In this case, as it is simple geometry, achieving good representation is expected. The mesh results shows that at the level of mesh generation, the results reached are very good. The heat distribution in the cylinder geometry can be seen in Figure 5.19. Cylinder heat

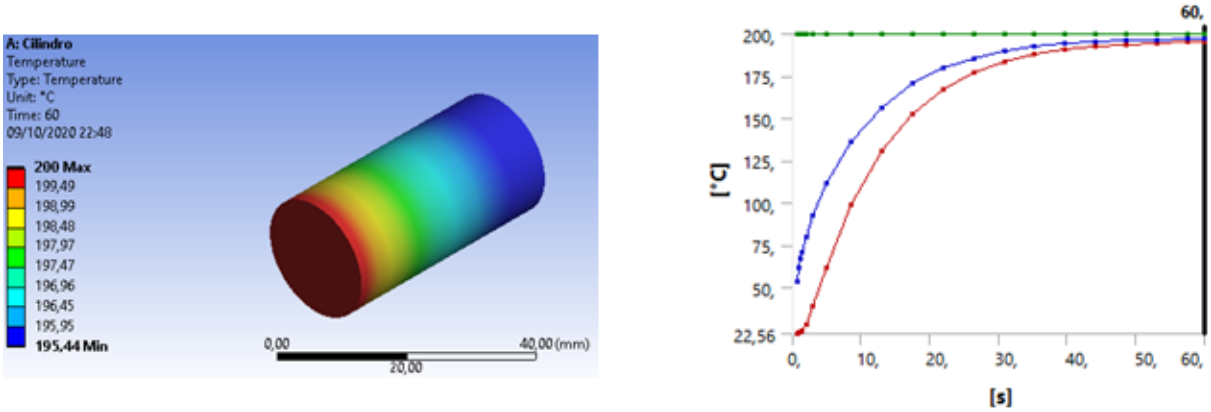


Figure 5.19: Cylinder heat with average graph results.

The red area of the cylinder represents the source of the heat, which spreads along the

geometry. At the graphic, the green line represents the heat source, that is continuous, the blue line is the average heat, and the red line represents the minimum heat.

The sensor curves result simulated in ANSYS can be seen in Figure 5.20.

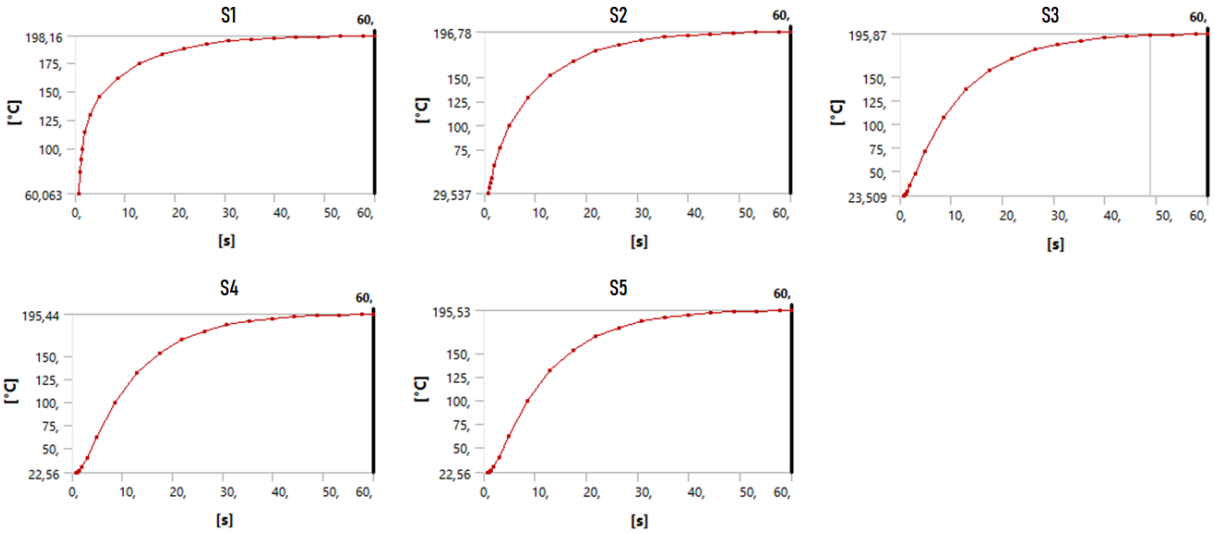


Figure 5.20: ANSYS cylinder sensors graph results.

The resulting curves from ANSYS simulation are distinct from each other, this is expected since the temperature sensors are arranged along the cylinder in the direction of the heat source. As the sensors move away from the heat source, there is a smoothing in the vertical inclination of the curves. In this sense, it is evident the smoothing of the slope of the curve when comparing the graphs of S1 and S5.

5.3.2 Cylinder Fitting Curves Results

To facilitate visualization of the process, the Figure 5.21 illustrates the position of the virtual sensors on the metal plate with 2 heat gradients. For this case study, the S1 and S2 sensors have a continuous behavior because they are in contact with the heat source, for this reason they are not present in the analyzes.

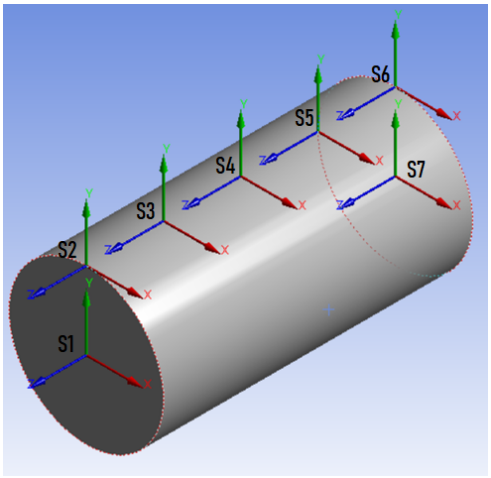


Figure 5.21: Cylinder virtual sensors position.

The results for fitting curves of 8th-degree functions can be seem in Figure 5.22.

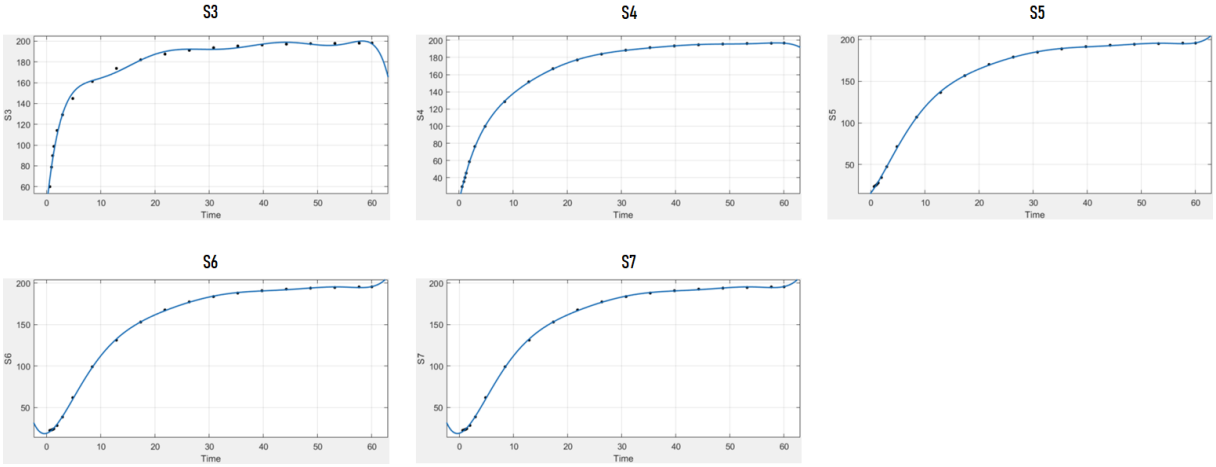


Figure 5.22: Cylinder 8th-degree polynomial curves fitting results.

Where the blue dots represent the original curve provided by the CAE software and the blue lines represent the adjustment curves, or in other words, they represent the approximation expressions of the virtual sensors.

For the S1 sensor, the function has oscillations along the curve, and accuracy is expected to be reduce for this reason.

The objective to use a 8th-degree function is the to reach the precision.

To corroborate with this information, the determination coefficients shows a high

values, as can be seen in Table 5.20.

Table 5.20: Cylinder 8th-degree polynomial determination coefficient [R^2] results.

S1	S2	S3	S4	SC
0.9958	0.9999	0.9999	0.9999	0.9999

As expected, the accuracy of S1 is slightly less representative than the other sensors, but it is still very high. In that sense, the 8th-degree polynomial prove to provide a high approximation for this case study.

The equations performed for the curve fitting process can be seen in the table 5.21.

Table 5.21: Cylinder 8th-degree polynomial equations resulting.

Sensors	Equations
S1	$(-7.469e-10)*x^8+(1.943e-07)*x^7+(-2.094e-05)*x^6+(0.001208)*x^5+(-0.04024)*x^4+(0.7808)*x^3+(-8.512)*x^2+(49.48)*x+40.81$
S2	$(-1.536e-10)*x^8+(4.142e-08)*x^7+(-4.67e-06)*x^6+(0.0002858)*x^5+(-0.01034)*x^4+(0.2281)*x^3+(-3.134)*x^2+(29.04)*x+12.06$
S3	$(2.001e-10)*x^8+(-5.145e-08)*x^7+(5.453e-06)*x^6+(-0.0003063)*x^5+(0.009708)*x^4+(-0.1667)*x^3+(1.117)*x^2+(8.682)*x+15.94$
S4	$(2.81e-10)*x^8+(-7.32e-08)*x^7+(7.888e-06)*x^6+(-0.0004536)*x^5+(0.01492)*x^4+(-0.2751)*x^3+(2.374)*x^2+(1.993)*x+19.07$
SC	$(2.809e-10)*x^8+(-7.317e-08)*x^7+(7.886e-06)*x^6+(-0.0004535)*x^5+(0.01492)*x^4+(-0.2751)*x^3+(2.375)*x^2+(1.993)*x+19.07$

These equations are the virtual sensors represented by 8th-degree polynomial expressions. The performance of this set of expressions will be explained in the *Plate Performance Tests Results* section using software that simulates the use of virtual sensors.

The fitting for 6th-degree polynomial equation is illustrated in Figure 5.23.

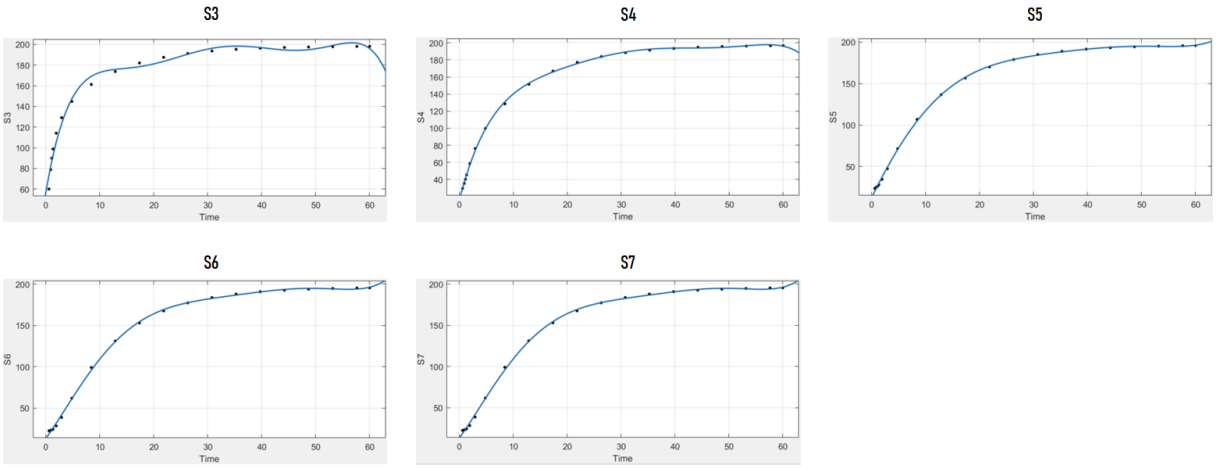


Figure 5.23: Cylinder 6th-degree polynomial curves fitting results.

Where the blue line is the real curve from CAE software and the blue dots are the virtual sensors expression. For the 6th-degree polynomial equation, the function also distorts at the limits of the graph. It is possible to see that the oscillation in sensor S1 is smoother, but it is still present.

The values of determination coefficients in Table 5.22 show the respective value for each sensor.

Table 5.22: Cylinder 6th-degree polynomials determination coefficients results [R^2].

S1	S2	S3	S4	SC
0.9879	0.9996	0.9997	0.9994	0.9994

These values show a slightly difference in relation with 8th-degree polynomial function. The values found are very good from the point of view of approximation of the function.

Table 5.23 present the 6th-degree polynomial functions.

Table 5.23: Cylinder 6th-degree polynomial equations resulting.

Sensors	Equations
S1	$(-2.772e-07)*x^6+(5.515e-05)*x^5+(-0.004269)*x^4+$ $(0.1623)*x^3+(-3.162)*x^2+(30.8)*x+56.27$
S2	$(-1.136e-07)*x^6+(2.334e-05)*x^5+(-0.001898)*x^4+$ $(0.07862)*x^3+(-1.806)*x^2+(24.29)*x+16.05$
S3	$(4.404e-08)*x^6+(-7.937e-06)*x^5+(0.0004998)*x^4+$ $(-0.01006)*x^3+(-0.2228)*x^2+(13.32)*x+12.13$
S4	$(9.687e-08)*x^6+(-1.852e-05)*x^5+(0.001323)*x^4+$ $(-0.04108)*x^3+(0.3487)*x^2+(9.074)*x+13.21$
SC	$9.699e-08)*x^6+(-1.855e-05)*x^5+(0.001324)*x^4+$ $(-0.04113)*x^3+(0.3494)*x^2+(9.075)*x+13.21$

These equations are the virtual sensors that represent the temperature behavior in the selected points for the cylinder. To determine the precision and performance of the equations, they were submitted to software that simulates the use of virtual sensors. This procedure will be described below,in the section *Plate Tests Results*.

For this case study, a logarithmic function has a good approximation of the curves as can be seen in Figure 5.24.

