Instituto Universitário de Lisboa

Department of Information Science and Technology

Development of new technologies for the THz domain with applications on structural health monitoring

João Pedro Calado Barradas Branco Pavia

A Dissertation presented in partial fulfilment of the Requirements for the Degree of

Master in Telecommunications and Computer Engineering

Supervisor:

PhD. Marco Alexandre dos Santos Ribeiro,

Assistant Professor at ISCTE-IUL

Co-supervisor:

PhD. Nuno Manuel Branco Souto,

Assistant Professor at ISCTE-IUL

December 2018

Acknowledgments

First of all, I would like to express my gratitude to my family for their continuous support and confidence. They played an important role on giving me the necessary conditions to accomplish this dissertation development, and for helping me getting my master's degree. Who I am today and what I have accomplished is only a consequence of all they have given and taught me.

My greater acknowledgement is for Professor Marco Ribeiro and for Professor Nuno Souto for giving me the opportunity of developing this dissertation as my research mentors, and for making this dissertation and R&D project possible. I'm grateful for their continuous advice on different subject matters, such as propagation techniques, structural health monitoring, metamaterials, filter design, software defined radio, electronics, experimental work tips and many other cases. I think that all required conditions have been provided to develop this work, and their knowledge was outstanding, which added a relevant and considerable value to this work.

I would like to leave a word of appreciation to the administrative staff at ISCTE-IUL and Instituto das Telecomunicações that made many unavoidable burocracies so much easier to handle: Marisa Manteigas from ISTA and Tereza Traquinas and Ana Rodrigues from Instituto das Telecomunicações.

I would like to acknowledge my fellow IEEE colleagues and volunteers of our local IEEE Student Branch at ISCTE-IUL, I never imagined the amazing things that we would end up accomplishing during these years: from kickstarting numerous technical workshops and to organizing huge events. It was fantastic to be able to learn from and collaborate with so many talented and passionated individuals, for what they do.

To all my friends, colleagues and students, which were always present during my entire path, which in one way or another gave me their support during the development of this dissertation. Their personal opinions and helpful advices increased this dissertation's value on various aspects.

Finally, I would like also to acknowledge the support of the Fundação para a Ciência e Tecnologia through the funding of TUBITAK/0002/2014 project.

Abstract

Structural health monitoring (SHM) plays a major role in industry and engineering today, as this methodology has completely changed the paradigm of how to assess the health status of structures. Previously, the methodology of reference, Non-Destructive Testing (NDT), consisted on conducting periodic inspections, made by specialized professionals that used external equipment. However, this method was quite expensive and time consuming.

SHM systems are considered as an alternative to traditional methods, since these systems allow the continuous acquisition of the structural responses from the disturbances experienced by the structure. Through the measurement data, it will be possible to know the health condition of the structure and the existence of structural damages. Sensors represent a fundamental component of these systems, since they are responsible for the acquisition of data related to the health and integrity of the structures and they should be accessible, light and discrete, so as not to impose cost and weight on the structure and not to interfere with the structural resistance. Although the THz domain is a relatively new field of research in electromagnetic sensors, it is known that the use of frequency selective surfaces (FSSs) allows the creation of compact devices which can be easily integrated into the surface of the structures without harming its functionality and aesthetic appearance.

The main goal of this dissertation is to study and numerically simulate stress sensors for the detection of THz radiation for applications in SHM, with high sensitivity and selectivity. Taylloring these characteristics to provide a good detection has been achieved through appropriate selection of materials and careful design of stress tunnable band pass filters for the THz domain. Numerical modelling of both the mechanical and electromagnetics, solving the elasticity equation and Maxwell's equations, respectively, has been undertaken for two types of assembling of the devices using different thermoplastic polymers such as high-density polyethylene (HDPE) and polytetrafluoroethylene (PTFE). It was observed that two methodologies can be adopted to control the level of force applied to the sensors, namely through, the variation of the electric current and the positioning of the coil on the plunger.

Keywords: frequency selective surfaces; filters; stress sensors; structural health monitoring; terahertz.

Resumo

A monitorização de saúde estrutural (SHM) desempenha um papel bastante importante na indústria e na engenharia hoje em dia, uma vez que esta metodologia mudou completamente o paradigma de como aferir o estado de saude das estruturas. Anteriormente, a metodologia de referência consistia em realizar inspeções periódicas com recurso a equipamentos externos, as quais eram realizadas por profissionais especializados. Contudo, este método revelou ser bastante dispendioso e moroso.

Os sistemas SHM são considerados uma alternativa aos métodos tradicionais, uma vez que esses sistemas permitem a aquisição contínua das respostas estruturais provenientes das perturbações experimentadas pela estrutura e saber qual o estado de saúde da estrutura com base nas medições efectuadas. Os sensores são base destes sistemas, devendo ser acessíveis, leves e discretos, para não impor custo e peso à estrutura e não interferir na resistência estrutural. Apesar da vasta gama de sensores disponíveis, existe alguma carência de sensores altamente selectivos para o domínio dos Terahertz. Embora o domínio THz seja um campo relativamente novo de investigação em sensores eletromagnéticos, sabe-se que o uso de superfícies seletivas de frequência (FSSs) permite a criação de dispositivos compactos que podem ser facilmente integrados na superfície das estruturas sem prejudicar sua funcionalidade e aspecto estético.

O objetivo principal desta dissertação é estudar e simular numericamente sensores de stress para a detecção de radiação THz para aplicações em SHM, com alta sensibilidade e seletividade. A determinação dessas características para fornecer uma boa detecção foi conseguida através da seleção apropriada de materiais e do design cuidadoso de filtros passa-banda sintonizáveis por tensão para o domínio dos THz.

Foram realizadas simulações numéricas das componentes mecânica e electromagnética dos sensores com objectivo de resolver a equação de elasticidade e as equações de Maxwell respectivamente, para dois tipos de montagem dos dispositivos, utilizando diferentes polímeros termoplásticos, tais como polietileno de alta densidade (HDPE) e politetrafluoretileno (PTFE). Observou-se que podem ser adotadas duas metodologias para controlar o nível de força aplicado aos sensores, tais como a variação da corrente elétrica e o posicionamento da bobina no êmbolo.

Palavras-Chave: superficies seletivas de frequência; filtros; sensores de stress; monitorização de saúde estrutural; terahertz.

Contents

List of Tables

List of Figures

Symbols

A – Area

A – Absorption Coefficient

a – Radius of the Wires

α – Attenuation Constant

B – Magnetic Flux Density

B – Susceptance

β – Propagation Constant

c – Speed of Light in Vacuum

D – Electric Displacement

d – Distance between Wires

Δd – Coil Displacement

E – Electric Field

E – Young's Modulus

ε – Electric Permittivity

F – Lorentz Force

F_0 – Operating Frequency

F_a – Applied Force

F_g – Generated Force

F_s – Frequency Shift

G – Conductance

g – Air Gap

H – Magnetic Field Intensity

I – Current Wave

J – Electric Current Density

k – Wavenumber

L – Coil Inductance

l – Length of the Structure

Δl – Displacement of the Structure

N – Number of Turns

n – Refractive Index

∇ – Nabla Operator

q – Particle's Charge

Q – Quality Factor

ω – Angular Frequency

ω_0 – Resonance Frequency

R – Reflection Coefficient

R_{air} – Air Reluctance

$R_{plunger}$ – Plunger Reluctance

ρ – Charge Density

S – Scattering

T – Transmission Coefficient

V – Voltage Wave

v – Particle's Velocity

v_p – Phase Velocity

λ – Wavelength

θ – Angle of Incidence

$\Phi_{\boldsymbol{B}}$ – Magnetic Flux

μ – Magnetic Permeability

Z – Impedance

Y – Admittance

Abbreviations

EM – ElectroMagnetic

FEMM – Finite Element Method Magnetics

FSS – Frequency Selective Surface

HDPE – High Density PolyEthylene

HSS – Highly Selective Structure

IR – Infrared Radiation

IT – Instituto de Telecomunicações

MATLAB – MATrix LABoratory

MEMS – Micro-Electro-Mechanical Systems

MR – Microwave Radiation

NDT – Non-Destructive Testing

PTFE – PolyTetraFluoroEthylene

RF – Radio Frequency

SHM – Structural Health Monitoring

SI – The International System of Units

TE – Transverse Electric

THz – Terahertz

TL – Transmission Line

TM – Transverse Magnetic

Chapter 1 – Introduction

This chapter presents the concept and definition of this dissertation development, as well as its motivation, target goals, research methods and questions. It is also mentioned what contributions this work provided.

1.1. Overview and Motivation

The primary goal of structural health monitoring (SHM) is to identify damage in a structure. However, SHM is not a new concept, as important structures have always been routinely inspected, usually by hand, to detect damages and their effect on the structural performance. With that said in recent years, several research studies have been performed to improve the accuracy and reliability of this approach. Traditionally, structural inspections, most often consisting of visual inspections, were carried out by trained or experienced individuals [1]. The effectiveness of such inspections depends upon the accessibility of the structural location to be examined and, to a large extent, on the expertise of the individual conducting such inspections. Such approaches tend to be highly labour intensive and costly. Therefore, alternative methods that can automatically collect and analyse structural data to ascertain its condition have been sought by practitioners and the research community [1].