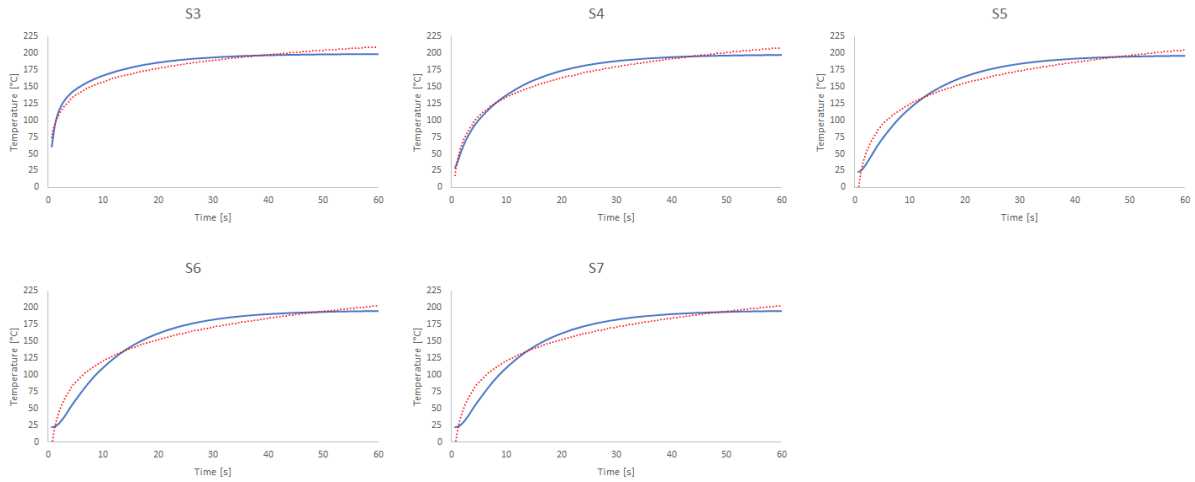


Figure 5.24: Cylinder logarithmic curves fitting results.

The blue line represents the original function from the CAE, and the red line the function approach by an virtual sensor logarithmic equation. To corroborate with the graphic results, the determination coefficients has a good values and can be seen in Figure 5.24.

Table 5.24: Cylinder logarithmic determination coefficients $[R^2]$ results.

S1	S2	S3	S4	SC
0.972	0.9871	0.9733	0.9652	0.9652

For these curves, all the results is above 0.9, and that represent a good result. Although the polynomial functions performed better in this regard, this does not prevent the use of the logarithmic function, as the difference in approximation to the original function is slightly smaller.

The logarithm equations can be seen in the Table 5.25.

Table 5.25: Cylinder logarithmic equations resulting.

Sensor	Equation
S1	$29.207ln(x) + 89.948$
S2	$41.268ln(x) + 39.353$
S3	$45.107ln(x) + 20.147$
S4	$45.889ln(x) + 15.317$
SC	$45.913ln(x) + 15.311$

These equations are the virtual sensors expressions, and are represented by the approximation of the logarithmic functions. Performance tests was made by a software that simulates the virtual sensors in the next section.

5.3.3 Cylinder Performance Tests Results

For these tests, the sensor S3 provides the input parameter in a 1 for n topology in a set of 15 points distributed along the curve. To make it easier to understand the position of the input sensor, as well as its curve, the Figure shows an example that was followed for all families of curves 5.25.

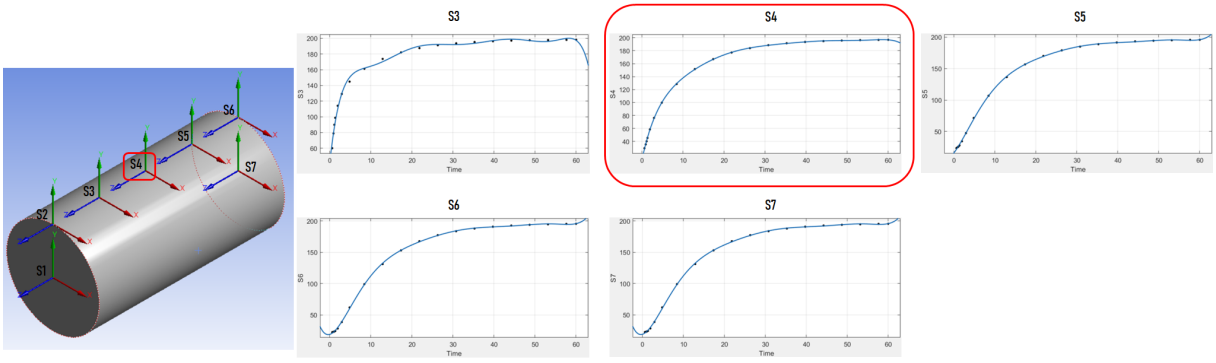


Figure 5.25: Cylinder sensors position and graphic example.

The tests consists in two parts, the first is the speed test and the second is the precision test. The speed tests results can be seen in Table 5.26.

Table 5.26: Virtual sensors speed tests for the cylinder.

Equation Type	Root Time [s]	Total Time [s]	Error [%]
8th-degree polynomial	24,27875	24,28529	0,00055
6th-degree polynomial	11,64629	11,65671	0,04564
Logarithmic	0,3072223	0,314386	0,09703

Despite the good approximation, the 8th-degree polynomial function does not show good results for real-time processing. A resolution of the inverse function takes considerable time, as can be seen in Root Time in the Table 5.26.

For the 6th-degree polynomial function, the Root Time was lower compared to the 8th-degree polynomial function. However, it still has a high refresh rate for real-time processing.

From the speed point of view, the logarithmic function is much faster in solving the inverse function, and in the total processing time of the virtual sensors. A point to highlight is a small increase in the error of determining the value of time in relation to the functions already studied.

For accuracy tests, the Table 5.27 presents the virtual sensors errors.

Table 5.27: Virtual sensors percent error for the cylinder.

Equation Type	S1	S2	S4	S5
8th-degree polynomial	26,0862	9,6496	5,92346	18,3525
6th-degree polynomial	5,1103	3,10762	0,58442	0,843421
Logarithmic	0,100010153	0,113084171	0,01973816	0,019774272

The error in sensor S3 was not calculated as it was used as an input parameter. In general, the error presented by the virtual sensors is moderate, due to the observation range of the data. Within the limits of the function, there are considerable distortions in the values, which caused the error to increase for all sensors.

For this case study, the 8th-degree polynomial function has the worst relative error performance among the compared functions

For the 6th-degree polynomial equation, the error decreased compared to the 8th-degree function. The definition of a large or small error is relative, as it depends on the instant in which it is being analyzed, due to the relevance of the point to the process.

The error presented by the logarithmic function is smaller when compared to polynomial functions. Through the results that were qualified, the natural choice to represent the virtual sensors in this case study is the logarithmic functions by the balance between precision and speed.

5.4 Pipe Case Study Results

In this section the main results for the pipe case study with 1 heat gradient will be presented.

5.4.1 Pipe Ansys Simulation Results

For the generation of the pipe mesh, it was necessary to focus on the edges. As the tube thickness is 2.5 mm, it is a good practice to set the size of the element to a more representative value. In this sense, a refinement element was added at the edges to improve the quality of the mesh.

Figure 5.26 illustrates the geometry and mesh results for the Pipe model.

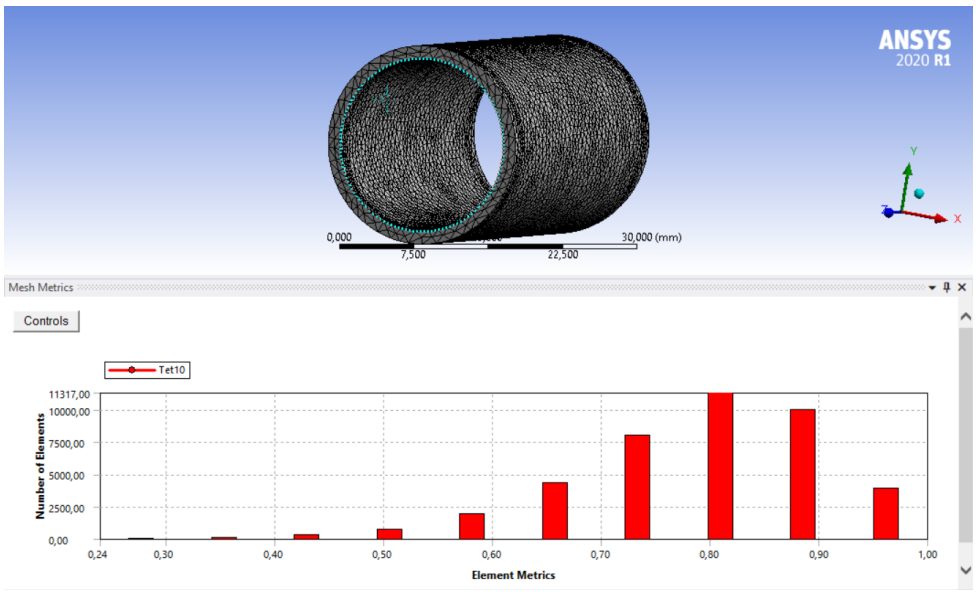


Figure 5.26: Pipe mesh graphical results.

The quality graph informs that the elements that make up the mesh are predominantly tetrahedrons with 10 connection nodes. The graph also informs that concerning quality, most of the mesh elements are above 0.6, which is good since 1 is ideal.

The mesh Table 5.28 results corroborates with the quality information about the pipe mesh.

TABELA MESH

Table 5.28: Pipe mesh results.

Quality		**Statistics**	
Mesh Metric	Element Quality	Nodes	Elements
Min	0,23933	67292	40391
Max	0,99999		
Average	0,79302		
Standard Deviation	0,10902		

The Table reports that the average of the elements generated is in the range of 0.79, which is an interesting result. The mesh has a considerable amount of knots and elements depending on the refining process. For this geometry, several configurations were tested, but the best fit is what is presented.

The pipe heat result represents the heat flux trough the part. The graph inform the average heat behavior in that geometry Figure 5.27 inform the general solution.

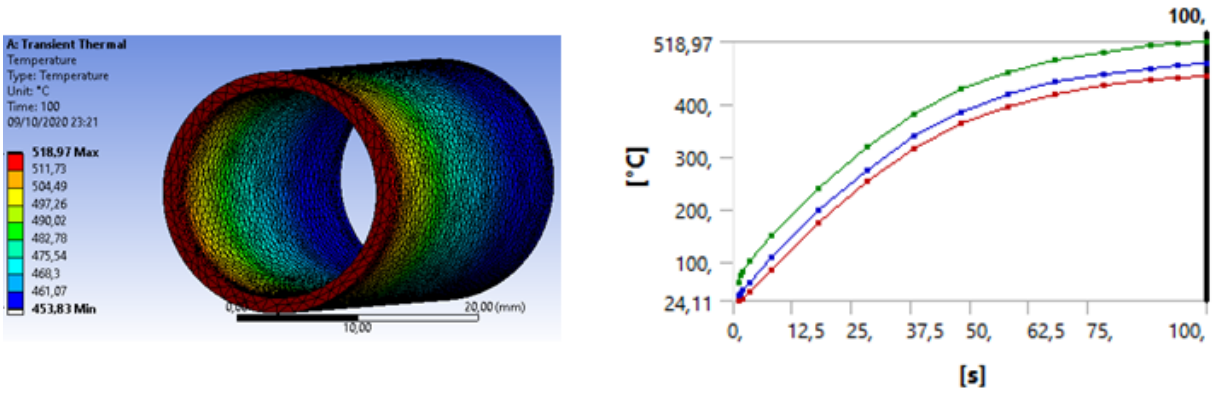


Figure 5.27: Pipe average heat distribution results.

Different from the other analyzes performed so far, the pipe presents heating through a heat flow, as can be seen in the graph. The green line represents the heat flow, the blue line is the average of the heat in the piece, and the red line is the minimum temperature.

The heat flow has an increasing behavior, varying in time, while the heat provided by the Temperature element is constant.

The sensors results of the ANSYS simulation can be seen in Figure 5.28.

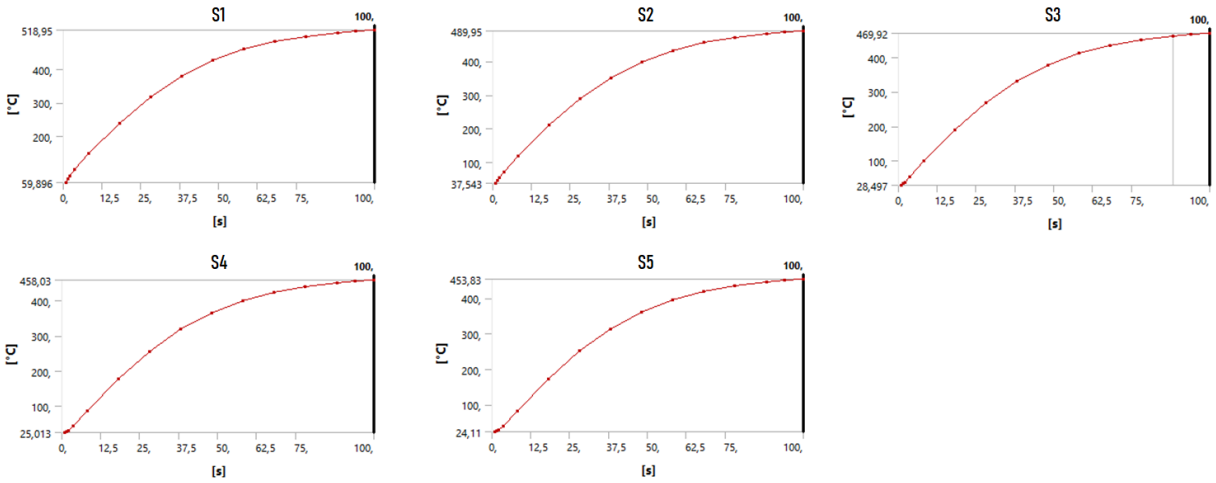


Figure 5.28: ANSYS pipe sensors graph results.

The curves provide from a heat flux process presented be more smooth inclination when compares with other case studies.

5.4.2 Pipe Fitting Curves Results

Before presenting the graphical results, to facilitate visualization of the process, the Figure 5.29 illustrates the position of the virtual sensors on the metal pipe.

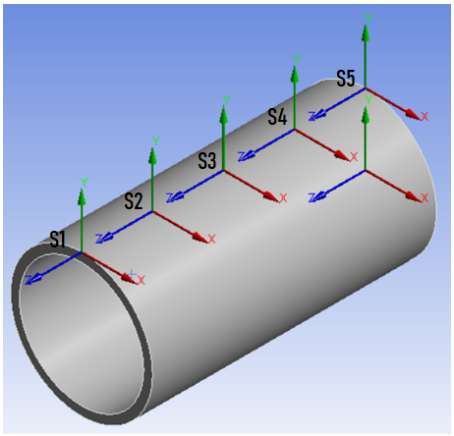


Figure 5.29: Pipe sensors position.

The fitting curve for that graphs was represented by an 3rd degree polynomial function. The fitting curve results can be seen in Figure 5.30.

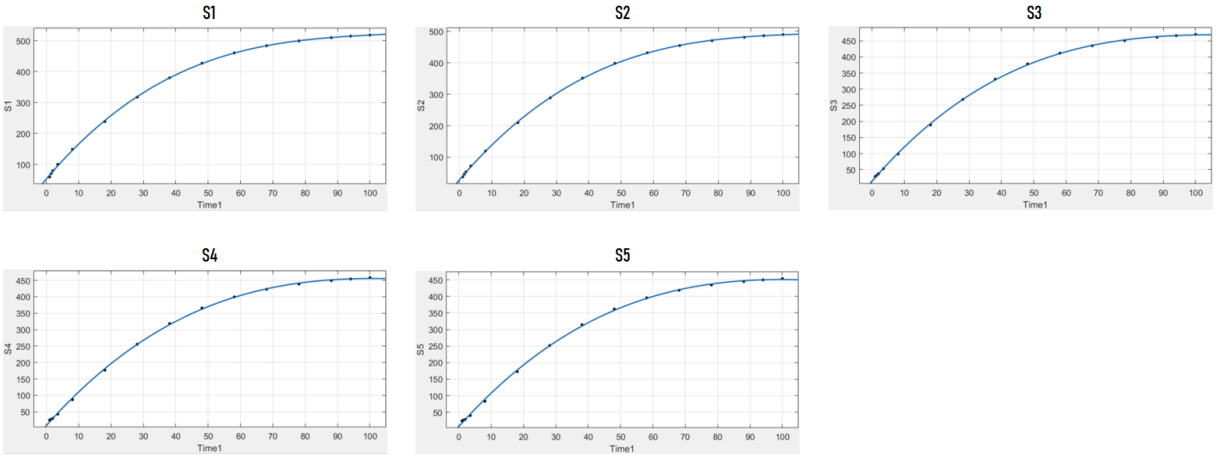


Figure 5.30: Pipe 3rd degree polynomial curves fitting results.

Where the blue dots represent the original function and the blue line represents the adjustment curve, or virtual sensors, there is a very interesting curve fitting performance. The piper curve adjustment approach was performed differently, compares with the case studies previously seen. This is because, for the resulting curves, a 3rd degree polynomial function was more than enough to approximate the original curve with great precision.

The determination coefficients shows the high values in the approach functions and can be seen in Table 5.29.

Table 5.29: Pipe 3rd degree polynomial determination coefficients results.

S1	S2	S3	S4	S5
0.9998	0.9999	0.9999	0.9997	0.9996

The determination coefficients, in this case, are very high and represent almost 100% of the original function.

Unlike the other case studies, due to the characteristics of the curve, the function that best fitted the curve was a 3rd degree polynomial.

The equations who represent the curves can be seen in Table 5.30.

Table 5.30: Pipe 3rd degree polynomial equations resulting.

Sensors	Equations
S1	$(0.0003635) * x^3 + (-0.1136) * x^2 + (12.38) * x + 53.46$
S2	$(0.0003307) * x^3 + (-0.1078) * x^2 + (12.09) * x + 27.94$
S3	$(0.0002697) * x^3 + (-0.09706) * x^2 + (11.55) * x + 14.61$
S4	$(0.0002201) * x^3 + (-0.08826) * x^2 + (11.1) * x + 8.448$
S5	$(0.0002021) * x^3 + (-0.08502) * x^2 + (10.93) * x + 6.683$

These equations are the virtual sensors, that describes the thermal behavior in each point selected. The approximation of the original function through the 3rd degree polynomial equations obtained an impressive result. This is possible due to the characteristics of the modeled curves. This proves that the adjustment of a curve with high precision by means of a polynomial function does not necessarily need to involve a function with a high degree, in fact, it depends on the curve.

To provide a comparison, an approach using a logarithmic function was made. The logarithmic fitting curves results can be seen int Figure 5.31.

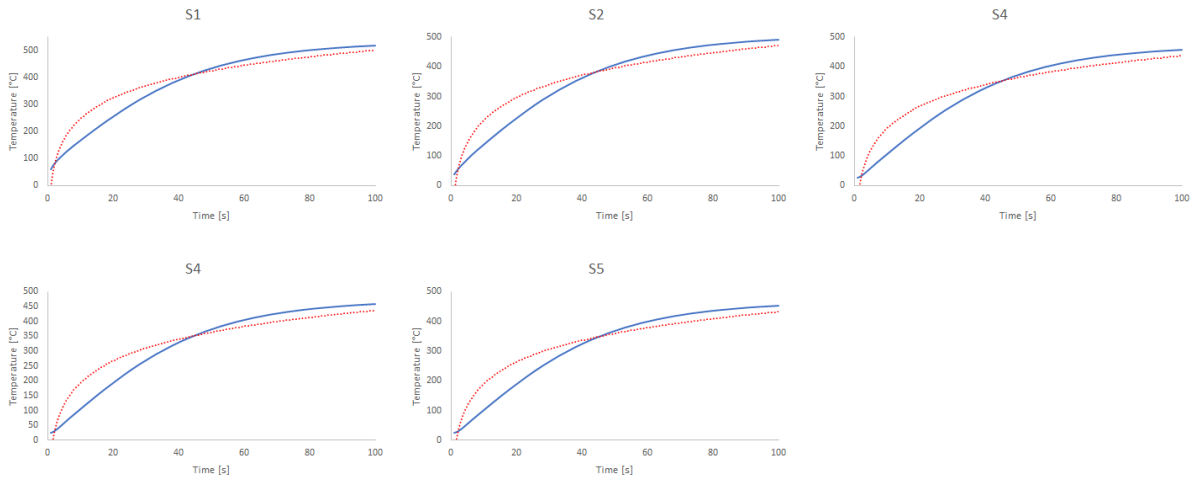


Figure 5.31: Pipe logarithmic equation fitting.

Where the blue line is the original function and the red line is the approach function, or the virtual sensors modeling by a logarithmic expression . In this case, the difference of the logarithmic function is visible in relation to the polynomial equation of 3rd degree.

At some points in the graph, the logarithmic function is too distant from the original function, in this sense there is a loss in the representativeness of the reading, causing the error at these points to be greater.

Through the logarithmic determination coefficient [R^2] Table 5.31 as possible to compare the function representation.

Table 5.31: Pipe logarithmic equation determination coefficients results [R^2].

S1	S2	S3	S4	S5
0.9528	0.9496	0.9435	0.9384	0.9365

The results of determination coefficients confirms the results in the graph, proving the loss of precision for this function.

The logarithm equations can be seen in Table 5.32.

Table 5.32: Pipe logarithmic equation resulting.

Sensors	Equations
S1	$110.03ln(x) - 5.1492$
S2	$109.11ln(x) - 30.685$
S3	$107.36ln(x) - 44.076$
S4	$105.88ln(x) - 50.294$
S5	$105.26ln(x) - 52.028$

These equations are the virtual sensors modeling by a logarithmic function. To evaluate the performance of this approach, a software that simulates the virtual sensors working was developed and the results can be seen in *Pipe Performance Tests Results*

5.4.3 Pipe Performance Tests Results

For the pipe performance tests the S3 sensor provide the input parameter through a set of 15 temperature values distributed along the curve equidistantly. The tests topology

was 1 for n, 1 real sensor as input parameter to infer 4 virtual sensors. To understand the position of the sensor and its characteristic curve, the Figure 5.32 illustrates the example.

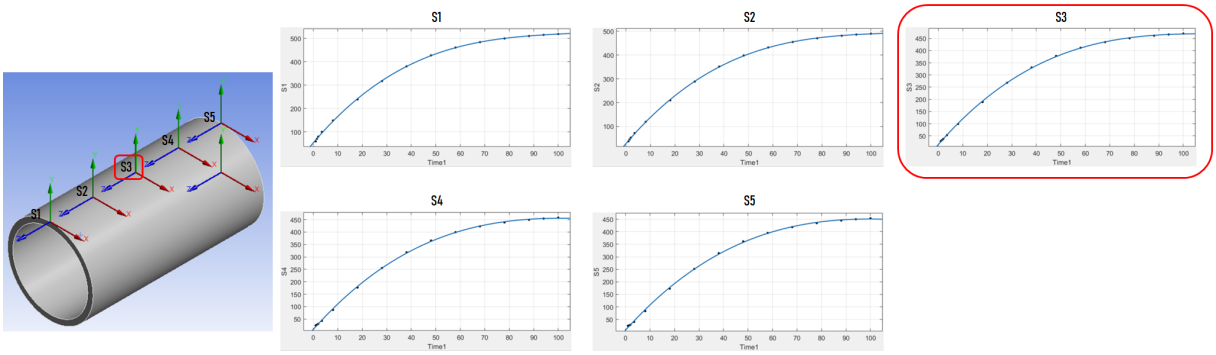


Figure 5.32: Pipe position and graphic example.

The tests for the pipe follow the same metrics as the previous case studies, the first part being the speed tests and the second the precision tests The virtual sensors speed tests is presented in the Table 5.33.

Table 5.33: Virtual sensors speed tests for the pipe.

Equation Type	Root Time [s]	Total Time [s]	Error [%]
3rd degree polynomial	0,14523	0,15395	0,0848348
Logarithmic	0,302986	0,3699	0,1398896

The curves resulting from this simulation favor adjustment through polynomial functions. For this particular case, a 3rd degree polynomial accomplished the task very well. The time to find the root is very small, as well as the total time of execution. For real-time processing, this adjustment would work very well.

For this case study, the result of the logarithmic function was a bit of a surprise. As seen in the other case studies, until then, the logarithmic function usually has the highest speed for solving the inverse function (Root Time) among all equations. However, in this model, the characteristic of the curve resulting from the CAE simulation favors the polynomial equation.

The virtual sensors percent error for the table can be seen in Table 5.34.

Table 5.34: Virtual sensors percent error for the pipe.

Equation Type	S1	S2	S4	S5
3rd degree polynomial	0,002462	0,002051	0,0017736	0,0023675
Logarithmic	0.01461	0,01429	0,013099	0,0175552

The accuracy test for the 3rd degree polynomial equation performed well when compared to the curves resulting from the CAE software. This can be verified through the set of results produced in the fitting curve stage and in the performance testing stage.

The logarithmic function has a little error when compared to the polynomial function. However, specifically for this case study, a logarithmic function is not ideal for representing the system, as it has insufficient performs than a polynomial function in all aspects.

5.5 Mold Results

In this section the main results for the mold case study will be presented.

5.5.1 Mold ANSYS Simulation Results

The mold mesh results and the quality graph can be seen in Figure 5.33.

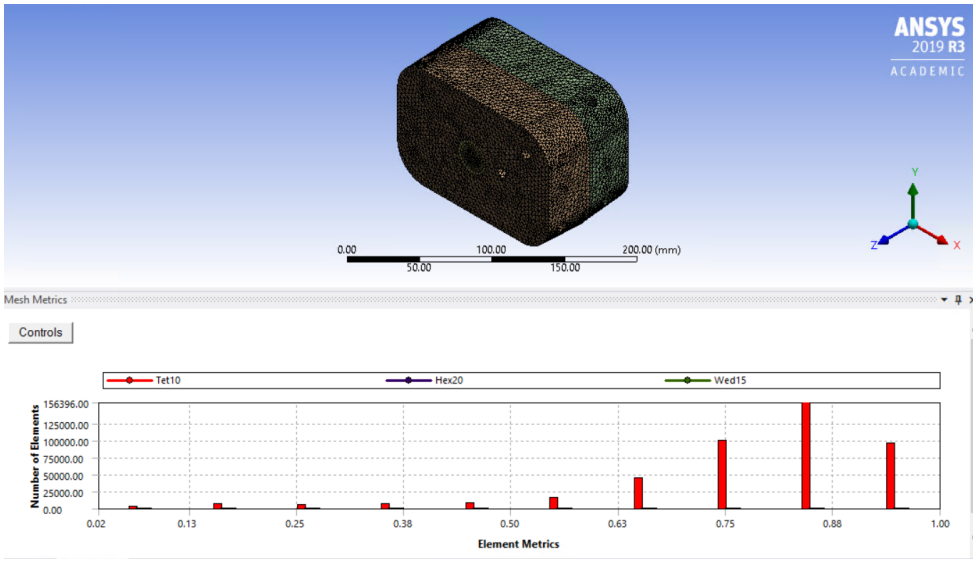


Figure 5.33: Mold mesh graphical results.

The graph shows the good performance of the mesh since most elements have quality above 0.6. It is good to remember that a dense mesh provides results closer to reality, however, computational resources are not always available to enable such simulation. In this sense, finding a balance point for the geometry of the mold was a difficult task, due to the amount of details of the geometry. Due to these characteristics, the simulation time of the thermal phenomena in the mold exceeded 4 hours. In this way, it was necessary to plan the boundary conditions very well to avoid unpleasant surprises. Another point is that the mesh is formed by elements with different geometries such as tetrahedron, hexahedron, and wedge, with the geometry of the tetrahedron being the predominant one. The use of these diverse geometries was necessary to represent the details of the part, which impacted on a high computational cost.

Others quality results of the mesh can be seen in Table 5.35.

Table 5.35: Mold mesh results.

Quality		**Statistics**	
Mesh Metric	Element Quality	Nodes	Elements
Min	1.8951e-002	657078	445681
Max	0.99995		
Average	0.78343		
Standard Deviation	0.16714		

The results in Table 5.35 shows the statistics of the elements present in the simulation. The number of the Nodes and the Elements is much bigger than the simple geometries studied. That occous because the geometry complexity.

The heat ilustration and the avegare graph results form ANSYS Transient Thermal can be seen in Figure 5.34.

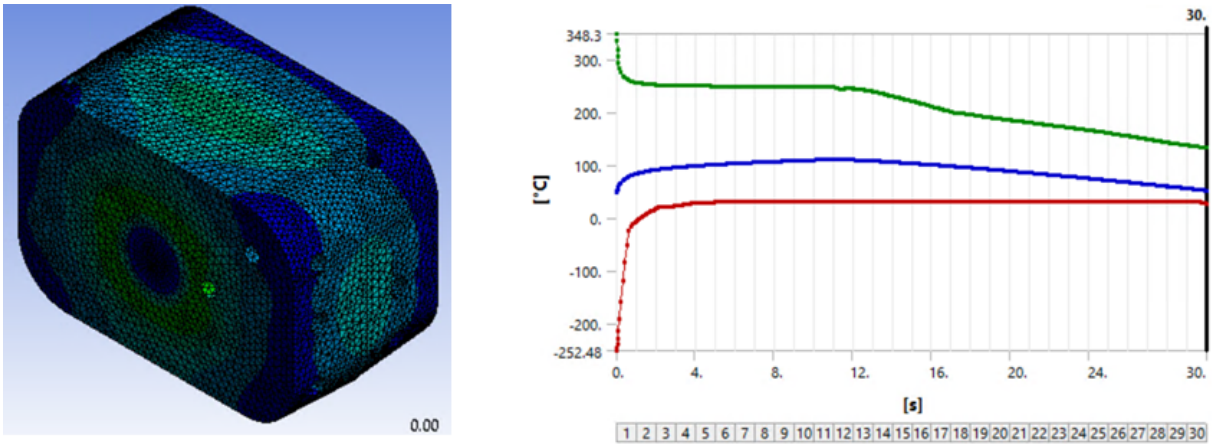


Figure 5.34: Mold average heat distribution results.