The Terahertz gap (1 THz = 1012 Hz) lies between the microwave (MR) and the infrared (IR) regions in the electromagnetic spectrum, ranging from 0.1 THz to 10 THz. Although, the first research works in the frequency region of THz started in 1975 and, since then, there has been a significant progress in the fields of instrumentation, imaging and spectroscopy. THz radiation is non-ionizing and can penetrate a wide variety of non-polar materials like clothing, paper, cardboard, wood, masonry, plastics and ceramics. However, it is strongly absorbed by polar molecules, such as water, and it is highly reflected by metals [2].

Given the fact that research involving the THz domain has grown rapidly over the past few years, there are already several applications for various areas such as: physics, chemistry, medicine, quality control, defense, security, among others [2]. In the domain of SHM, the utilization of Frequency Selective Surfaces (FSSs) allows us to create compact devices that can effortlessly be integrated in the surface of the structures without harming its functionality and aesthetic appearance, allowing mainly its monitoring [3].

There are already some devices that have been designed for this purpose and that operate in the microwave domain, which makes it legitimate to extend this idea to the THz domain.

1.2. Goals

With the present dissertation, we intend to study THz filters and stress sensors for the detection of THz radiation with high sensitivity, selectivity and responsiveness. To this end, different types of stress sensors will be designed and tested in order to discover a set of geometric characteristics, which can provide a good detection and make them suitable for their application to new technologies in particularly relevant areas such as Structural Health Monitoring.

In the first stage, it will be necessary to study each of these devices theoretically and to characterize its working principle. In a second stage, the devices will be simulated using numerical simulation methods to validate the theoretical models and to prove the working principle of the structures. Figure 1 ilustrates the overview of the working principle of the stress sensors that we intended to design and simulate on this dissertation.

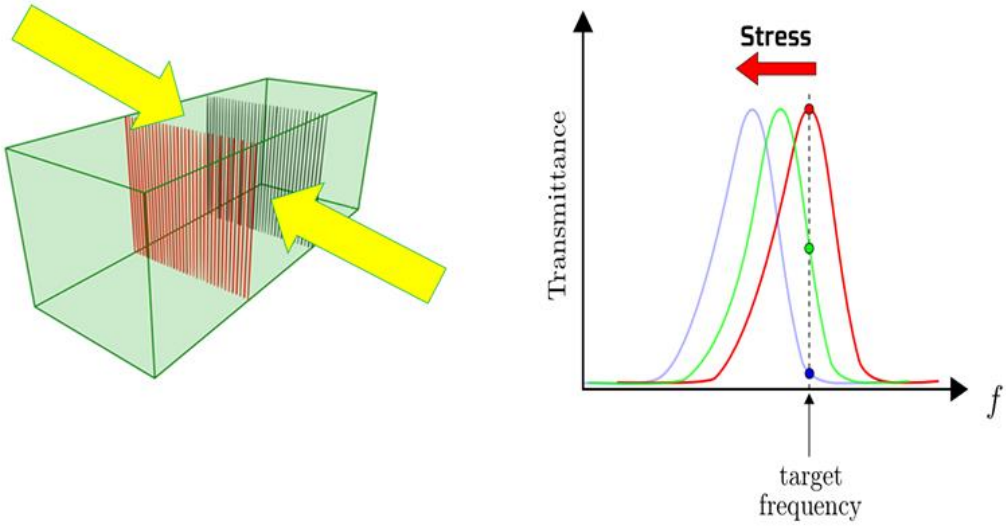

Figure 1 – Overview of the working principle of the stress sensors.

1.3. Research Questions and Methods

This dissertation also aims at answering certain questions, such as:

- What are the detection ranges that will be possible to achieve?

- What will be the set of parameters that will allow a good detection?
- Can the proposed filter be used in other applications?

To achieve the main goal specified in section 1.2 (Goals), it will be necessary to study the working principle of the structures and to perform some experimental procedures by simulating the structures and exploring some quantitative variables. By analyzing the generated data from numerical simulations, some conclusions can be formulated. It will be also possible to understand if there is the necessity to perform major or minor improvements in the design of the structures.

Considering these procedures, it possible to claim that the research method used in this dissertation is cyclical. It is important to refer that the most significant component of this work is the theoretical study of the working principle of the structures. Through this study, it is possible to analyze and to characterize the role of the resonant effects in the THz radiation detection mechanism. Furthermore, this study allows the understanding of how it may be possible to adjust the sensitivity and responsiveness of the devices under certain conditions.

1.4. Contributions

This dissertation presents the following contributions:

- It reviews existing approaches;
- It proposes a new THz filter design based on FSSs which is suitable for reconfigurable THz filters; stress sensors and optical modulators;
- It presents a theoretical study and respective characterization of the role of the resonant effects in the mechanism of detection of THz radiation of a new stress sensor for SHM applications;
- It presents two possible approaches for the application of the desired compression level to the sensor.

The work conducted in this dissertation has resulted in three publications:

- J.P. Pavia, W.J. Otter, S. Lucyszyn, M.A. Ribeiro,” Design of a THz-MEMS Frequency Selective Surface for Structural Health Monitoring”, META 2016 - The 7th International Conference on Metamaterials, Photonic Crystals and Plasmonics, Málaga, Spain, July 2016;

- J.P. Pavia, N. Souto, M.A. Ribeiro, "Highly Selective Filter Design for the THz Domain using Frequency Selective Surfaces", EIEC 2018 - The XII Iberian Meeting on Computational Electromagnetics, Coimbra, Portugal, May 2018;
- J.P. Pavia, M.A. Ribeiro, N. Souto, "Pressure Controlled Light Valve for the Terahertz", IET – Electronic Letters, 2018 (paper is being written for submission).

1.5. Dissertation Structure

This dissertation is composed of five chapters, each with several sections and subsections. The first chapter includes an overview about SHM and the THz domain. It also features the motivations that lead to the accomplishment of this dissertation as well as the proposed objectives are presented. Finally, the structure of the dissertation is indicated with a short summary about each chapter and the contributions of the same.

In the second chapter, a survey of the state of the art is presented in order to understand the fundamental concepts related with this work, the existing methodologies in the field of SHM, what already exists and what has been developed in international research related to the subject under study.

In the third chapter, a theoretical study and the respective characterization of the role of the resonant effects in the mechanism of detection of THz radiation in the scope of SHM applications will be presented.

In the fourth chapter, the results of the work developed will be presented and discussed properly. Tools such as ANSYS HFSS, Elmer, FEMM and MATLAB will be used to model, study and prove the working principle of the sensors.

In the last chapter, the main conclusions of this dissertation are presented, and some suggestions that could be developed in a future work perspective are mentioned.

Chapter 2 – State of Art

This chapter introduce the concepts inherent to the topic under analysis. As previously mentioned, this work focuses on the design of filters for the THz domain and later application in the development of sensors for structural health monitoring. In this sense, it will crucial to understand some fundamental concepts related to the topics of frequency selective surface filters and structural health monitoring. In addition, it will be also necessary to review what is being done in this field in the context of research.

2.1. Frequency Selective Surfaces Filters

In this sub-chapter will be discussed several topics related to frequency-selective surface filters. This approach will focus on the explanation of the physical mechanism under which these filters work, in the classification and representation according to their typology to obtain the equivalent circuit according to the transmission line theory and finally in their applications.

2.1.1. Definition and Classification

Frequency selective surfaces (FSS) are composed by a periodic arrangement of identical elements on a dielectric substrate excited by an incident plane wave, which are designed to reflect, transmit or absorb electromagnetic waves (EM) [4]. Given their characteristics, these structures can act as spatial filters, which have the capability to pass or block the waves at certain frequency bands. However, their response is influenced by the frequency, the angle and the polarization of the incident wave [4, 5]. According to Sanz Fernandez, this type of filters is commonly used in the design of plane-wave filters, which can operate from RF to THz frequencies [5].

FSS structures are designed based on resonant elements and can be classified mainly into two types: patch-type FSS and aperture-type FSS [6, 7]. A simple FSS structure consisting of periodic array of metallic crossed-dipole resonators and aperture-type crossed-dipole resonators are shown in Figure 2 and Figure 3, respectively.

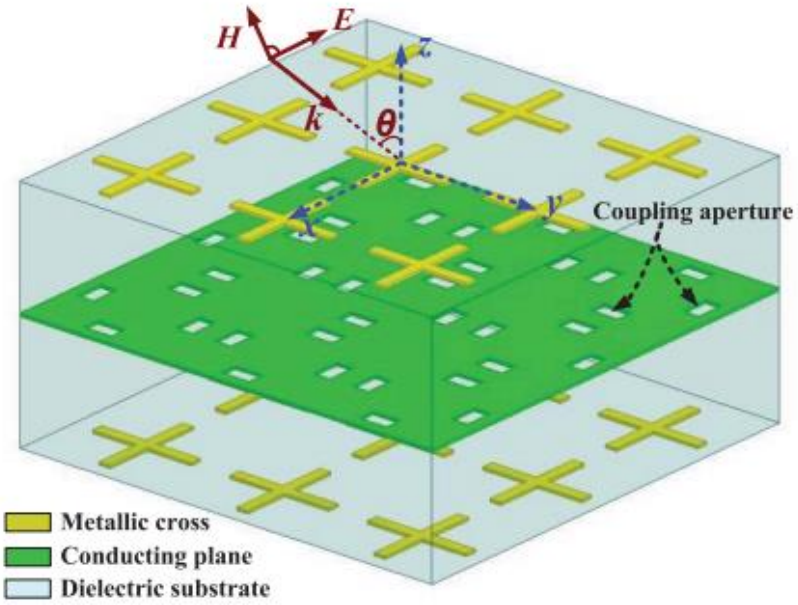

Figure 2 – FSS structure with metallic resonators, taken from [6].