This illustration represents the heat distribution in the mold. In the Graph, the green line represents the heat source, in this case the polymer inside a mold, the blue line is the average heat and the red line is the minimum heat in the mold. As can be seen, in fists

3 seconds of the simulation, the software adjust the values to start the simulation. That adjust no cause unexpected modifications in the simulation.

The results of each sensor in the CAE simulation can be seen in Figure 5.35.

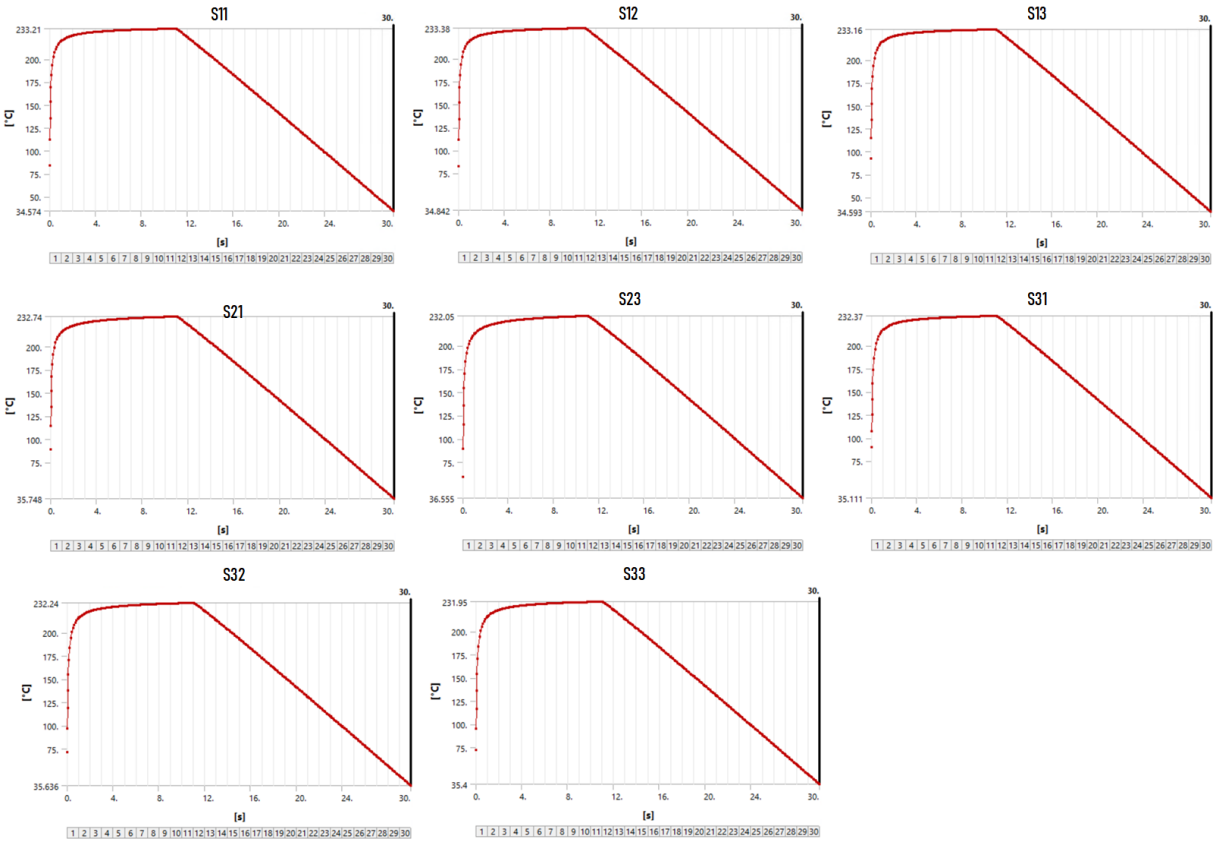


Figure 5.35: ANSYS mold sensor graph results.

The behavior of each sensor is very close, this is explained due to the geometrical positioning of each sensor, the fact that the part has a small thickness, and the injection occurs in a very small fraction of time. When looking closely, it is possible to notice differences at the beginning of each graph, as well as the slope of the curve.

Since metal is a good conductor of heat, there is a peak of heating at the beginning of the process that is registered by the sensors, this peak tends to 240 °C, so the temperature tends to come into balance. As there is forced cooling inside the mold, the temperature reduces to the extraction temperature of the part, around 30 °C.

5.5.2 Mold Fitting Curves Results

Before presenting the graphical results, to facilitate visualization of the process, the Figure 5.36 illustrates the position of the virtual sensors on the mold.

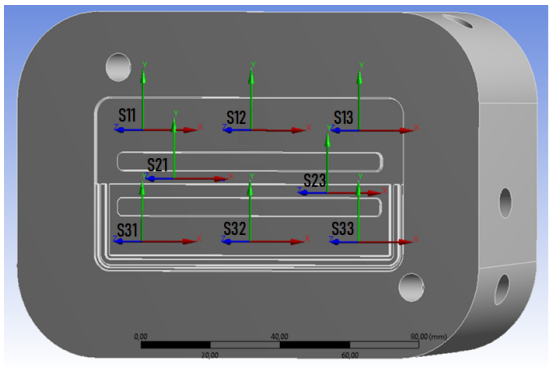


Figure 5.36: Mold sensors position.

Each curve below represents the behavior of the corresponding sensor. Figure 5.37 illustrate the fitting curves results for the mold sensors.

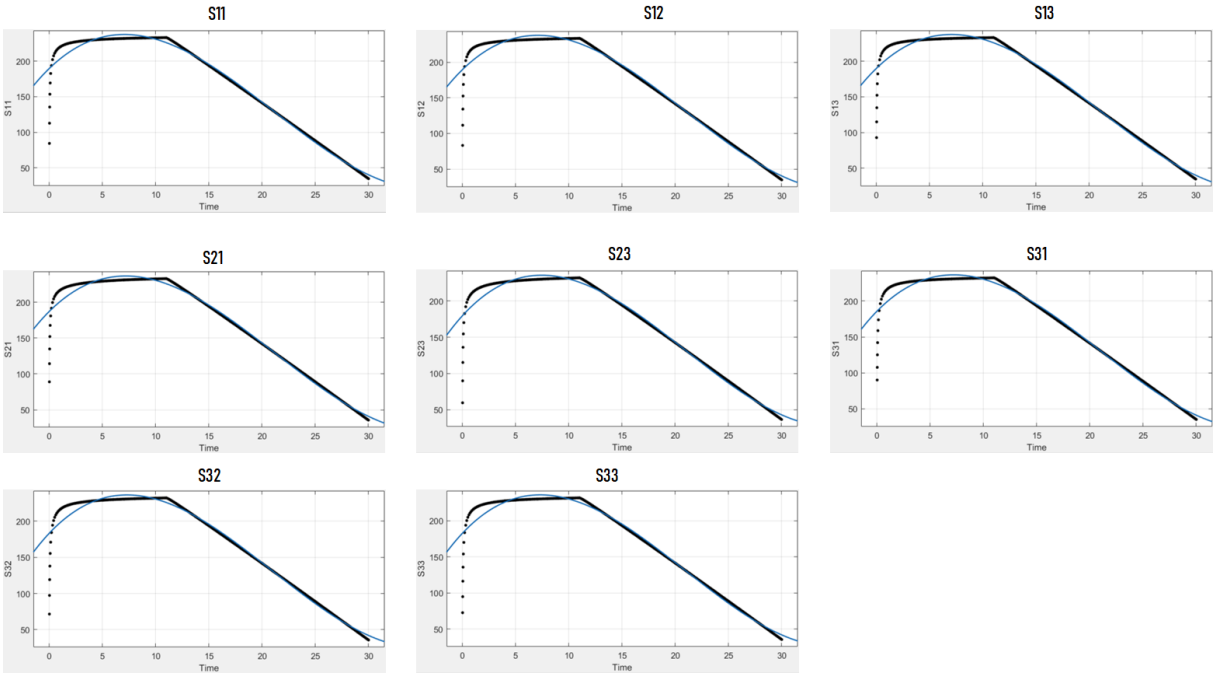


Figure 5.37: Mold 4th-degree polynomial curves fitting results.

The adjustment curve process tested polynomial equations from 2nd to 8th degree,

with the objective of finding an efficient equation, from the point of view of computational cost and precision. The chosen equation was the 4th-degree polynomial, that equation represents a good balance between performance and precision.

The determination factor [R^2] of 4th-degree polynomial can be seen in Table 5.36.

Table 5.36: Mold 4th-degree polynomials determination coefficients.

S11	S12	S13	S21	S23	S31	S32	S33
0.9814	0.981	0.9826	0.9828	0.9754	0.981	0.9772	0.9768

The values of the determination factor show how much the fitted equations represent the original curve. The result is good since the representativeness values are above 0.97.

The equations that describe the graphs can be seen in the Table 5.37.

Table 5.37: Mold 4th-degree polynomial equation.

Sensors	Equations
S11	$(-0.0004145) * x^4 + 0.0441 * x^3 + (-1.702) * x^2 + 17.56 * x + 185.5$
S12	$(-0.0004212) * x^4 + 0.04454 * x^3 + (-1.713) * x^2 + 17.67 * x + 185.2$
S13	$(-0.0004024) * x^4 + 0.04329 * x^3 + (-1.684) * x^2 + 17.4 * x + 185.9$
S21	$(-0.000385) * x^4 + 0.0422 * x^3 + (-1.668) * x^2 + 17.55 * x + 183.1$
S23	$(-0.0005876) * x^4 + 0.05583 * x^3 + (-1.982) * x^2 + 20.48 * x + 173.8$
S31	$(-0.0004897) * x^4 + 0.04915 * x^3 + (-1.819) * x^2 + 18.7 * x + 181$
S32	$(-0.0005641) * x^4 + 0.0541 * x^3 + (-1.932) * x^2 + 19.75 * x + 177.6$
S33	$(-0.0005812) * x^4 + 0.05526 * x^3 + (-1.958) * x^2 + 19.96 * x + 177$

These equations are the virtual sensors that represent the temperature behavior in the selected points developed for the mold, and each sensor are modeled by a 4th-degree polynomial expression. The process that determined the characteristic of the response curves of the sensors guarantees the precision according to the boundary conditions defined in the physical model configured in the CAE software. For this case study, the best approximations occurs by a 4th-degree polynomial equation. This technique has great robustness since the time used represents the real-time of the process.

5.5.3 Mold Performance Tests Results

This test phase, as already mentioned in other case studies, serves to evaluate the speed of resolution of the expressions and their respective accuracy. For that, the results are presented in two parts, the first referring to the speed tests and the second referring to the precision tests. For this geometry, performance tests were performed with topology 1 for n and n for n.

For the n for n topology, the S21 sensor provided the temperature input parameter, simulating a real sensor. Due to this fact, the S21 sensor does not present statistical results in this analysis. Twenty temperature points were selected along the curve distributed evenly.

To facilitate understanding, the Figure 5.38 shows the location of the real sensor and its respective curve. The other curves are the behavior of the virtual sensors in response to the input of the real sensor.

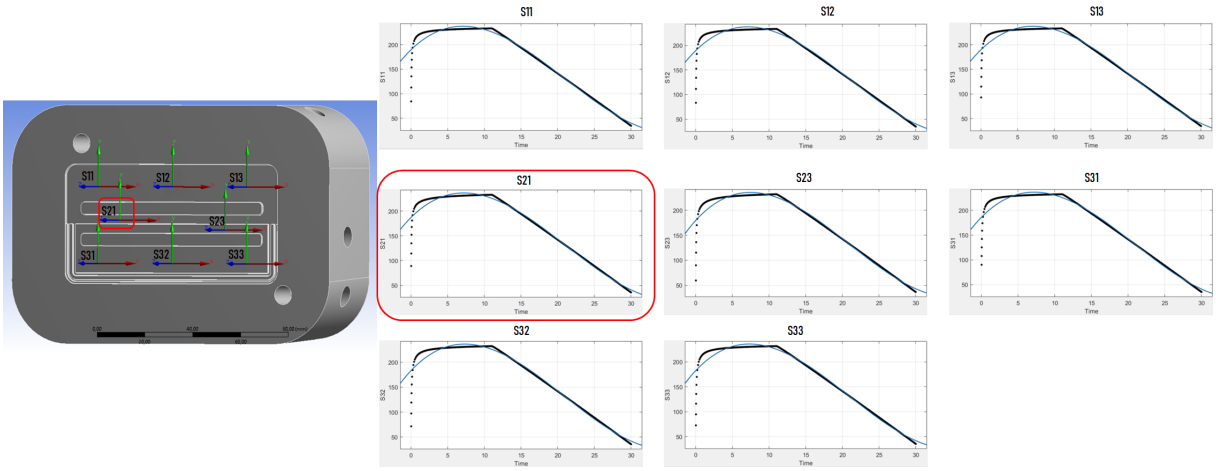


Figure 5.38: Mold input sensor position and characteristic curve.

For this test, the times to obtain the roots of the inverse function (time in seconds), the total code execution time (input + inverse function + calculation of the virtual sensors), and the error between the time value calculated using the inverse function and the respective time in the fitting curve table. The results obtained can be seen in the Table 5.38.

Table 5.38: Virtual sensor topology 1 to n speed tests for the mold.

Root Time [s]	Total Time [s]	Error [%]
0,220029	0,231517	0,001835

As important as the measurement of the total execution time, the result reveals a match point in the analysis, the resolution time of the inverse function. The resolution of the inverse function is the moment when the code demands more computational resources, in this way, the speed of resolution of the function is directly linked to the equation degree, the higher is the degree, more computational resources are requests, spending more processing time.

Through the use of 4th-degree polynomial functions, the result in velocity parameters is close to the value of the sampling rate of the real sensor, which is 120 ms.

Regarding the real sensor, it is important to realize that exist other times involved, like conditioning and processing the signal, so for the virtual sensor, a 200 ms sample rate is not bad.

From the point of view of the code, considering that the operation of solving the inverse function occurs in real-time, this means that approximately every two cycles of the real sensor, a cycle of the virtual sensor occurs. Another point to be noted is that the total time to solve the equations for all 7 sensors takes around 11 ms, which is a very good result.

The virtual sensors values accuracy was verified through the known fitting curves table. The S12 sensor will not be calculated in the Table 5.39 because this sensor was the input parameter of the analysis. For the others sensors, the error percentage was calculated. The sensors error percentage results can be seen in Table 5.39.

Table 5.39: Virtual sensor topology 1 for n sensors percent error for the mold.

S11	S12	S13	S23	S31	S32	S33
0,00438	0,00131	0,00321	0,00027	0,00352	0,00135	0,0021

Despite having a 3-second range at the beginning of the heating cycle that is not

represented by the equation, the results found from the point of view of precision are very good. It is possible to notice that the difference in the calculated values and the values of the CAE simulation does not reach 0.05%. From that perspective, the results are promising.

For a topology n for n, the procedure for performing the speed and accuracy tests is identical to the format described in the previous topology. The sensors that provide the input temperature parameter for this topology are S21, and S23, their positions and respective characteristic curves can be seen in the Figure 5.39, the response behavior of the virtual sensors are also described in the graphs presented.

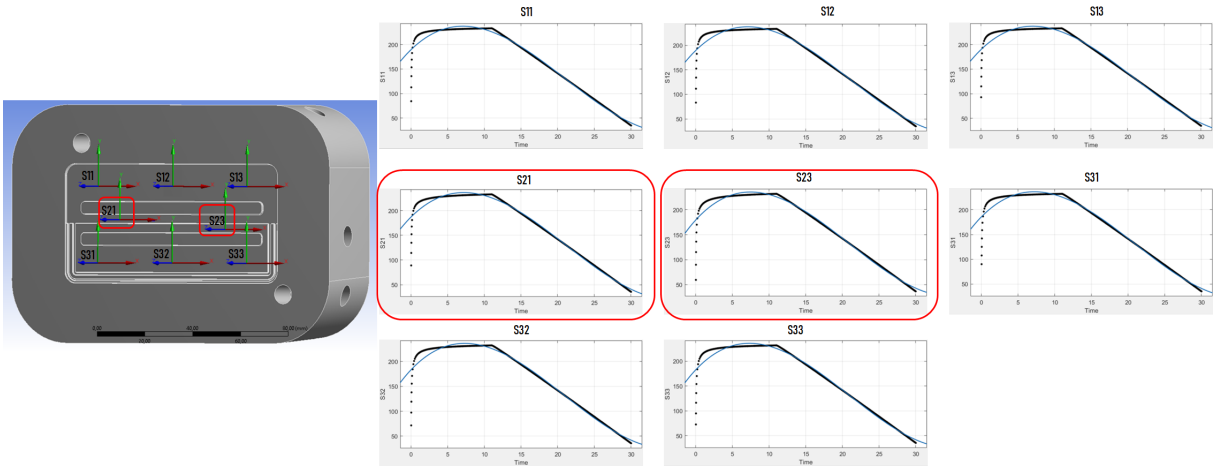


Figure 5.39: Mold input sensor position and characteristic curve n for n topology.

The results of the speed test can be seen in the Table 5.40.

Table 5.40: Virtual sensor topology n for n speed tests for the mold.

Root Time [s]	Total Time [s]	Error [%]
0,46435	0,47355	0,02068

It is possible to notice that times have increased in relation to topology 1 for n. This result is expected since in the tested algorithm two inverse functions are operating at the same time. The proposed methodology for multiple real input sensors provides for multiple resolutions of inverse functions. In this sense, it is noticed that when increasing

the number of inverse equations, the processing time increases proportionally. It can be said that for this simulation there is an increase of about 230 ms, for each inverse function not compared to any code.

At the real-time processing level, the measured time corresponds to four times the sampling rate of the real sensor. It is not possible to state that for the situation with two real sensors as input, the processing is impaired, but it is easy to see that the greater the number of real input senors, the greater the delay concerning the sampling rate of the real sensor.

One way to get around the delay is to perform a pre-processing, store the data, and use the calculations already solved at run time.

The sensors S21 and S23 are not listed in the Table because they were the input parameters of the system. The percent error value of each sensor can be seen in the Table 5.41.

Table 5.41: Virtual sensor topology n for n sensors percent error for the mold.

S11	S12	S13	S31	S32	S33
0,00357	0,00052	0,00238	0,00288	0,00083	0,00158

From the point of view of the error found in the virtual sensors when compared to the data of the CAE software, it is noticed that a rate is less than 0.004 %, in for the set of values used as the system input. As in simulation 1 for n, the virtual sensor equations do not represent the initial 3 s of the simulation. However, for the remaining 27 s of the process, the equations have a very good representation.

Chapter 6

Conclusion and Future Work

Even though the thin surface sensors are a technological revolution from the point of view of data acquisition, they still suffer restrictions of use regarding their positioning and quantity of sensors in the injection chamber that are imposed by the mold geometry. Thus, it is impossible to know the thermal behavior in real-time in some parts of the mold cavity.

To overcome this restriction, virtual sensors were implemented using a technique that extracts the thermal behavior of the mold in three stages, the first being the use of ANSYS software to simulate the thermal model in any point of geometry, the second, the use of Matlab and Excel software to find mathematical expressions that represent the thermal behavior of these points, and the third, a software developed in Python that uses these mathematical expressions to simulate the use of virtual sensors.

The development of virtual sensors was conceived through 5 case studies, initially, simple geometries were studied, and finally, this knowledge was used for the implementation of virtual sensors in the injection mold.

Using the proposed method, virtual temperature sensors for simple geometries and mold geometry were implemented.

As explained earlier, the results found in simple geometries vary according to the boundary conditions considered. There is an evolution in the understanding of the CAE software simulation, from the plate in ideal conditions and heating by conduction, passing

through the plate with two heat sources by conduction, later the addition of losses in the study of the cylinder geometry, as well as the change an absolute temperature value for a heating rate and with additional losses in the pipe simulation. This understanding made it possible to transport the phenomena involved in the injection molding process.

A very important consideration has always been the balance between processing speed and precision. In this sense, the polynomial equations of degree 8, for the majority of the cases, was the one that presented more precision, and according to the analogy described above, it was the one that most consumed computational resources when performing the inverse function. Regarding the calculation of the direct function, concerning speed, all functions (polynomial and logarithmic) showed equivalent performance.

With regard to the CAE software, ANSYS is a powerful simulation tool. However for the physical behavior of the geometry to be consistent with the expected results, it is necessary to carry out the configurations with attention and critical reasoning. There was a difficulty in finding a balance point in the simulation, with regard to the relationship between mesh refinement x computational cost x result accuracy.

In the case of the mold, at the level of CAE simulation, it was realized that it is necessary to have a solid theoretical background to be able to describe the physical phenomena in the simulation. With regard to geometry, a particular challenge was to simplify the analysis surfaces, as the original geometry contains all the parts that make an injection mold.

Another point to consider is the time required to process the simulation of the mold in the CAE software. The time involved in each configuration attempt was over 4 hours. This made the process more difficult, since for each configured adjustment, its result would only be known after the geometry was fully processed. This high processing time is directly linked to the mesh density due to the complexity of the analyzed geometry. Even with various simplifiing precautions, the amount of details was still large, implying in a high computational cost.,

The process of adjusting the curves and obtaining the equations was facilitated through the use of Matlab and Excel. What was perceived during the fitting curves process, and

that the equations do not always represent the phenomenon in full. Errors are present in the fitting curves process, it is relative to say whether an error is large or small, as it depends on the region analyzed. Another point to consider, it is relative to say that a curve is well represented by analyzing only the factor of R^2, since it depends on the point to be analyzed.

The code implementation for simple geometries focused on topology 1 for n and the exploration of the different degrees of polynomials (3th-degree until 8th-degree functions) and logarithmic function. Through the code that some functions have performance limitations in their extreme values, presenting higher error rates. For the implementation of the code, a key point is a time that the inverse function takes to find the system time. In this sense, logarithmic functions took advantage of polynomial functions for simple geometries, except for the case study of the pipe that the nature of the system curve was naturally best represented by a 3th-degree polynomial. Regarding the error, most of the studied functions had low indices when compared to the CAE simulations curves, except for specific cases, and as already mentioned, at the ends of the functions the error tends to be higher.

For the mold, topologies 1 for n and n for n were tested. An interesting fact is that the characteristic curves of the molding process favored the implementation of the n to n methodology, as they present little variation between them. Topology 1 for n showed excellent results both from the point of view of speed and precision. In this topology, no performance problems were found regarding the resolution of the inverse function, since the polynomials used in this simulation were 4th-degree. The values obtained through the virtual sensors showed very low errors when comparing the CAE software curves.

For the development of n to n test code, the inverse function is a point to be improved, since the processing time of the inverse functions corresponds to more than 90% of the total code execution time. The consequence of the implemented architecture is that as more real sensors need to be processed at the input of the system, the cost of processing increases linearly.

Due to the characteristic curves of the process, for the mold, only polynomial functions

were able to describe the process. In this sense, to maintain a balanced ratio between processing cost and error in the inference, the 4th-degree polynomial was chosen. From the point of view of precision, for the mold, the results obtained by the virtual sensors showed a very low error when compared with the CAE simulation However, it is known that the code has limitations that need to be improved, such as not representing the system in the first 3 seconds of the process.

For future work, there are some interesting ways of suggested improvements and ways of extrapolating this project, as can be seen below:

- Perform physical implementation of the method, starting with tests on simple surfaces to validate the method, and after in injection mold;
- Develop an embedded code to run on the developed hardware in On-Surf project;
- Combine the technique of virtual sensors with a multi-agent system, where each sensor behaves as an agent and the virtual sensors interact with the real sensors in the system;
- Perform analysis from fluidodynamics, using the same CAE tool or other software available on the market;
- Develop the modeling of virtual pressure sensors;
- Use the developed method combined intelligent artificial or machine learning techniques, like back-box and gray-box models;
- Use a thermal imaging system with the virtual sensors method to infers temperature values.

This project demonstrated how simulation software can be used to provide data for the implementation of virtual temperature sensors.

Bibliography

[1] *Cefamol report 2020*, 2020. [Online]. Available: `http://onsurf.teandm.pt/index.html` (visited on 2020).

[2] *Projeto on-surf.* [Online]. Available: `http://onsurf.teandm.pt/index.html` (visited on 2020).

[3] M. A. Bussmann, "Condicionamento de sinal para sistema de aquisição de dados em superfícies metálicas finas," Master's thesis, Dissertação apresentada à Escola Superior de Tecnologia e Gestão (ESTiG) de Bragança no âmbito do programa de dupla diplomação com a Universidade Tecnológica Federal do Paraná (UTFPR) para obtenção do título de Mestre em Engenharia Industrial, ramo eletrotécnica., 2020.

[4] N. Nonato, "Data acquisition system for fine surface process monitoring," Master's thesis, Dissertation presented to the School of Technology and Management of Bragança to obtain the Masters Degree in Industrial Engineering in Double Degree program with UTFPR, 2020.

[5] J. V. G. Neto, "Development of a sensory data concentrator for network broadcasting," Master's thesis, Dissertation presented to the School of Technology and Management of Polytechnic Institute of Bragança to the Fulfillment of the Requirements for the Master of Science Degree in Industrial Engineering (Electrical Engineering branch), in the scope of Double Degree with Federal University of Technology - Paraná., 2020.

[6] T. Ageyeva, S. Horváth, and J. G. Kovács, "In-mold sensors for injection molding: On the way to industry 4.0," *Sensors*, vol. 19, no. 16, p. 3551, 2019.

[7] F. A. Lotufo and C. Garcia, "Sensores virtuais ou soft sensors: Uma introdução," in *Proceedings of the 7th Brazilian Conference on Dynamics, Control and Applications, Presidente Prudente*, 2008, pp. 1–9.

[8] L. Fortuna, S. Graziani, A. Rizzo, and M. G. Xibilia, *Soft sensors for monitoring and control of industrial processes*. Springer Science & Business Media, 2007.

[9] "Moldes portugueses uma industria reconhecida mundialmente," *Portugalglobal*, no. 117, p. 6, 2019.

[10] P. Nagorny, M. Pillet, E. Pairel, R. Le Goff, J. Loureaux, M. Wali, and P. Kiener, "Quality prediction in injection molding," in *2017 IEEE International Conference on Computational Intelligence and Virtual Environments for Measurement Systems and Applications (CIVEMSA)*, IEEE, 2017, pp. 141–146.

[11] J. Harada, *Moldes para injeção de termoplásticos: projetos e princıépios básicos*. Artliber, 2004.

[12] *Custompart.net - injection molding*. [Online]. Available: `http://www.custompartnet.com/wu/InjectionMolding` (visited on 2020).

[13] S. Karthikeyan and S. Jayabal, "S. kalyanasundaram, c boopathi,"influence of cooling channel position and form on polymer solidification and temperature in injection molding die"," *International Journal For Research In Applied Science & Engineering Technology (IJRASET)*, vol. 3,

[14] *Tudo sobre plasticos - o ciclo de injeção*. [Online]. Available: `https://www.tudosobreplasticos.com/processo/cicloinjecao.asp` (visited on 2020).

[15] H. Karbasi and H. Reiser, "Smart mold: Real-time in-cavity data acquisition," in *First Annual Technical Showcase & Third Annual Workshop, Canada*, Citeseer, 2006.

[16] L. C. Martin, J. D. Wrbanek, and G. C. Fralick, "Thin film sensors for surface measurements [in aerospace simulation facilities]," in *ICIASF 2001 Record, 19th International Congress on Instrumentation in Aerospace Simulation Facilities (Cat. No. 01CH37215)*, IEEE, 2001, pp. 196–203.

[17] D. Thomazini, *Sensores industriais: fundamentos e aplicações*. Saraiva Educação SA, 2005.

[18] M. (Max), *Sensor fundamentals*. [Online]. Available: `https://maxembedded.wordpress.com/2011/06/18/sensor-fundamentals/` (visited on 2020).

[19] G. Cosoli, P. Chiariotti, M. Martarelli, S. Foglia, M. Parrini, and E. P. Tomasini, "Development of a soft sensor for indirect temperature measurement in a coffee machine," *IEEE Transactions on Instrumentation and Measurement*, vol. 69, no. 5, pp. 2164–2171, 2019.

[20] F. Amara, K. Agbossou, A. Cardenas, Y. Dubé, S. Kelouwani, *et al.*, "Comparison and simulation of building thermal models for effective energy management," *Smart Grid and renewable energy*, vol. 6, no. 04, p. 95, 2015.