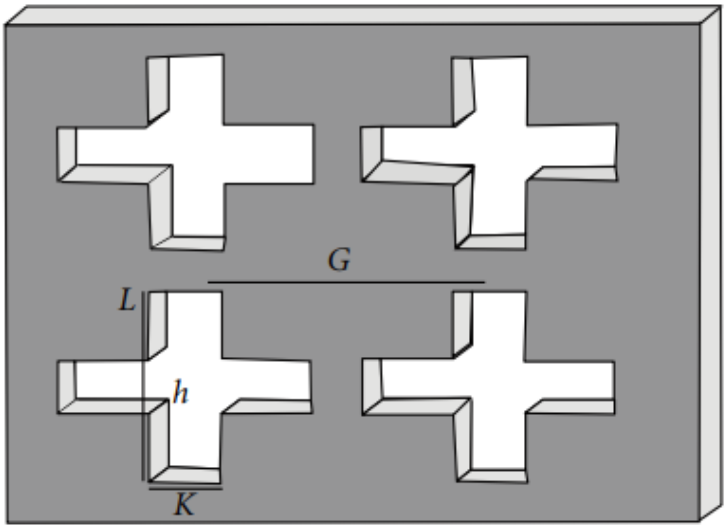

Figure 3 – FSS structure with aperture-type resonators, taken from [7].

According to Wang and Melo, both FSS structures produce a band-pass response, although in each of them it has been developed with different resonators [6, 7]. However, depending on the structure type and geometry chosen, it is possible to design a surface to act as bandpass, band-stop, low-pass or high-pass filters [4]. There should be some considerations when designing the structure, as it may be necessary to achieve certain specifications related to filter order, 3dB bandwidth, quality factor (Q), insertion loss, return loss, among others [4].

According to Munk, based on the shapes and geometry the structures are divided into four groups of elements, as shown in Figure 4, such as [4]:

- Group 1 - center connected or N pole;
- Group 2 - loop type;
- Group 3 - solid interior or plate type structures;
- Group 4 - combination of structures of all the previous groups.

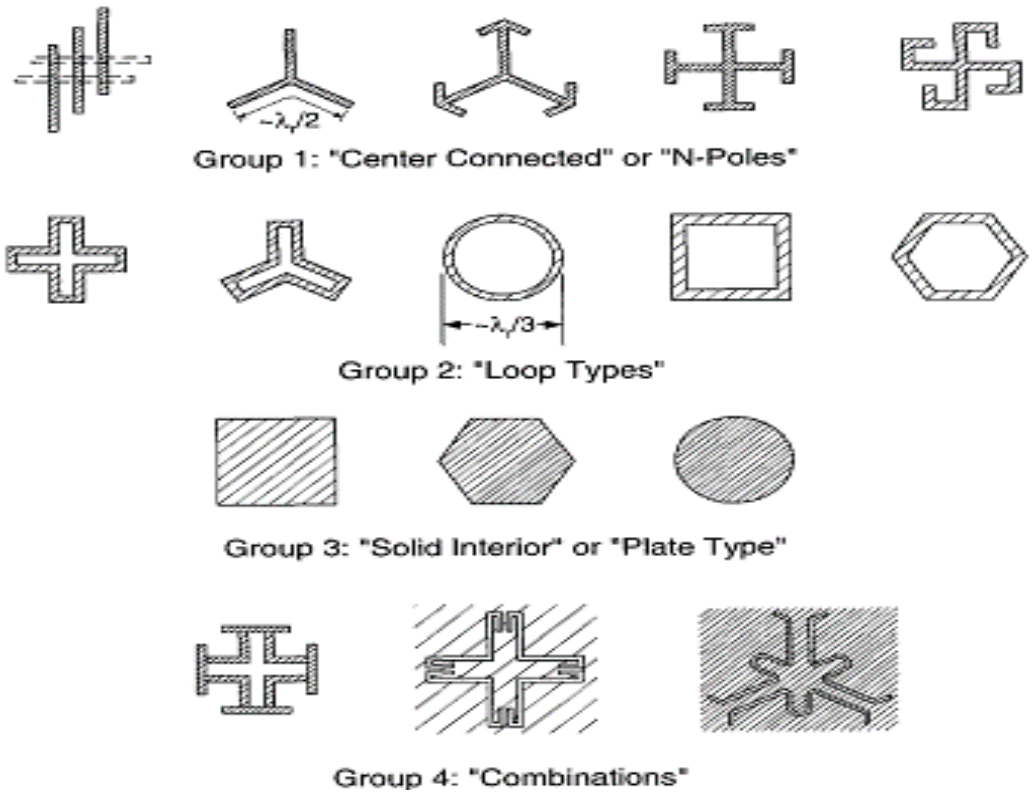

Figure 4 – Classification of different structures used for frequency selective surfaces, taken from [4].

According to Hooberman, the working principle of FSS filters can be easily understood through the following examples on Figure 5 [8].

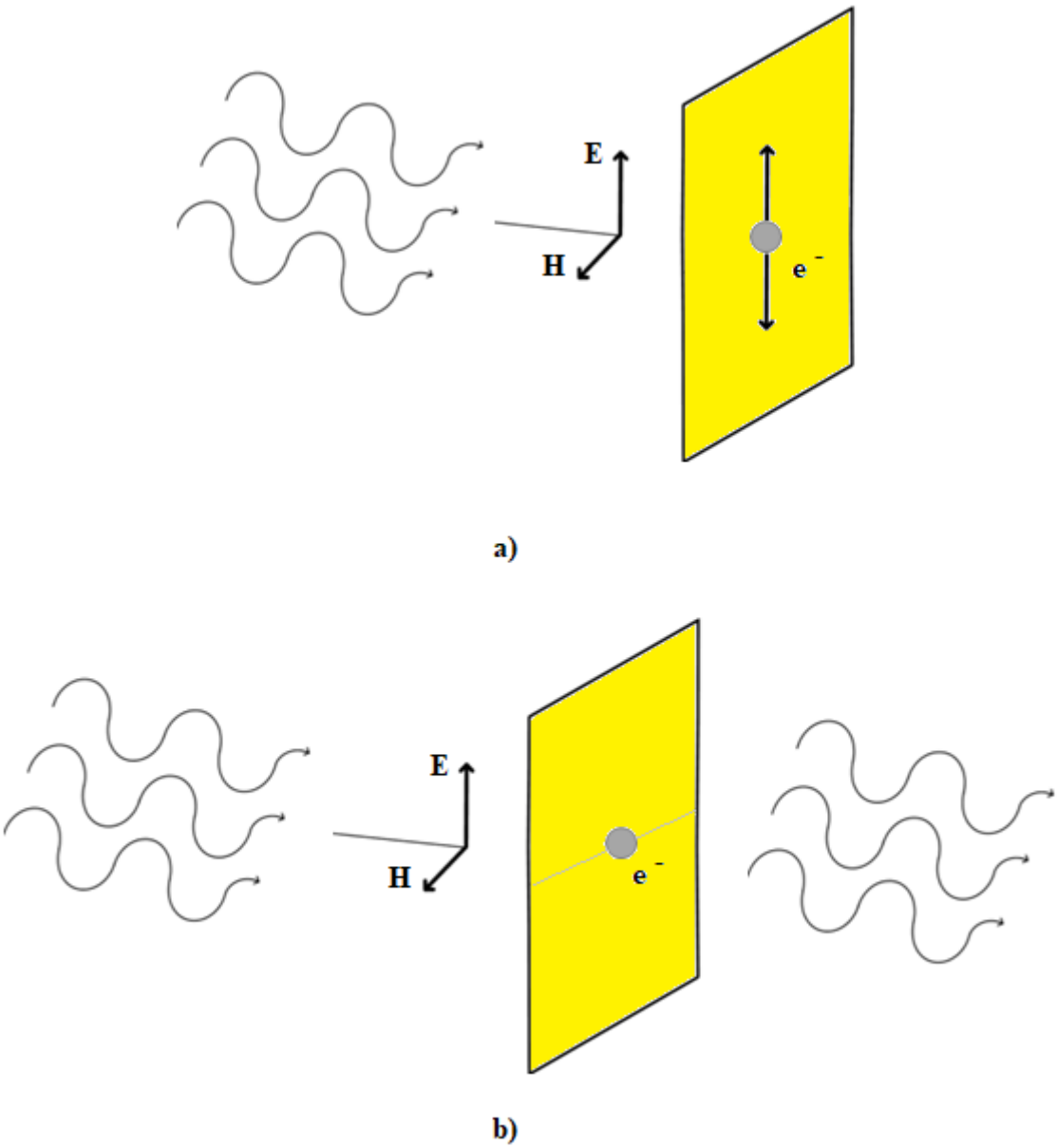

Figure 5 – Working principle of FSS under electric field a) parallel and b) perpendicular to the filter plane, adapted from [8].