[21] L. A. Aguirre, *Introdução à Identificação de Sistemas–Técnicas Lineares e Não-Lineares Aplicadas a Sistemas Reais*. Editora UFMG.

[22] N. A. B. Monteiro, J. J. da Silva, and J. S. da Rocha Neto, "Uso do conceito de sensor virtual aplicado ao monitoramento e controle de temperatura de uma estufa térmica,"

[23] P. R. P. Susana, "Sensores virtuais usando aprendizagem online para processos industriais," Master's thesis, 2015.

[24] F. M. Soares and R. C. Oliveira, "Inferência de temperatura de fornos de redução de alumıénio primário através de sensores virtuais," in *IX Congresso Brasileiro de Redes Neurais, Ouro Preto-MG*, 2010, pp. 25–28.

[25] Y. Wu and X. Luo, "A design of soft sensor based on data fusion," in *2009 International Conference on Information Engineering and Computer Science*, IEEE, 2009, pp. 1–4.

[26] A. Balakrishna, R. S. Babu, D. N. Rao, D. R. Raju, and S. Kolli, "Integration of cad/cam/cae in product development system using step/xml," *Concurrent Engineering*, vol. 14, no. 2, pp. 121–128, 2006.

[27] ESSS, *Engenharia assistida por computador: O que é e como funciona?* [Online]. Available: `https://www.esss.co/blog/engenharia-assistida-por-computador-o-que-e-e-como-funciona/` (visited on 2020).

[28] S. Moaveni, *Finite element analysis theory and application with ANSYS, 3/e*. Pearson Education India, 2011.

[29] *Finite element mesh refinement*. [Online]. Available: `https://uk.comsol.com/multiphysics/mesh-refinement?parent=physics-pdes-numerical-042-32` (visited on 2020).

[30] T. G. Blog, *Finite element method : Introduction and steps of finite element analysis.* [Online]. Available: `https://www.geniuserc.com/finite-element-method-introduction-and-steps-of-finite-element-analysis/` (visited on 2020).

[31] K. Himasekhar, J. Lottey, and K. Wang, "Cae of mold cooling in injection molding using a three-dimensional numerical simulation," 1992.

[32] G. V. Salmoria, C. H. Ahrens, F. A. Villamizar, and A. d. C. Sabino Netto, "Influência do desempenho térmico de moldes fabricados com compósito epóxi/alumiénio nas propriedades de pp moldado por injeção," *Polıémeros*, vol. 18, no. 3, pp. 262–269, 2008.

[33] M. J. E. Raposo, "Análise do efeito da temperatura do sistema de injeção nas peças plásticas obtidas," PhD thesis, 2018.

[34] R. Stout, P. D. Billings, and P. Semiconductor, "Accuracy and time resolution in thermal transient finite element analysis," in *ANSYS^{TM} users conference*, 2002.

[35] F. P. Incropera, D. P. Dewitt, and T. L. Bergman, *Fundamentos de Transferência de Calor E de Massa* . Grupo Gen-LTC, 2000.

[36] *Encyclopaedia britannica, inc - heat transfer*. [Online]. Available: `https://www.britannica.com/science/heat-transfer` (visited on 2020).

[37] *Heat transfer*. [Online]. Available: `http://hyperphysics.phy-astr.gsu.edu/hbase/thermo/heatra.html#c1l` (visited on 2020).

[38] J. R. S. Moreira, *PME -2361 Processos de Transferência de Calor*. 2014.

[39] *Física - condução térmica*. [Online]. Available: `https://brasilescola.uol.com.br/fisica/conducao-termica.htm` (visited on 2020).

[40] D. V. Widder, *The heat equation*. Academic Press, 1976, vol. 67.

[41] N. Flyer and B. Fornberg, "Accurate numerical resolution of transients in initial-boundary value problems for the heat equation," *Journal of Computational Physics*, vol. 184, no. 2, pp. 526–539, 2003.

Appendix A

Original Project Proposal

Proposta de Dissertação / Projeto
Mestrado em Engenharia Industrial
2019/2020

Título Provisório: Desenvolvimento de sensores virtuais em superfícies finas para manutenção preditiva

---

Aluno de mestrado: Fábio Augusto Amaral

Orientador: Paulo Leitão

Coorientador: Nelson Rodrigues

Coorientador (UTFPR): Roberto Ribeiro Neli

Principais objetivos a atingir
Pretende-se iniciar uma nova linha de estudos junto ao projeto ON-Surf sobre o desenvolvimento de sensores virtuais, onde serão estudados sensores de temperatura e pressão. Para tanto, modelos serão desenvolvidos e analisados com o auxílio do software ANSYS e análises matemáticas. Deseja-se estudar o comportamento de sólidos metálicos de diferentes formas e volumes, expostos a condições de aquecimento e pressão. A partir desses modelos serão desenvolvidas formas de estabelecer fatores de correlação entre os sensores físicos e os virtuais. As informações dos dados correlacionados serão utilizadas para realizar a implementação via software dos sensores virtuais. O desenvolvimento de uma interface gráfica poderá ser feito para que o entendimento da leitura dos sensores virtuais seja simplificado.

Resultados esperados
Pretende-se com esta proposta compreender o mecanismo por trás do funcionamento dos sensores virtuais, e com isso fazer uso da técnica para obter mais pontos de medidas dos objetos analisados. Espera-se que esta abordagem seja um ponto de partida para a integração do sistema de sensores virtuais ao hardware já desenvolvido em outras três partes do projeto On-Surf. Pretende-se também com essa linha de pesquisa, proporcionar trabalhos futuros no campo de sensores virtuais, como o estudo aplicado aos moldes das máquinas de injeção de plástico e máquinas de estampagem metálica.

Caracterização do trabalho
No domínio da fabricação, a estampagem metálica e a injeção plástica são alguns procedimentos essenciais, de forma que as empresas que atuam nesta área necessitam otimizar estes processos para ganharem uma vantagem competitiva. Neste sentido, o projeto On-Surf pretende desenvolver processos de modificação de superfície que promovam soluções avançadas e desenvolvimento deste tipo de superfícies com propriedades sensoriais. Devido a limitações físicas do próprio ambiente onde não há a possibilidade da implementação de um sensor físico, seja por dificuldades no design da máquina, pelo meio ser corrosivo ou agressivo, é necessária uma alternativa para conseguir colecionar toda a informação necessária para tomar melhores decisões. Este trabalho propõe o estudo de técnicas baseadas em sensores virtuais para melhor monitorizar o processo, bem como estado real dos recursos, sejam as máquinas ou os seus processos, em vez de algum valor teórico de expectativa de vida.

Calendarização das fases do trabalho:
O desenvolvimento da presente proposta de trabalho será realizado através da execução das seguintes etapas:

1. Familiarização do hardware desenvolvido no projeto On-Surf (M1-M2);
2. Estudo de estratégias de implementação de sensores virtuais e escolha de método de implementação (M3-M4);
3. Desenvolvimento do(s) modelo(s) escolhido(s) para implementação de sensores virtuais (M5-M6);
4. Estudo de interface para representação gráfica da superfície analisada (M7-M8);
5. Escrita da dissertação e defesa final do trabalho (M4-M9)

Appendix B

B.1 Mold Steel Datasheet

SIMOLD 2738 Steel

Designation by Standards

Brand Name	Ravne	Mat. No.	DIN	EN	AISI/SAE
SIMOLD 2738	UTOPNIN	1.2738	40CrMnNiMo8-6-4	-	P20 Mod.

Chemical Composition (in weight %)

C	Si	Mn	Cr	Mo	Ni	V	W	Others
0.40	0.30	1.45	1.95	0.20	1.05	-	-	-

Description
This steel is in the category generally labeled as Mold Steels. Nickel and chromium are the alloying elements for hardness and toughness. Cold work tool steel with good machinability, excellent polishability, suitable for texturing. Improved through hardenability comapred to Mat.No. 1.2311.

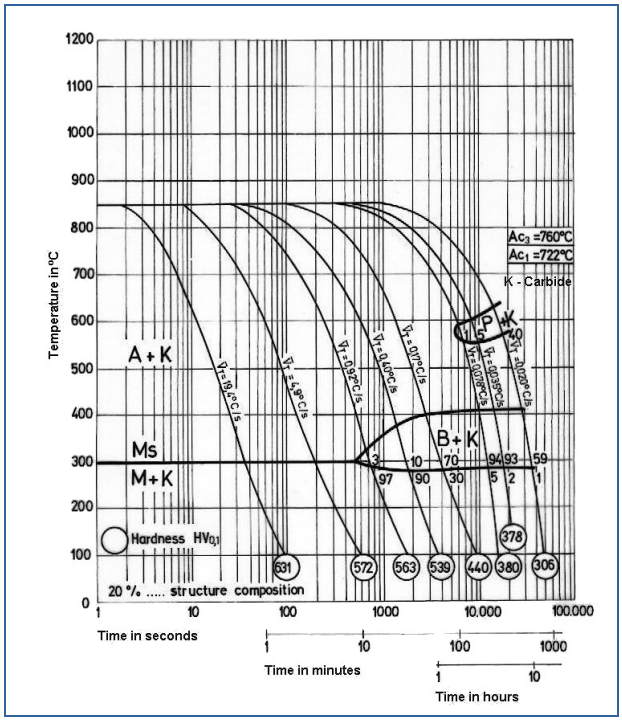
Applications
Typically used for relatively low temperature applications such as die casting dies and injection molds, synthetic plastic moulds dies, for large moulds.

Physical properties (average values) at ambient temperature
Modulus of elasticity [10^3 x N/mm^2]: 205
Density [g/cm^3]: 7.80
Thermal conductivity [W/m.K]: 29.0
Specific heat capacity[J/g.K]: 0.46

Coefficient of Linear Thermal Expansion 10^{-6} $^oC^{-1}$

20-100^{o}C	20-200^{o}C	20-300^{o}C	20-400^{o}C	20-500^{o}C	20-600^{o}C	20-700^{o}C
11.7	12.3	13.0	13.3	13.7	13.7	14.0

Continuous Cooling Transformation (CCT) Diagram



Soft Annealing

Heat to 710-740^{O}C, cool slowly. This will produce a maximum Brinell hardness of 235.

Stress Relieving

Stress relieving to remove machining stresses should be carried out by heating to 650^{O}C, holding for one hour at heat, followed by air cooling. This operation is performed to reduce distortion during heat treatment.

Hardening

Harden from a temperature of 840-880^{O}C followed by oil, air ot warm bath (180-220^{O}C) quenching. Hardness after quenching is 52 HRC.

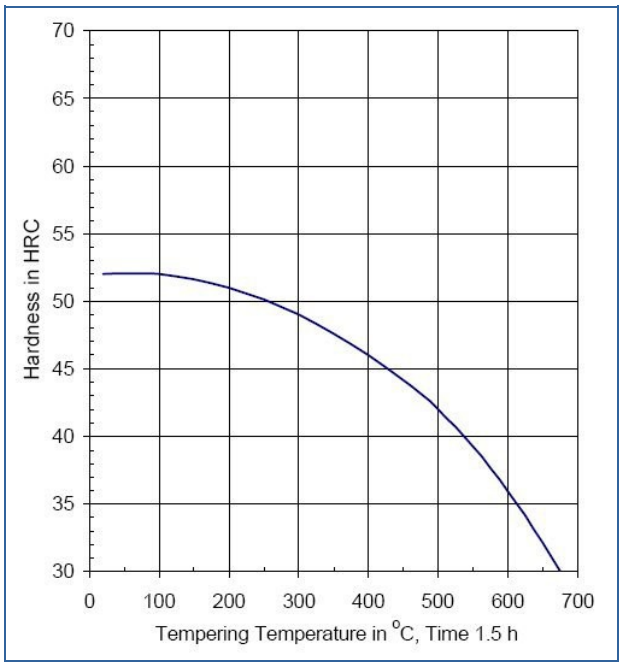
Tempering

Tempering temperature: See the data bellow.

Tempering Temperature (^{O}C) vs. Hardness (HRC) vs. Tensile Stregth (N/mm^{2})

100^{O}C	200^{O}C	300^{O}C	400^{O}C	500^{O}C	600^{O}C	700^{O}C
52	51	49	46	42	36	28
1790	1730	1620	1480	1330	1140	920

Tempering Diagram



Forging

Hot forming temperature: 1093-898°C.

Machinability

Machinability is relatively good at about 80% that of the W group water hardening steels.

Welding

This alloy is weldable by conventional methods. Contact the alloy supplier for details and weld procedures.

Forms manufactured: Please see the Dimensional Sales Program.

Disclaimer

The information and data presented herein are typical or average values and are not a guarantee of maximum or minimum values. Applications specifically suggested for material described herein are made solely for the purpose of illustration to enable the reader to make his own evaluation and are not intended as warranties, either express or implied, of fitness for these or other purposes. There is no representation that the recipient of this literature will receive updated editions as the become available.

B.2 Plate 1G Code

```
#CHAPA - Teste com Função Polinomial grau 8
#Sensor Virtual com 1 Gradiente de Temperatura, com Polinômio de Grau 8, 1
Sensor Entrada, 4 SV, Tempo Func. Pol. Inv
#Versão 1.0
from sympy import *
x=Symbol('x', real=True)
#Mede o tempo de execução
import time
start_time = time.time()
#Simula imput de temperatura do sensor real
T=199.760
print('Temperatura Real:',T,'ºC')
print('---------------------------------------\n')
#Encontra o tempo através da equação do sensor Real
v=solve((-2.035e-15)*x**8+(2.162e-12)*x**7+(-9.616e-10)*x**6+(2.329e-
07)*x**5+(-3.362e-05)*x**4+(0.003013)*x**3+(-0.1749)*x**2+(7.1)*x+14.61-T, x)
#v=solve((-8.472e-15)*x**8+(8.794e-12)*x**7+(-3.792e-09)*x**6+(8.79e-
07)*x**5+(-0.0001185)*x**4+(0.009418)*x**3+(-0.4301)*x**2+(10.92)*x+46.15-
T, x)
print('Raizes:\n',v)
x=v[0]
print('Raiz 1 selecionada\n',x)
print('---------------------------------------\n')
#Informa o tempo usado para encontrar a raiz (função inversa, encontra o tempo
em segundos do sistema)
print("---Tempo Raiz --- %s segundos ---" % (time.time() - start_time))
print('---------------------------------------\n')
#Calcula os sensores Virtuais a partir do tempo encontrado na equação do
sensor Real
S1=(-8.472e-15)*x**8+(8.794e-12)*x**7+(-3.792e-09)*x**6+(8.79e-07)*x**5+(-
0.0001185)*x**4+(0.009418)*x**3+(-0.4301)*x**2+(10.92)*x+46.15
print('Sensor virtual 1:',S1)
S2=(-8.473e-15)*x**8+(8.795e-12)*x**7+(-3.792e-09)*x**6+(8.79e-07)*x**5+(-
0.0001185)*x**4+(0.009418)*x**3+(-0.4301)*x**2+(10.92)*x+46.15
```

```
print('Sensor virtual 2:', S2)
S3=(2.744e-15)*x**8+(-2.836e-12)*x**7+(1.212e-09)*x**6+(-2.756e-
07)*x**5+(3.545e-05)*x**4+(-0.002454)*x**3+(0.06152)*x**2+(2.647)*x+14.72
print('Sensor virtual 3:', S3)
S4=(2.745e-15)*x**8+(-2.836e-12)*x**7+(1.212e-09)*x**6+(-2.756e-
07)*x**5+(3.545e-05)*x**4+(-0.002454)*x**3+(0.06153)*x**2+(2.647)*x+14.72
print('Sensor virtual 4:', S4)
print('-------------------------------------\n')
#v=solve((-2.035e-15)*x**8+(2.162e-12)*x**7+(-9.616e-10)*x**6+(2.329e-
07)*x**5+(-3.362e-05)*x**4+(0.003013)*x**3+(-0.1749)*x**2+(7.1)*x+14.61-T, x)
#print('Sensor 5:', SV)
print("---Tempo final %s segundos ---" % (time.time() - start_time))
#########################################################################

#CHAPA - Teste com Função Polinomial grau 6
#Sensor Virtual com 1 Gradiente de Temperatura, com Polinômio de Grau 6, 1
Sensor Entrada, 4 SV, Tempo Func. Pol. Inv
#Versão 1.0
from sympy import *
x=Symbol('x', real=True)
import time
start_time = time.time()
#Imput de temperatura do sensor real
T=199.760
print('Temperatura Real:',T,'ºC')
print('-------------------------------------\n')
#Encontra o tempo através da equação do sensor Real
v=solve((-1.98e-11)*x**6+(1.656e-08)*x**5+(-5.572e-
06)*x**4+(0.000979)*x**3+(-0.09861)*x**2+(5.877)*x+19.64-T,x)
#v=solve((-8.472e-15)*x**8+(8.794e-12)*x**7+(-3.792e-09)*x**6+(8.79e-
07)*x**5+(-0.0001185)*x**4+(0.009418)*x**3+(-0.4301)*x**2+(10.92)*x+46.15-
T, x)
print('Raizes:\n',v)
x=v[0]
```

```
print('Raiz selecionada\n',x)
#Informa o tempo usado para encontrar a raiz (função inversa, encontra o tempo em segundos do sistema)
print("---Time 1 --- %s seconds ---\n" % (time.time() - start_time))
#Calcula os sensores Virtuais a partir do tempo encontrado na equação do sensor Real
S1=(-4.611e-11)*x**6+(3.688e-08)*x**5+(-1.158e-05)*x**4+(0.001815)*x**3+(-0.1502)*x**2+(6.516)*x+63.96
print('Sensor virtual 1:',S1)
S2=(-4.611e-11)*x**6+(3.688e-08)*x**5+(-1.158e-05)*x**4+(0.001815)*x**3+(-0.1502)*x**2+(6.516)*x+63.96
print('Sensor virtual 2:', S2)
S3=(9.29e-12)*x**6+(-6.347e-09)*x**5+(1.404e-06)*x**4+(-4.41e-05)*x**3+(-0.02684)*x**2+(4.034)*x+9.137
print('Sensor virtual 3:', S3)
S4=(9.291e-12)*x**6+(-6.348e-09)*x**5+(1.404e-06)*x**4+(-4.414e-05)*x**3+(-0.02683)*x**2+(4.033)*x+9.139
print('Sensor virtual 4:', S4)
print('-------------------------------------\n')
#SC=(-1.98e-11)*x**6+(1.656e-08)*x**5+(-5.572e-06)*x**4+(0.000979)*x**3+(-0.09861)*x**2+(5.877)*x+19.64
#print('Sensor 5:', SV)
print("---Time end %s seconds ---" % (time.time() - start_time))
#########################################################################

# CHAPA - Teste com Função Logaritmica
#Testes Chapa topologia 1 para n
#Sensor Virtual com 1 Gradiente de Temperatura, com Função Logaritmica, 1 Sensor Entrada, 4 SV de Saida, Resolve tempo com log
#Versão 1.0
from sympy import *
x=Symbol('x', real=True)
#Mede o tempo de execução
import time
```

```
start_time = time.time()
#Simula ipmut de temperatura do sensor real
T=199.760
print('Temperatura Real:',T,'ºC')
print('--------------------------------------\n')
#Encontra o tempo através da equação do sensor Real
v=solve(40.935*log(x) - 9.9564 - T)
print('Raizes:',v)
x=v[0]
#Informa o tempo usado para encontrar a raiz (função inversa, encontra o tempo em segundos do sistema)
print("---Tempo Raiz --- %s segundos ---" % (time.time() - start_time))
#Calcula os sensores Virtuais a partir do tempo encontrado na equação do sensor Real
S1 = 27.542*log(x) + 60.696
print('Sensor virtual 1:',S1)
S2 = 27.542*log(x) + 60.695
print('Sensor virtual 2:', S2)
S3 = 47.873*log(x) - 47.405
print('Sensor virtual 3:', S3)
S4 = 47.873*log(x) - 47.405
print('Sensor virtual 4:', S4)
#SC = 40.935*log(x) - 9.9564
#print('Sensor virtual 4:', SC)
print('--------------------------------------\n')

print("---Tempo final %s segundos ---" % (time.time() - start_time))
```

B.3 Plate 2G codes

```
#CHAPA 2G - Teste com Função Polinomial grau 8
#Sensor Virtual, com Polinômio de Grau 8, 1 Sensor Entrada, 4 SV, Tempo Func.
Pol. Inv
#Versão 1.0
from sympy import *
x=Symbol('x', real=True)
#Mede o tempo de execução
import time
start_time = time.time()
#Simula imput de temperatura do sensor real
T=199.990
print('Temperatura Real:',T,'ºC')
print('--------------------------------------\n')
#Encontra o tempo através da equação do sensor Real
v=solve((-2.229e-13)*x**8 + 1.415e-10*x**7 + (-3.778e-08)*x**6 + 5.536e-
06*x**5 +(-0.0004869)*x**4 + 0.02655*x**3 + (-0.9001)*x**2 + 18.42*x + 8.753-
T, x)
print('Raizes:\n',v)
x=v[0]
print('Raiz 1 selecionada\n',x)
print('--------------------------------------\n')
#Informa o tempo usado para encontrar a raiz (função inversa, encontra o tempo
em segundos do sistema)
print("---Tempo Raiz --- %s segundos ---" % (time.time() - start_time))
print('--------------------------------------\n')
#Calcula os sensores Virtuais a partir do tempo encontrado na equação do
sensor Real
S1=(-4.638e-13)*x**8 + 2.864e-10*x**7 + (-7.389e-08)*x**6 + 1.034e-05*x**5 +(-
0.0008533)*x**4 + 0.04245*x**3 + (-1.256)*x**2 + 21.17*x + 24.6
print('Sensor virtual 1:',S1)
S2=(-8.423e-13)*x**8 + 5.126e-10*x**7 + (-1.297e-07)*x**6 + 1.767e-05*x**5 +
(-0.001402)*x**4 + 0.06553*x**3 + (-1.743)*x**2 + 24.22*x + 55.24
print('Sensor virtual 2:', S2)
```

```
S3=3.085e-13*x**8 + (-1.847e-10)*x**7 + 4.55e-08*x**6 + (-5.904e-06)*x**5 +
0.0004248*x**4 + (-0.01563)*x**3 + 0.1554*x**2 + 7.198*x + 10.35
print('Sensor virtual 3:', S3)
S4=(-4.919e-13)*x**8 + 3.002e-10*x**7 + (-7.63e-08)*x**6 + 1.048e-05*x**5 + (-
0.0008437)*x**4 + 0.04067*x**3 + (-1.159)*x**2 + 18.95*x + 41.42
print('Sensor virtual 4:', S4)
#SC=(-2.229e-13)*x**8 + 1.415e-10*x**7 + (-3.778e-08)*x**6 + 5.536e-06*x**5
+(-0.0004869)*x**4 + 0.02655*x**3 + (-0.9001)*x**2 + 18.42*x + 8.753
#print('Sensor 5:', SC)
print('-------------------------------------\n')
print("---Tempo final %s segundos ---" % (time.time() - start_time))
```

```
#CHAPA 2G - Teste com Função Polinomial grau 6
#Sensor Virtual, com Polinômio de Grau 6, 1 Sensor Entrada, 4 SV, Tempo Func.
Pol. Inv
#Versão 1.0
from sympy import *
x=Symbol('x', real=True)
#Mede o tempo de execução
import time
start_time = time.time()
#Simula imput de temperatura do sensor real
T=81.443
print('Temperatura Real:',T,'ºC')
print('-------------------------------------\n')
#Encontra o tempo através da equação do sensor Real
v=solve((-1.038e-09)*x**6 + 5.068e-07*x**5 + (-9.859e-05)*x**4 + 0.009774*x**3
+ (-0.5244)*x**2 + 14.8*x + 17.92-T, x)
print('Raizes:\n',v)
x=v[0]
print('Raiz 1 selecionada\n',x)
print('-------------------------------------\n')
#Informa o tempo usado para encontrar a raiz (função inversa, encontra o tempo
em segundos do sistema)
```

```
print("---Tempo Raiz --- %s segundos ---" % (time.time() - start_time))
print('-------------------------------------\n')
#Calcula os sensores Virtuais a partir do tempo encontrado na equação do sensor Real
S1=(-1.331e-09)*x**6 + 6.359e-07*x**5 + (-0.0001201)*x**4 + 0.0114  *x**3 + ( -0.5731) *x**2 + 14.7 *x + 40.73
print('Sensor virtual 1:',S1)
S2=(-1.657e-09)*x**6 + 7.747e-07 *x**5 + (-0.0001418)*x**4 + 0.01284  *x**3 + (-0.5984) *x**2 + 13.5  *x + 81.69
print('Sensor virtual 2:', S2)
S3= 9.145e-11*x**6 + (-8.712e-09)*x**5 + (-8.044e-06) *x**4 + 0.002184  *x**3 + (-0.2257)*x**2 + 10.71*x + 1.79
print('Sensor virtual 3:', S3)
S4=(-1.113e-09)*x**6 + 5.304e-07 *x**5 +(-9.996e-05)*x**4 + 0.009494  *x**3 + (-0.4805)*x**2 + 12.58  *x + 57.18
print('Sensor virtual 4:', S4)
#SC=(-1.038e-09)*x**6 + 5.068e-07*x**5 + (-9.859e-05)*x**4 + 0.009774*x**3 + (-0.5244)*x**2 + 14.8*x + 17.92   + 18.42*x + 8.753
#print('Sensor 5:', SC)
print('-------------------------------------\n')
print("---Tempo final %s segundos ---" % (time.time() - start_time))
#########################################################################
#CHAPA 2G - Teste com Função Logaritmica
#Placa com 2 Gradientes
#Sensor Virtual, com Função logaritmica, 1 Sensor Entrada 4 SV, Tempo Func. Log. Inv
#Versão 1.0
from sympy import *
x=Symbol('x', real=True)
#Mede o tempo de execução
import time
start_time = time.time()

#Simula imput de temperatura do sensor real
```

```
T=81.443
print('Temperatura Real:',T,'ºC')
print('--------------------------------------\n')
#Encontra o tempo através da equação do sensor Real
v=solve(31.971*log(x) + 58.208-T, x)
print('Raizes:\n',v)
x=v[0]
print('Raiz 1 selecionada\n',x)
print('--------------------------------------\n')
#Informa o tempo usado para encontrar a raiz (função inversa, encontra o tempo em segundos do sistema)
print("---Tempo Raiz --- %s segundos ---" % (time.time() - start_time))
print('--------------------------------------\n')
#Calcula os sensores Virtuais a partir do tempo encontrado na equação do sensor Real
S1=25.801*log(x) + 86.154
print('Sensor virtual 1:',S1)
S2=16.159*log(x) + 129.52
print('Sensor virtual 2:', S2)
S3= 41.835*log(x) + 12.559
print('Sensor virtual 3:', S3)
S4=24.149*log(x) + 93.09
print('Sensor virtual 4:', S4)
#SC=31.971*log(x) + 58.208
#print('Sensor 5:', SC)
print('--------------------------------------\n')
print("---Tempo final %s segundos ---" % (time.time() - start_time))
```