In Figure 5 a), we have a plane wave source incident which stricks a metal plane at normal incidence. In the metal plane, we have an electron, which is only allowed to move in the vertical direction [8]. In this scenario, the electric field will exert a force on the electron, inducing its oscillation. A portion of the source energy is transferred to electron as kinetic energy, which allows it to remain oscillating and the remaining energy will be absorved by the electron according to the law of conservation of energy [5, 8]. As a consequence of this interaction, a radiated energy caused by the oscillation of the electrons will be formed. This energy will interfere destructively with the incident wave

on the right side of the plane, but the energy will be reflected on the left side [5, 8]. Considering this situation, we can affirm that the transmittance through the structure will be zero [8].

The previous situation shows a simple case where energy is not transmitted through the structure, but such as was mentioned above, this type of structures can also transmit EM waves. Considering a different scenario as shown in Figure 5 b), the direction of the electric field of incident plane wave is the same direction, but the electron is now restricted to move along a wire placed at horizontal direction [8]. In this situation, the electron is fully invisible to the incident wave and the striking wave will transmit through structure because the exerted force can't oscillate the electron in the horizontal direction [5, 8]. Considering this situation, we can affirm that the source energy does not suffer any absorption, what will imply that the transmittance through the structure will be very high [8].

According to Sanz Fernandez and Hooberman, the transmittance and the reflectance through the structure is highly dependent on frequency since the electrons in metals will absorb and re-radiate some wavelengths with higher efficiency than others [5, 8]. In addition, there are other types of structure-related parameters that can influence their behaviour, such as: the dimensions, the shape of the elements and its spacing on the structure [4, 5].

2.1.2. Equivalent Circuit Representation

According to Costa et all, the equivalent representation of FSSs can be achieved with series or shunt connections of inductances and capacitances [9]. Hooberman mentions that the most common classes of filters are strip grating filters and mesh filters, which which have their own peculiarities. Given this fact, the autor suggests addressing the case of strip grating structures as fundamental structures to later explain the most complex geometries [8].

There are two type of configuration related to strip grating filters, which can be seen in Figure 6 and Figure 7. The main differences between these two configurations are the resulting behaviour from the interation between the structure and the direction of the electric field. When the electric field is parallel to the metallic strips we should expect an inductive behaviour, but if the electric field is perpendicular to metallic strips we should expect a capacitive behaviour [10].

Considering the situation described in Figure 6 and the fact that the response of these structures present a dependency on frequency, it will be useful to predict what should happen when we have sources with low and high frequencies.

If we consider a source with low frequency, when the incident wave interacts with the structure it will be induced a current on the strips caused by the movement of the electrons. However, since the electric field of a long-wavelength source varies slowly, the electrons in the metal remain stationary for large periods of time and won't absorb energy. In this situation, it expected that just a small portion of the energy will be absorbed by the electrons, which will give rise to a high transmittance through the filter [8].

On the other hand, when we have a short-wavelength source the electric field of the same will vary faster, which means that electrons will constantly oscillating, and a large portion of the energy is absorved by the electrons. For this situation we should expect a low transmittance through the filter [8].

Therefore, this structure will behave as a low-pass filter since it transmits at low frequencies and blocks higher frequencies. Given this fact, the structure can be modeled as capacitor connected in shunt between the transmission line and the ground [8, 10].

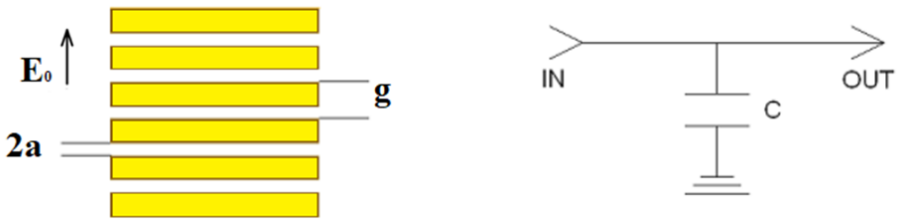

Figure 6 – Horizontal metallic strips and the corresponding equivalent circuit model, adapted from [8].

Considering the situation illustrated in Figure 7, it is also possible to predict the response of the structure as a function of the frequency of the source. Here, the polarization of the electric field is parallell and the expected behaviour will be slightly different, when compared to the previous situation. When we have a long-wavelength source, the interaction between the electric field and the electrons will cause the movement of the last ones along the strips. However, they can move without changing direction for large periods of time, which will lead to a large portion of energy being absorved by the electrons and therefore, the transmittance through the filter will be very low [8].

On the other hand, when we have a short-wavelength source the interaction between the electric field and the electrons will cause the fast movement of the last ones along the strips. Contrarly to the previous situation, the electrons will move short distances before changing its direction. Despite of their oscillation, they can't achieve the long-range motion of the previous case, which will lead to a small portion of energy being absorbed by the electrons and therefore, the transmittance through the filter will be very high [8].

This structure will behave as a high-pass filter since it transmits at high frequencies and blocks low frequencies. Given this fact, it can be modeled as an inductor connected in shunt between the transmission line and the ground [8, 10].

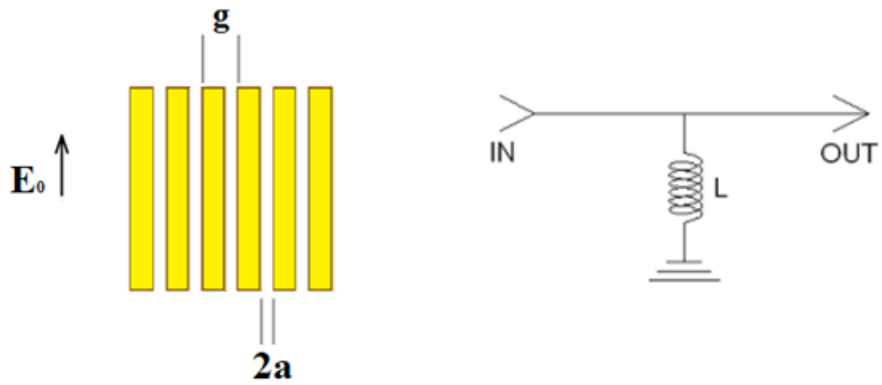

Figure 7 – Vertical metallic strips and the corresponding equivalent circuit model, adapted from [8].

It is possible to conclude that the main disadvantage of these structures is the polarization dependence, since the polarization of the electric field of the source can give rise to a capacitive or an inductive behaviour in the structures.

After understanding the operation of strip grating filters, now we can adress mesh filters. This type of filters presents two characteristic topologies, which can also be modeled using the same type of electric components of the previous cases as shown in Figure 8 and Figure 9 [11, 12]. Contrarly to the previous cases, this type of structures presents an interesting characteristic, since they are polarization independent. This means that the transmittance will not be affected if we decide to apply a 90º rotation to the structures [8, 10].

Considering the situation represented in Figure 8, Hooberman refers that electrons are confined to move through the squares when a plane wave interacts with the structure and, depending of the frequency of the source, we should expect a similar behaviour to what

happens in capacitive strip grating filters [8]. Therefore, we can conclude that this structure will behave as a low-pass filter.

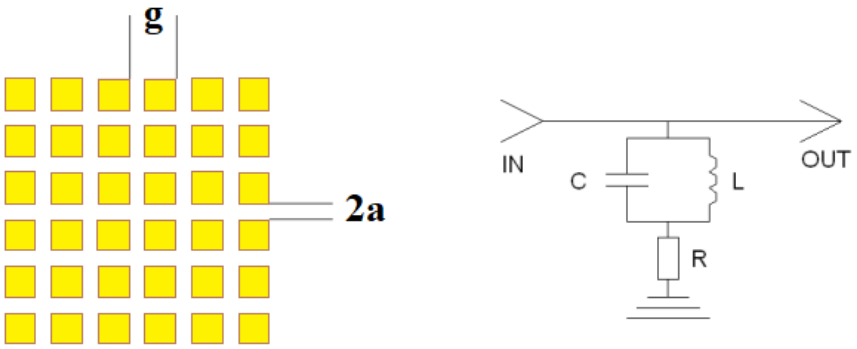

Figure 8 – Capacitive metallic mesh structure and its equivalent circuit model, adapted from [8].

On the other hand, when our focus is on the situation illustred in Figure 9, the autor mentions that the electrons will move along the structure when a plane wave interacts with the structure and, depending of the frequency of the source, we should expect a similar behaviour to what happens in inductive strip grating filters [8]. Here, we can conclude that this structure will behave as a high-pass filter.

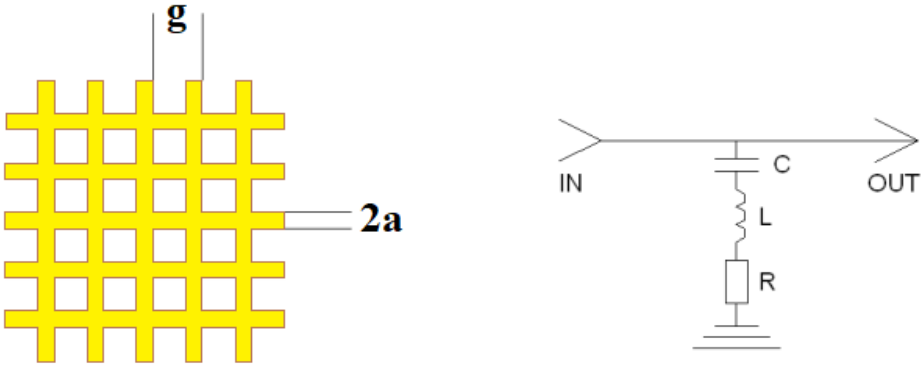

Figure 9 – Inductive metallic mesh structure and its equivalent circuit model, adapted from [8].

According to Hayt et all, we can use resistors to model the loss in the conductors in the circuits. However, when a low loss conductor is used for the circuit design, we can ignore this element in the analysis of the circuit [13].

Although there are several types of FSS designs, if we understand the models of the most basic classes of this type of filters, it will be possible to understand more easily the models that present more complex geometries. This becomes possible, since the principles discussed throughout this section may also apply to other geometries.

2.1.3. Applications

According to Rashid et all, FSSs structures can be employed in a wide range of applications, such as: spatial/angular filters, dichroic sub-reflectors, diplex/beam splitters, polarizers and circuit-analog absorbers [14]. Recently, the theory and designs of FSSs have also been extended to applications in metamaterials, planar lenses, reflectarray antennas, thermophotovoltaic systems and sensing systems [15].

Every application will have specific requirements, which should be achieved to allow its correct operation. However, when our focus is on the structure of the FSS, we realize that the most of these the applications mentioned above will require that the structure should function as as a close-to-ideal filter for plane waves, in order to obtain a flat in-band filtering response and a sharp out-of-band rejection skirt [14]. Furthermore, it is also desired that the frequency response becomes insensitive to the angle of incidence, which requires that the unit cell size becomes much smaller than the operating wavelength [16].

2.2. Structural Health Monitoring

In this sub-chapter will be discussed several topics related to structural health monitoring. This approach will focus on the contextualisation of the problems underlying this issue, on the discussion of methodologies for monitoring the health of structures, on the types of sensors used and, finally, on research in the field of applications in the region of frequency of THz.

2.2.1. Definition and Methods

The research related to structural health monitoring (SHM) began in the ‘80s and since then there have been major developments with the emergence of new technologies applied to this area [17]. Despite the wide range of methods and devices developed, they all serve a common purpose in identifying damages that affect the integrity of structures. However, this was not always the most used methodology to assess the integrity of the structure.

The reference methodology that has a long tradition in the identification of damages in components and structures is Non-Destructive Testing (NDT). Nowadays, this methodoly is commonly used for quality control processes employed by factories and inspections centers. Inspections are performed by an operator who evaluates structural integrity using external equipment (namely ultrasound, x-ray, etc.). According to Dwivedi

et all, sometimes more advanced non-destructuive methods are needed to assess the integrity of some structures. Methods such as eddy current testing, magnetic particle inspection, acoustic emission testing and among others, can be also employed for specific situations, in which we have some constrains related to the structure under analysis [18].

All methodologies present their limitations, and, in this case, there is no exception. In this particular case, the constraints mainly focus on the ability to analyze the structure as a whole and the cost / efficiency ratio. Unfortunately, most NDT-related methods can only detect macroscopic and surface/subsurface-level flaws. On the other hand, sometimes the efficiency of inspections depends on the accessibility of the component of the structure to be analyzed and on the expertise of the operators who are responsible for carrying out these inspections, which may entail substantially higher costs and more hours of work. Considering this type of limitations, it is natural to have a greater focus on other types of methodologies that allow collecting data to detect damages, locate them, know their extent and predict the useful life of the structures [19].

According to Doebling and Plankis, methods of detecting structures damage in SHM can be divided into two main categories, such as: global health monitoring and local health monitoring. This division becomes useful because, based on the type of data that the user wishes to obtain by monitoring the structure and the areas to be analyzed, the methods and devices that are best suited to the accomplishment of the previously established objectives can be chosen [19, 20].

The next two sections will discuss the existing methods in each category in order to understand what characterizes them and in what kind of situations they are used.

2.2.2. Global Health Monitoring Methods

When a structure suffers environmental and/or external disturbances that inflict damages, it is known at the outset that some characteristics of the structure will change because of this event. According to Plankis, the change of some characteristics such as the stiffness, mass, or damping properties of a structure will affect the global dynamic properties the structure [20].

Doebling et all refers that this kind of methods do not need a prior selection of the type and location of sensors in the structure. However, this is just possible if we don’t have any constrain related to critical areas, which need to be considered or complex structures [19].

Fan and Qiao affirmed that the related methods used in this type of methodoly of damage detection can also be classified in model-based method and responsed-based method. Model-based methods require prior availability of a detailed numerical model of the structure, but this is not always possible and therefore it becomes necessary to focus on approaches based on the experimental response of structures to external perturbations [21].

According to Doebling and Chang, the response-based methods are based on a large number of different modal properties of the structures and if we can understand how the relationship between the damage occurred and the changes in the dynamic properties of the structure, it will be easier to detect possible damage that may occur. These methods can be categorized as follows [19, 22]:

- Frequency-based methods;
- Mode shape-based methods;
- Curvatures mode shape-based methods;
- Other methods based on modal parameters;

Frequency-based methods focus on the most basic vibrational properties of a structure, since every structure can be characterized by unique resonant frequencies for each mode of vibration [19]. The occurrence of damages can be detected analysing the frequency shifts in resonant frequencies. However, it is not possible to locate them and for large structures the resonant frequencies at higher modes becomes more sensitive to minor damage. Despite the usefulness of these kinds of methods, it will be necessary to consider the usage of additional analysis techniques to locate and know the extent of the damages [20].

Mode shape-based methods are centered on the calculus of mode shape vectors. When a structure suffers a damage, dynamical properties such mass, stifness and damping are affected. Mode shape vectors can be calculated based on the change of these properties, which later will provide informations about both the existence and location of damage, which represents a major advantage in damage detection over the previous class of methods based on frequency [20]. However, this class of methods presents some important limitations such as the requirement of a considerable number of sensors in several locations to provide accurate measurements and the high sensivity to minor local damage at higher frequency modes of vibration [22].

Curvatures mode shape-based methods are more precise when compared to the previous class of methods based on mode shape vectors. The value of curvature can be obtained following two different types of approaches described in detail by Fan and Qiao in [21]. The first approach consists on differentiate the mode shape vectors twice and last one consists on fitting a localized cubic polynomial curve to the mode shape curvature and later calculate the difference between the cubic and the Laplacian [21]. This method can be used to detect damages, locate them and estimate their extent by studying the fluctuations in curvature values [20]. According to Plankis and Alvandi, these types of methods have also some disadvantages, namely, the calculation of the curvature from mode shape vectors can cause false readings and the requirement of a prior baseline data available about the structure [19, 23].

Doebling and other authors refers that there are several approaches based on modal parameters, but the most referred in context of global health monitoring are the dicamically calculation of the flexibility matrix and the calculation and later update of the matrices related to the modal parameters of the structure [19-23]. Through the first approach it is possible to estimate changes in the static behaviour of the structure and by studying the changes in the dynamic flexibility matrix over time, it will be possible to detect the existence and location of damages [23]. However, this method requires a prior calculations of mode shapes and it also necessary to know the frequencies related to the structure [19, 20].

The last method involves the update of some modal property matrices such as mass, stiffness, or damping. The damage detection, location and extension are achieved through the comparison between the updated and the original matrices [19]. However, this method estimates the updated modal matrices through constrained optimization techniques, based on equations of motion, the structural model, and the data obtained from measurements, which could lead to false damage results [20, 21].

2.2.3. Local Health Monitoring Methods

Local health monitoring methods allows the detection, location and progression of damages related to the components of the structure under analysis. In fact, Cawley refers that these types of methods are very useful for tracking the progress of damage already identified in routine NDT inspections and in the monitoring of known critical areas [24].

Despite the existence of several techniques, all of them are focused on basic concepts, such as [20]:

- Visual inspections;
- Strain or displacement monitoring;
- Acoustic properties monitoring.

Visual inspections are the oldest and simplest method to gauge whether there are problems in a component or structure. However, when it is necessary to recognize changes in a structure, it can sometimes be difficult to obtain a definitive prognosis. This happens mainly because of the concept of change that each individual has, which makes it possible to verify some subjectivity in the results of the inspection [20]. Given this limitation, we can say that this method alone will not be enough to know the current state of the structure.