B.4 Cylinder codes

```
# CILINDRO - Teste com Função Polinomial grau 8
#Sensor Virtual com 1 Gradiente de Temperatura, com Polinômio de Grau 8, 1
Sensor Entrada, 4 SV, Tempo Func. Pol. Inv
#Versão 1.0
from sympy import *
x=Symbol('x', real=True)
#Mede o tempo de execução
import time
start_time = time.time()
#Simula imput de temperatura do sensor real
T=184.980
print('Temperatura Real:',T,'ºC')
print('--------------------------------------\n')
#Encontra o tempo através da equação do sensor Real
v=solve((2.001e-10)*x**8 + (-5.145e-08)*x**7 + (5.453e-06)*x**6 + (-
0.0003063)*x**5 + (0.009708)*x**4 + (-0.1667)*x**3 + (1.117)*x**2 + (8.682)*x
+ 15.94-T, x)
print('Raizes:',v)
if(v[0]>0):
   x=v[0]
if(v[1]>0):
   x=v[1]
print('Raiz selecionada:',x)
print('--------------------------------------\n')
#Informa o tempo usado para encontrar a raiz (função inversa, encontra o
tempo em segundos do sistema)
print("---Tempo Raiz --- %s segundos ---" % (time.time() - start_time))
print('--------------------------------------\n')
#Calcula os sensores Virtuais a partir do tempo encontrado na equação do
sensor Real
S1=(-7.469e-10)*x**8 + (1.943e-07)*x**7 + (-2.094e-05)*x**6 + (0.001208)*x**5
+ (-0.04024)*x**4 + (0.7808)*x**3 + (-8.512)*x**2 + (49.48)*x + 40.81
print('Sensor virtual 1:',S1)
```

```
S2=(-1.536e-10)*x**8 + (4.142e-08)*x**7 + (-4.67e-06)*x**6 + (0.0002858)*x**5
+ (-0.01034)*x**4 + (0.2281)*x**3 + (-3.134)*x**2 + (29.04)*x + 12.06
print('Sensor virtual 2:', S2)
#S3=(2.001e-10)*x**8 + (-5.145e-08)*x**7 + (5.453e-06)*x**6 + (-
0.0003063)*x**5 + (0.009708)*x**4 + (-0.1667)*x**3 + (1.117)*x**2 + (8.682)*x
+ 15.94
#print('Sensor virtual 3:', S3)
S4=(2.81e-10)*x**8 + (-7.32e-08)*x**7 + (7.888e-06)*x**6 + (-0.0004536)*x**5
+ (0.01492)*x**4 + (-0.2751)*x**3 + (2.374)*x**2 + (1.993)*x + 19.07
print('Sensor virtual 4:', S4)
S5=(2.809e-10)*x**8 + (-7.317e-08)*x**7 + (7.886e-06)*x**6 + (-
0.0004535)*x**5 + (0.01492)*x**4 + (-0.2751)*x**3 + (2.375)*x**2 + (1.993)*x +
19.07
print('Sensor virtual 5:', S5)
print('--------------------------------------\n')
print("---Tempo final %s segundos ---" % (time.time() - start_time))
########################################################################

# CILINDRO - Teste com Função Polinomial grau 6
#Sensor Virtual com 1 Gradiente de Temperatura, com Polinômio de Grau 6, 1
Sensor Entrada, 4 SV, Tempo Func. Pol. Inv
#Versão 1.0
from sympy import *
x=Symbol('x', real=True)
#Mede o tempo de execução
import time
start_time = time.time()
#Simula imput de temperatura do sensor real
T=195.870
print('Temperatura Real:',T,'ºC')
print('--------------------------------------\n')
#Encontra o tempo através da equação do sensor Real
#v=solve((4.404e-08)*x**6 + (-7.937e-06)*x**5 + (0.0004998)*x**4 + (-
0.01006)*x**3 + (-0.2228)*x**2 + (13.32)*x + 12.13-T, x)
```

```
v=solve((-2.772e-07)*x**6 + (5.515e-05)*x**5 + (-0.004269)*x**4 +
(0.1623)*x**3 + (-3.162)*x**2 + (30.8)*x + 56.27 -T, x)
print('Raizes:',v)
if(v[0]>0):
   x=v[0]
if(v[1]>0):
   x=v[1]
print('Raiz selecionada:',x)
print('-------------------------------------\n')
#Informa o tempo usado para encontrar a raiz (função inversa, encontra o
tempo em segundos do sistema)
print("---Tempo Raiz --- %s segundos ---" % (time.time() - start_time))
print('-------------------------------------\n')
#Calcula os sensores Virtuais a partir do tempo encontrado na equação do
sensor Real
S1=(-2.772e-07)*x**6 + (5.515e-05)*x**5 + (-0.004269)*x**4 + (0.1623)*x**3 +
(-3.162)*x**2 + (30.8)*x + 56.27
print('Sensor virtual 1:',S1)
S2=(-1.136e-07)*x**6 + (2.334e-05)*x**5 + (-0.001898)*x**4 + (0.07862)*x**3 +
(-1.806)*x**2 + (24.29)*x + 16.05
print('Sensor virtual 2:', S2)
S3=(4.404e-08)*x**6 + (-7.937e-06)*x**5 + (0.0004998)*x**4 + (-0.01006)*x**3
+ (-0.2228)*x**2 + (13.32)*x + 12.13
print('Sensor virtual 3:', S3)
S4=(9.687e-08)*x**6 + (-1.852e-05)*x**5 + (0.001323)*x**4 + (-0.04108)*x**3 +
(0.3487)*x**2 + (9.074)*x + 13.21
print('Sensor virtual 4:', S4)
S5=(9.699e-08)*x**6 + (-1.855e-05)*x**5 + (0.001324)*x**4 + (-0.04113)*x**3 +
(0.3494)*x**2 + (9.075)*x + 13.21
print('Sensor virtual 5:', S5)
print('-------------------------------------\n')
print("---Tempo final %s segundos ---" % (time.time() - start_time))
######################################################################
```

```
# CILINDRO - Teste com Função Logaritmica

#Sensor Virtual com 1 Gradiente de Temperatura, com Função Logaritmica, 1 Sensor Entrada, 4 SV de Saida, Resolve tempo com log
#Versão 1.0
from sympy import *
x=Symbol('x', real=True)
#Mede o tempo de execução
import time
start_time = time.time()
#Simula ipmut de temperatura do sensor real
T=195.870
print('Temperatura Real:',T,'ºC')
print('---------------------------------------\n')
#Encontra o tempo através da equação do sensor Real
v=solve(45.107*log(x) + 20.147 - T)
print('Raizes:',v)
print('---------------------------------------\n')
x=v[0]
#Informa o tempo usado para encontrar a raiz (função inversa, encontra o tempo em segundos do sistema)
print("---Tempo Raiz --- %s segundos ---" % (time.time() - start_time))
print('---------------------------------------\n')
#Calcula os sensores Virtuais a partir do tempo encontrado na equação do sensor Real
S1 = 29.207*log(x) + 89.948
print('Sensor virtual 1:',S1)
S2 = 41.268*log(x) + 39.353
print('Sensor virtual 2:', S2)
#S3 = 45.107*log(x) + 20.147
#print('Sensor virtual 3:', S3)
S4 = 45.889*log(x) + 15.317
print('Sensor virtual 4:', S4)
S5 = 45.913*log(x) + 15.311
```

```
print('Sensor virtual 5:', S5)
print('--------------------------------------\n')
print("---Tempo final %s segundos ---" % (time.time() - start_time))
```

B.5 Pipe codes

```
#TUBO - Teste com Função Polinomial grau 3
#Sensor Virtual com 1 Gradiente de Temperatura, com Polinômio de Grau 3, 1
Sensor Entrada, 4 SV, Tempo Func. Pol. Inv
#Versão 1.0
from sympy import *
x=Symbol('x', real=True)
#Mede o tempo de execução
import time
start_time = time.time()
#Simula imput de temperatura do sensor real
T=28.497
print('Temperatura Real:',T,'ºC')
print('---------------------------------------\n')
#Encontra o tempo através da equação do sensor Real
v=solve((0.0002697)*x**3 + (-0.09706)*x**2 + (11.55)*x + 14.61-T, x)
print('Raizes:',v)
x=v[0]
print('Raiz selecionada:',x)
print('---------------------------------------\n')
#Informa o tempo usado para encontrar a raiz (função inversa, encontra o tempo
em segundos do sistema)
print("---Tempo Raiz --- %s segundos ---" % (time.time() - start_time))
print('---------------------------------------\n')
#Calcula os sensores Virtuais a partir do tempo encontrado na equação do
sensor Real
S1=(0.0003635)*x**3 + (-0.1136)*x**2 + (12.38)*x + 53.46
print('Sensor virtual 1:',S1)
S2=(0.0003307)*x**3 + (-0.1078)*x**2 + (12.09)*x + 27.94
print('Sensor virtual 2:', S2)
#S3=(0.0002697)*x**3 + (-0.09706)*x**2 + (11.55)*x + 14.61
#print('Sensor virtual 3:', S3)
S4=(0.0002201)*x**3 + (-0.08826)*x**2 + (11.1)*x + 8.448
print('Sensor virtual 4:', S4)
S5=(0.0002021)*x**3 + (-0.08502 )*x**2 + (10.93)*x + 6.683
```

```
print('Sensor virtual 5:', S5)
print('-------------------------------------\n')
print("---Tempo final %s segundos ---" % (time.time() - start_time))

######################################################################

#TUBO - Teste com Função Logaritmica
#Sensor Virtual com 1 Gradiente de Temperatura, com Função Logaritmica, 1 Sensor Entrada, 4 SV de Saida, Resolve tempo com log
#Versão 1.0
from sympy import *
x=Symbol('x', real=True)
#Mede o tempo de execução
import time
start_time = time.time()
#Simula ipmut de temperatura do sensor real
T=469.920
print('Temperatura Real:',T,'ºC')
print('-------------------------------------\n')
#Encontra o tempo através da equação do sensor Real
v=solve(107.36*log(x) - 44.076 - T)
print('Raizes:',v)
print('-------------------------------------\n')
x=v[0]
#Informa o tempo usado para encontrar a raiz (função inversa, encontra o tempo em segundos do sistema)
print("---Tempo Raiz --- %s segundos ---" % (time.time() - start_time))
print('-------------------------------------\n')
#Calcula os sensores Virtuais a partir do tempo encontrado na equação do sensor Real
S1 = 110.03*log(x) - 5.1492
print('Sensor virtual 1:',S1)
S2 = 109.11*log(x) - 30.685
print('Sensor virtual 2:', S2)
```

```
#S3 = 107.36*log(x) - 44.076
#print('Sensor virtual 3:', S3)
S4 = 105.88*log(x) - 50.294
print('Sensor virtual 4:', S4)
S5 = 105.26*log(x) - 52.028
print('Sensor virtual 5:', S5)
print('--------------------------------------\n')
print("---Tempo final %s segundos ---" % (time.time() - start_time))
```

B.6 Mold codes

```
#Molde 1 para n
#Sensor Virtual, com Polinômio de Grau 4, Tempo Func. Pol. Inv
#Versão 1.0
from sympy import *
x=Symbol('x', real=True)
#Mede o tempo de execução
import time
start_time = time.time()
#Simula imput de temperatura do sensor real
T=39.602
print('Temperatura Real:',T,'ºC')
print('---------------------------------------\n')
#Encontra o tempo através da equação do sensor Real
v=solve((-0.000385)*x**4 + 0.0422*x**3 + (-1.668)*x**2 + 17.55*x + 183.1-T,x)
print('Raizes:',v)
x=v[1]
#x=v[0]
print('Raiz selecionada:',x)
print('---------------------------------------\n')
#Informa o tempo usado para encontrar a raiz (função inversa, encontra o tempo
em segundos do sistema)
print("---Tempo Raiz --- %s segundos ---" % (time.time() - start_time))
print('---------------------------------------\n')
#Calcula os sensores Virtuais a partir do tempo encontrado na equação do
sensor Real
S11= (-0.0004145)*x**4 + 0.0441*x**3 + (-1.702)*x**2 + 17.56*x + 185.5
print('Sensor virtual S11:',S11)
S12=(-0.0004212)*x**4 + 0.04454*x**3 + -1.713*x**2 + 17.67*x + 185.2
print('Sensor virtual S12:', S12)
S13=(-0.0004024)*x**4 + 0.04329*x**3 + -1.684*x**2 + 17.4*x + 185.9
print('Sensor virtual S13:', S13)
#S21=(-0.000385)*x**4 + 0.0422*x**3 + (-1.668)*x**2 + 17.55*x + 183.1
#print('Sensor virtual S21:', S21)
S23=(-0.0005876)*x**4 + 0.05583*x**3 + (-1.982)*x**2 + 20.48*x + 173.8
```

```
print('Sensor virtual S23:', S23)
S31=(-0.0004897)*x**4 + 0.04915*x**3 + (-1.819)*x**2 + 18.7 *x + 181
print('Sensor virtual S31:', S31)
S32=(-0.0005641)*x**4 + 0.0541*x**3 + (-1.932)*x**2 + 19.75*x + 177.6
print('Sensor virtual S32:', S32)
S33=(-0.0005812)*x**4 + 0.05526*x**3 + (-1.958)*x**2 + 19.96*x + 177
print('Sensor virtual S33:', S33)
print('--------------------------------------\n')
print("---Tempo final %s segundos ---" % (time.time() - start_time))
```

```
#Molde n para n
#Sensor Virtual com 1 Gradiente de Temperatura, com Polinômio de Grau 4, Tempo Func. Pol. Inv
#Versão 1.0
from sympy import *
x=Symbol('x', real=True)
#Mede o tempo de execução
import time
start_time = time.time()
#Simula imput de temperatura do sensor real
T1=39.602
T2=40.397
print('Temperatura Real S21:',T1,'ºC')
print('--------------------------------------\n')
print('Temperatura Real S23:',T2,'ºC')
print('--------------------------------------\n')
#Encontra o tempo através da equação do sensor Real
v=solve((-0.000385)*x**4 + 0.0422*x**3 + (-1.668)*x**2 + 17.55*x + 183.1-T1,x)
print('Raizes:',v)
#if(v[0]>0):
#x1=v[0]
#if(v[1]>0):
x1=v[1]
```

```
r=solve((-0.0005876)*x**4 + 0.05583*x**3 + (-1.982)*x**2 + 20.48*x + 173.8-T2,x)
print('Raizes:',v)
#if(r[0]>0):
#x2=v[0]
#if(r[1]>0):
x2=r[1]
print('---------------------------------------\n')
print('Raiz 1 selecionada:',x1)
print('---------------------------------------\n')
print('Raiz 2 selecionada:',x2)
#Informa o tempo usado para encontrar a raiz (função inversa, encontra o tempo em segundos do sistema)
print("---Tempo para calular Raizes --- %s segundos ---" % (time.time() - start_time))
print('---------------------------------------\n')
#Encontra o tempo médio
x=(x1+x2)/2
print('Tempo medio calculado:',x,'s')
print('---------------------------------------\n')
#Informa o tempo usado para encontrar media das raizes
print("---Tempo Raiz --- %s segundos ---" % (time.time() - start_time))
print('---------------------------------------\n')
#Calcula os sensores Virtuais a partir do tempo encontrado na equação do sensor Real
S11= (-0.0004145)*x**4 + 0.0441*x**3 + (-1.702)*x**2 + 17.56*x + 185.5
print('Sensor virtual S11:',S11)
S12=(-0.0004212)*x**4 + 0.04454*x**3 + -1.713*x**2 + 17.67*x + 185.2
print('Sensor virtual S12:', S12)
S13=(-0.0004024)*x**4 + 0.04329*x**3 + -1.684*x**2 + 17.4*x + 185.9
print('Sensor virtual S13:', S13)
#S21=(-0.000385)*x**4 + 0.0422*x**3 + (-1.668)*x**2 + 17.55*x + 183.1
#print('Sensor virtual S21:', S21)
```

```
#S23=(-0.0005876)*x**4 + 0.05583*x**3 + (-1.982)*x**2 + 20.48*x + 173.8
#print('Sensor virtual S23:', S23)
S31=(-0.0004897)*x**4 + 0.04915*x**3 + (-1.819)*x**2 + 18.7 *x + 181
print('Sensor virtual S31:', S31)
S32=(-0.0005641)*x**4 + 0.0541*x**3 + (-1.932)*x**2 + 19.75*x + 177.6
print('Sensor virtual S32:', S32)
S33=(-0.0005812)*x**4 + 0.05526*x**3 + (-1.958)*x**2 + 19.96*x + 177
print('Sensor virtual S33:', S33)
print('-------------------------------------\n')
print("---Tempo final %s segundos ---" % (time.time() - start_time))
```