On the other hand, the monitoring of strains or displacements constitutes a good option to know the current state of the structure. The materials constituting the structure suffer tensions and deflections in response as soon as the structure is loaded, which makes it possible to detect damage each time there are values higher than stipulated [20]. The measurement of both characteristics can be obtained using sensors suitable for this purpose such as strain gauges, but according to Kang et all it is also possible to estimate the displacement with a good degree of accuracy using fiber Bragg grating strain sensors [25]. This type of monitoring is mainly used in critical areas of a structure and if there is a problem already identified, it will be possible to know the progression of the damage based on the data from measurements [20, 25].

The detection and measurement of the progression of damages can be performed through the acoustic properties of materials. According to Nor, acoustic emission techniques can be used to detect, locate and assess damage that occurs in the structure caused by fatigue. The fundamental concept behind these methods are the stress waves, which travel from the acoustic emission source to the surface of the material, where they are captured by the accoustic emission sensors. The propagation of those waves is influenced by the existance of damages. After the acquisition and the analysis of data, we will be able to know all details about the current state of the material and the present damages [26].

2.2.4. Sensors

Sensors are devices that, when subjected to a physical / chemical stimulus, originate a specific response that can be transformed into another physical quantity for measurement/monitoring purposes. In the context of SHM systems, sensors represent a fundamental component of these systems, since they are responsible for the acquisition of data related to the health and integrity of the structures. According to Moreno-Gomez the sensors used in SHM can measure three different types of physical quantities: kinematic, mechanical and environmental [27]. The focus of this section will be the sensors related to the measurement of kinematic and mechanical quantities, since through these sensors we can measure accelarations, displacements, forces, strains and among others. The most common types of sensors used in SHM applications are [27]:

- Accelerometers;
- Acoustic emission sensors;
- Displacement sensors;
- Fiber optic sensors;
- Micro-Electro-Mechanical Systems (MEMS) sensors;
- Piezoelectric sensors;
- Strain Gauges;
- Tiltmeters and Inclinometers.

Accelerometers are devices that can measure the oscillatory movement experienced by a structure at a particular location [27]. Plankis also refers that its possible to calculate the frequency, damping, and mode shapes of a structure through the measurements performed by these devices [20]. According to Zhu et all, these devices represent a fundamental component of modern SHM systems. Despite the existence of various types of accelerometers, the most common type used by most of these systems is the low-power MEMS-based accelerometer [28]. The selection of this type of devices is due to the requirement of low power consumption that is of great importance in wireless sensor networks. However, these devices have a better resolution when compared to other types of accelerometers. Considering this limitation, Zhu et all proposed the creation of high-sensitivity wireless accelerometers for SHM in [28], which showed good agreement between the measurements and their working principle.

The acoustic emission sensors convert the stress waves generated by the mechanical deformation of the materials into proportional output electrical signals. According to Manthei, the most common type of acoustic emission sensors encompasses piezoelectric materials in their constitution, which gives them a high sensitivity at their resonant frequency. The author also mentions that through the measurements it possible to estimate other physical quantities such as the acceleration, displacement and velocity, which are demonstrated in [29]. This type of devices can also be used in a distributed way in the context of a sensor network to detect and locate the damages in the structures [20].

Displacement sensors can measure the displacement of targets relative to a fixed position. According to Li et all, the displacement can be also derived from acceleration and strain measurements by using numerical integration algorithms. However, this procedure is not as widely used because of the difficulty in setting initial and boundary conditions, which can lead to considerable errors [30]. The are several types of types of sensors, such as optical displacement sensors, linear proximity sensors, and ultrasonic displacement sensors, but the two most common sensors utilized to measure displacement are linear potentiometers and linear variable differential transformers due to their accuracy, simplicity in operation and low cost [27].

Fiber optic sensors are an emerging technology, which can be used for strain, deflection, acceleration, angular velocity, corrosion, and displacement estimation [27]. Although, the most referred sensors in literature are the fiber Bragg gratings sensors, there are several types of fiber optic sensors and they share a common working principle between them. The working principle consists on the sending of light beams in fiber optic cable at regular time intervals and later monitorization of the occurred changes in wavelength of reflected light sources [31]. These devices are very sensitive and accurate, but they present other interesting advantages, once they are immune to electromagnetic interference and to environmental factors, which is a common problem related to the other classes of sensors [27, 31].

MEMS sensors can convert a mechanical signal into an electrical signal and they are seen as a promising technology in the field of SHM [3]. According to Krüger et all, its inclusion in wireless monitoring systems becomes possible thanks to some exceptional characteristics such as: its easy integration into structures due to its miniature size, a low energy consumption, a high sensitivity, a good resolution and a low-cost manufacturing process for large-scale productions [32]. In fact, its application will allow to collect data

continuously to assess the condition of the structure and know when maintenance is needed.

Piezoelectric sensors work based on the principle of direct piezoelectric effect, i.e. conversion of mechanical energy into electrical form. According to Thomas et all, these devices present a low energy consumption and a reliable rate of strain measurement. On the other hand, these devices are also quite effective in damage identification procedure based on changes in vibration responses such as mass, damping and stiffness and they can also achive a better perfomance when compared to conventional strain gauges [33].

Strain gauges presents some interesting characteristics that make them suitable for SHM purposes, since they are inexpensive, easy to install and very sensitive to damages [34]. According to Plankis and Moreno-Gomez, these devices can measure the deformation of material at a specific location, but it is also possible to combine the information of all the sensors to detect damages in the structure [20, 27]. Choi also mentions that the use of these devices is very common in SHM, since it has been accumulated enough knowledge about the interpretation of strain data thanks to the research developed over the past few years [34].

Tiltmeters and inclinometers are the sensors utilized to measure the slope or tilt at a particular location. According to Ha et all, the combination of the slope or tilt measurements with existing data about acceleration, displacement and strain, allow the evaluation of the deflection suffered by the strucuture during structural deformation [35]. Plankis also mentions that the most common devices for SHM purposes are vibrating wire tiltmeters, electrolytic tiltmeters and inertial based inclinometers, whose choice higly depends on the applicatication requirements (i.e. sensivity to dynamic slope changes, long-term angular changes, among others) [20].

Through the research carried out in the literature, we conclude that SHM has experienced great developments with the emergence of new technologies applied to this area. These developments cover a number of innovations, both at the level of computational methods and at the level of sensors that allow the acquisition of data particularly relevant for health monitoring of structures. In the next section, it is intended to address the main innovations in the research context related to this topic for the THz domain.

2.2.5. Research involving THz

Research involving the field of THz has grown rapidly over the past few years and, since then, there has been an increasing demand for electronic and optoelectronic components of high quality from such field [2]. Nevertheless, these components cannot be easily manufactured from natural materials because these materials usually have no resonant response in the THz frequency region. Hence the manufacture of artificial materials is an increasingly relevant topic and, because of that, frequency selective surfaces (FSSs) are especially interesting due to their successful application in the field of microwave and millimeter waves [3].

Despite the wide range of available sensors for the SHM domain, there is some lack of highly selective sensors for the THz domain. However, the utilization of FSSs allows us to create compact devices, similar to MEMS sensors, which can be easily integrated in the surface of the structures without harming its functionality and aesthetic appearance, allowing mainly its monitoring [36, 37]. This is also possible, since FSSs and metamaterials can offer higher sensitivity and resolution compared to traditional structures and possess a greater potential to deliver strong enhancement and localization of fields, being especially suitable for the development of wireless strain sensors, that can operate in the microwave and terahertz ranges [38].

The application of THz technologies for SHM purposes is new and only a few articles have been published on this topic. In addition to the articles concerning sensors, there is also a significant number of articles that refer to the use of THz imaging as the primary method for detecting structural damages [39]. Another interesting tecnhology based on THz is described by Chen and Kaushik in [40], where these authors propose a terahertz sensor that measures vibrations behind optically opaque barriers. Through the usage of this sensor, the displacement in response to some change in mechanical loading can be measured and surface irregularities can be found.

The development of THz stress sensors based on metal mesh filters is introduced by M. A. Ribeiro and W.J. Otter in [36, 41]. The resonances of this type of sensors are obtained thanks to the periodicity of the unit cells in the direction of propagation and, according to the authors, can be used in several applications, including SHM. After their integration into the structures, it will be possible to detect cracks and other damages,

since, in this type of critical situations, it is known that the transmittance values of the sensors decrease abruptly.

Chapter 3 – Circuit Theory and Design

In this chapter we introduce the required tools to design a highly selective structure (HSS) which is the centrepiece of the sensor. This HSS works as a reconfigurable selective THz window in which only radiation of certain desired frequencies is allowed to pass. In literature, as described in [36, 41], this kind of behaviour is obtained using a technique related to the periodicity of the unit cells of the structure in the direction of propagation. Here we shall use a much simpler approach where the resonances are obtained from the propagation in a THz HSS composed by two arrays of gold wires. Our goal is therefore to study the interaction between the waves and the wires to be able to understand the principle of operation of the HSS. More specifically we intend to devise a method to optimize the relation between the radius of the wires and the distance between wires and adjust the sensitivity and responsiveness of the sensor. Moreover, for simplicity, we shall assume that the structure is composed by two arrays of wires because using a single array would not be sufficient to achieve the desired degree of selectivity.

3.1. Microwave Network Analysis

According to D. M. Pozar, the basic procedure for microwave network analysis is as follows [42]:

- To treat a set of basic, canonical problems rigorously, using field analysis and Maxwell's equations (e.g. transmission line and waveguide problems);
- To obtain quantities that can be directly related to a circuit or transmission line parameter (e.g. propagation constant and characteristic impedance), since this allows a transmission line or a waveguide to be treated as an idealized distributed component characterized by its length, propagation constant, and characteristic impedance;
- To interconnect the various components and to use network and/or transmission line theory to analyze the behaviour of the entire system of components, including effects such as multiple reflections, loss, impedance transformations, and transitions from one type of transmission medium to another.

In microwave theory there are four type of parameters which are commonly used to analyse the propagation of waves in devices. Impedance (Z) and admittance (Y) parameters are related to the total voltages and currents at the ports while scattering parameters (S) are related to the equivalent voltages of the waves entering and exiting the device. In high frequency propagation studies, it is common to use S parameters because they are directy related to the waves travelling in the structure. S parameters, however, are not easy to use when studying cascading structures, which are very common in network analysis. It is therefore very useful to use another type of parameters, called ABCD parameters, and define transformations that allow the conversion of S parameters into ABCD parameters and vice versa [42].

3.1.1 S Parameters

S parameters are especially useful for describing interactions of voltage and current waves with a multi-port network. Let us consider the N-port network of Figure 10, which is introduced by Pozar in [42]:

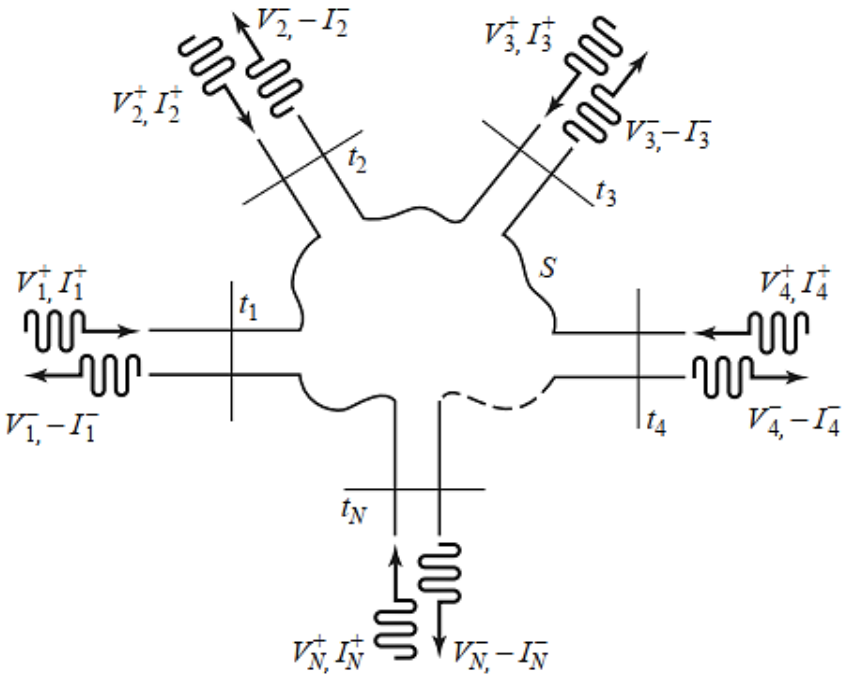

Figure 10 – Representation of an arbitray N-port microwave network introduced by Pozar, taken from [42].

where V_N^+ is the amplitude of the incident voltage wave at port N, V_N^- is the is the amplitude of the reflected voltage wave at port N, I_N^+ is the amplitude of the incident current wave at port N, I_N^- is the is the amplitude of the reflected current wave at port N and t_n terminal plane at port N. Here, the convection assumed is e^{jwt}.

According to Pozar, the scattering matrix [S] of this microwave network can be defined in relation to the incident and reflected voltage waves, which is expressed as [42],

$$\begin{bmatrix} V_1^- \\ \vdots \\ V_N^- \end{bmatrix} = \begin{bmatrix} S_{11} & \cdots & S_{1N} \\ \vdots & \ddots & \vdots \\ S_{N1} & \cdots & S_{NN} \end{bmatrix} \cdot \begin{bmatrix} V_1^+ \\ \vdots \\ V_N^+ \end{bmatrix} \quad (1)$$

or,

$$[V^-] = [S][V^+] \quad (2)$$

where $[V^+]$ and $[V^-]$ are vectors whose components are the amplitudes of the incident and reflected voltage waves at the ports. It is also possible to determine each element of the scattering matrix through the following expression [42],

$$S_{ij} = \left.\frac{V_i^-}{V_j^+}\right|_{V_k^+=0,\ \forall k \neq j} \quad (3)$$

in this expression the wave amplitude ratio is defined "from" port *j* "to" port *i.* The author also mentions that the incident waves on all ports except the j^{th} port must be set to zero, since all ports should be terminated in matched loads to avoid reflections [42].

3.1.2 ABCD Parameters

ABCD parameters are especially useful for cascading two-port networks. In the literature, this set of parameters is commonly known as ABCD matrix or transmission matrix. Let us consider the two-port network of Figure 11, which is introduced by Pozar in [42]:

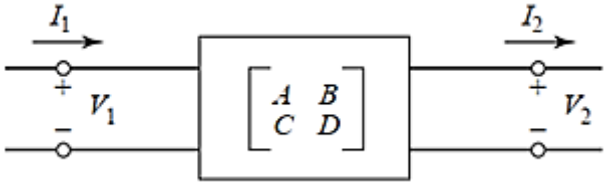

Figure 11 – Representation of the two-port network containing a transmission matrix, taken from [42].

The transmission matrix of this two-port network can be expressed as [42]

$$\begin{bmatrix} V_1 \\ I_1 \end{bmatrix} = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} V_2 \\ I_2 \end{bmatrix} \quad (4)$$

where V_1 is the voltage across port 1, I_1 is the current into port 1, V_2 is the voltage across port 2 and I_2 is the current out of port 2. Each entry of the ABCD matrix can be expressed as follows [43]:

$$A = \left.\frac{V_1}{V_2}\right|_{I_2=0}, B = \left.\frac{V_1}{I_2}\right|_{V_2=0}$$
$$C = \left.\frac{I_1}{V_2}\right|_{I_2=0}, D = \left.\frac{I_1}{I_2}\right|_{V_2=0} \quad (5)$$

According to Whites, the ABCD matrix of a cascade connection of two or more two-port networks can be characterized by simply multiplying their ABCD matrices [43]. If we cascade two two-port networks as shown in Figure 12 a) the system is equivalent to a single two-port network as shown in Figure 12 b) where the resulting ABCD matrix is the product of the ABCD matrices of the original networks.

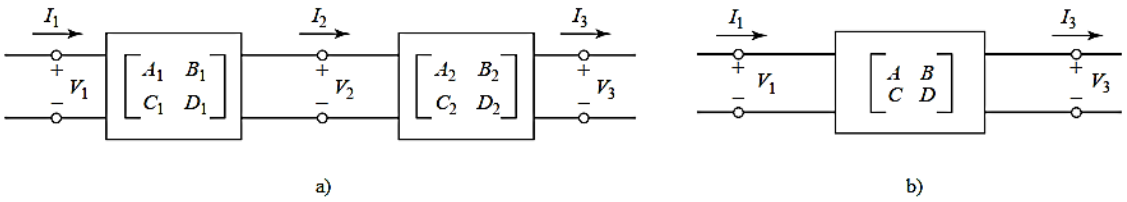

Figure 12 – Representation of a) a cascade connection of two-port networks and b) its equivalent network circuit, taken from [42].

In fact, we have

$$\begin{bmatrix} V_1 \\ I_1 \end{bmatrix} = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} V_3 \\ I_3 \end{bmatrix}, \quad (6)$$

where,

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} = \begin{bmatrix} A_1 & B_1 \\ C_1 & D_1 \end{bmatrix} \begin{bmatrix} A_2 & B_2 \\ C_2 & D_2 \end{bmatrix} \quad (7)$$

Pozar and Whites also mention that the order of matrix multiplication must be the same as the order in which the two ports are arranged, since matrix multiplication is not commutative, in general [42, 43].

3.1.3 Conversion between ABCD and S Parameters

Let us introduce three main components of network analysis and their transmission matrices (ABCD) [42]:

1) Transmission line (TL)

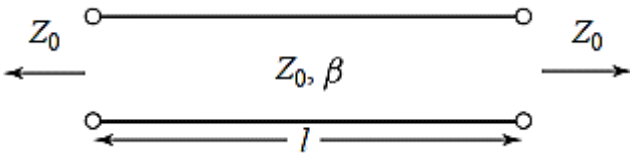

Figure 13 – Representation of a transmission line circuit, taken from [42].

which has the following transmission matrix associated:

$$T = \begin{bmatrix} \mathrm{A} = \cos(\beta l) & \mathrm{B} = \mathrm{j}\, Z_0 \sin(\beta l) \\ \mathrm{C} = \mathrm{j}\frac{1}{Z_0}\sin(\beta l) & \mathrm{D} = \cos(\beta l) \end{bmatrix} \quad (8)$$

where l is the length of the line, Z_0 its characteristic impedance and β the wavenumber.

2) Admittance (Y)

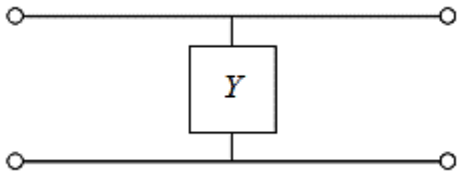

Figure 14 – Representation of admittance circuit, taken from [42].

For the admittance circuit its transmission matrix can be written as:

$$T = \begin{bmatrix} \mathrm{A} = 1 & \mathrm{B} = 0 \\ \mathrm{C} = \mathrm{Y} & \mathrm{D} = 1 \end{bmatrix} \quad (9)$$

3) T network

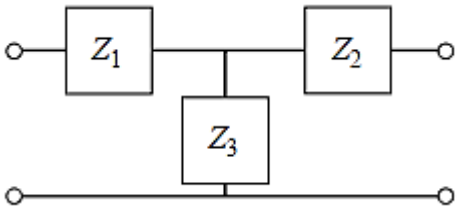

Figure 15 – Representation of impedance circuit, taken from [42].

Finally, in this situation, the transmission matrix can be expressed as:

$$T = \begin{bmatrix} \mathrm{A} = 1 + \frac{Z_1}{Z_3} & \mathrm{B} = Z_1 + Z_2 + \frac{Z_1 Z_2}{Z_3} \\ \mathrm{C} = \frac{1}{Z_3} & \mathrm{D} = 1 + \frac{Z_2}{Z_3} \end{bmatrix} \quad (10)$$

The conversion between 2-port network parameters is as follows [42]:

1) ABCD to S parameters

$$S_{11} = \frac{A + {}^{B}/_{Z_0} - CZ_0 - D}{A + {}^{B}/_{Z_0} + CZ_0 + D}, \qquad S_{12} = \frac{2(AD - BC)}{A + {}^{B}/_{Z_0} + CZ_0 + D}$$

$$S_{21} = \frac{2}{A + {}^{B}/_{Z_0} + CZ_0 + D}, \qquad S_{22} = \frac{-A + {}^{B}/_{Z_0} - CZ_0 + D}{A + {}^{B}/_{Z_0} + CZ_0 + D} \quad (11)$$

2) S to ABCD parameters

$$A = \frac{(1 + S_{11})(1 - S_{22}) + S_{12}S_{21}}{2S_{21}}, \qquad B = z_0 \frac{(1 + S_{11})(1 + S_{22}) - S_{12}S_{21}}{2S_{21}}$$

$$C = \frac{1}{z_0} \frac{(1 - S_{11})(1 - S_{22}) - S_{12}S_{21}}{2S_{21}}, \qquad D = \frac{(1 - S_{11})(1 - S_{22}) + S_{12}S_{21}}{2S_{21}} \quad (12)$$

Using these conversion rules, it is possible to obtain the S parameters for the TL shown above:

$$S = \begin{bmatrix} 0 & e^{-j\beta l} \\ e^{-j\beta l} & 0 \end{bmatrix} \quad (13)$$

In this situation, it is assumed that the line is taken from both sides with an impedance Z_0. In general, however, the characteristic impedance of the line is not the same as the connection networks and the resulting expressions are more complicated. To analyse this situation, consider the circuit shown in Figure 16. We consider two infinitesional lines connecting the line of impedance Z_0 to outside circuits of impedances Z_1.

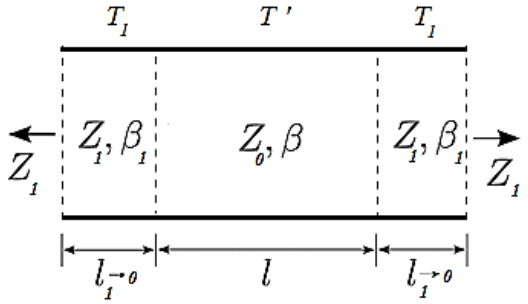

Figure 16 – Representation of non-adapted transmission line circuit.

We have $T = T_1\, T'\, T_1$, where

$$T_1 = \begin{bmatrix} \cos(\beta_1 l_1) & \mathrm{j}Z_1 \sin(\beta_1 l_1) \\ \mathrm{j}\frac{1}{Z_1}\sin(\beta_1 l_1) & \cos(\beta_1 l_1) \end{bmatrix} \xrightarrow[l_1 \to 0]{} T_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad (14)$$

and therefore $T = T'$. In order to carry out the conversion to the S parameters it is necessary to consider the use of the impedance Z_1 instead of the impedance Z_0 in the conversion formulas given above. Therefore S_{11} can be expressed as:

$$S_{11} = \frac{\cos(\beta l) + \mathrm{j}\frac{Z_0}{Z_1}\sin(\beta l) - \mathrm{j}\frac{Z_1}{Z_0}\sin(\beta l) - \cos(\beta l)}{\cos(\beta l) + \mathrm{j}\frac{Z_0}{Z_1}\sin(\beta l) + \mathrm{j}\frac{Z_1}{Z_0}\sin(\beta l) + \cos(\beta l)} \quad (15)$$

We shall introduce here a normalized impedance $z_0 = Z_0 / Z_1$ and proceed in the same way to find the expressions for S_{12}, S_{21} and S_{22}. The following expressions for the S parameters are obtained:

$$S_{11} = \frac{\mathrm{j}(z_0 - {z_0}^{-1})\sin(\beta l)}{2\cos(\beta l) + \mathrm{j}(z_0 + {z_0}^{-1})\sin(\beta l)}, \qquad S_{12} = \frac{2}{2\cos(\beta l) + \mathrm{j}(z_0 + {z_0}^{-1})\sin(\beta l)} \quad (16)$$

$$S_{21} = \frac{2}{2\cos(\beta l) + \mathrm{j}(z_0 + {z_0}^{-1})\sin(\beta l)}, \qquad S_{22} = \frac{\mathrm{j}(z_0 - {z_0}^{-1})\sin(\beta l)}{2\cos(\beta l) + \mathrm{j}(z_0 + {z_0}^{-1})\sin(\beta l)}$$

We conclude that $S_{11} = S_{22}$ and $S_{12} = S_{21}$, which was expected because the structure is symmetric, lossless and reciprocal [42]. However, in contrast to the adapted line, S_{11} and S_{22} are generally different from zero and we expect reflections in this structure.

3.1.4 Reflectance and Transmittance Concept

Now, we shall introduce the concepts of reflectance and transmittance. The scattering parameters are defined as relations between fields (or equivalent voltages) and do not provide a direct interpretation of the power that is reflected and transmitted by the structure. This is the purpose of reflectance (*R*) and transmittance (*T*) which can be defined through the S parameters as follows

$$R = |S_{11}|^2\,, \quad (17)$$

and

$$T = |S_{12}|^2. \quad (18)$$

For the non-adapted line studied in the last section, it follows that:

$$R = \frac{(z_0 - z_0^{-1})^2 \sin^2(\beta l)}{4\cos^2(\beta l) + (z_0 + z_0^{-1})^2 \sin^2(\beta l)}, \quad (19)$$

and

$$T = \frac{4}{4\cos^2(\beta l) + (z_0 + z_0^{-1})^2 \sin^2(\beta l)}. \quad (20)$$

The expression of absorption as a function of reflectance and transmittance can also be defined as

$$A = 1 - R - T = 1 - (R + T) \quad (21)$$

It can easily be shown that $A = 0$, which implies that the structure does not absorb energy because it has no losses.

The expressions for R and T can be rewritten more compactly, so that:

$$R = \frac{X-2}{X+2}, \quad (22)$$

and

$$T = \frac{4}{X+2}. \quad (23)$$

where

$$X = 2\cos^2(\beta l) + (z_0^2 + z_0^{-2})\sin^2(\beta l) \quad (24)$$

As expected,

$$R + T = 1 \rightarrow A = 0 \quad (25)$$

To understand the evolution of R and T in terms of the parameters β, l and z_0, let us first note that the function X is a periodic function with a period characterized by the following equation:

$$X(g) = X(g + \pi), \; g = \beta l. \quad (26)$$

Which in turn means that R and T will also have a periodicity π in g, i.e.,

$$R(g) = R(g + \pi), \; T(g) = T(g + \pi). \quad (27)$$

When analyzing the behaviour of X with g, it is possible to identify some interesting cases:

- $z_0^2 + z_0^{-2} = 2$

In this case, $\frac{{Z_0}^2}{{Z_1}^2}+\frac{{Z_1}^2}{{Z_0}^2}=2 \rightarrow Z_0=Z_1$. This corresponds to the simplest case studied above for the adapted line, such that

$$R=0, \quad T=1. \tag{28}$$

- $Z_0 \neq Z_1$

In this case we always have

$${z_0}^2+{z_0}^{-2}>2$$

In fact, by writing $Z_0 = a\,Z_1$ we get ${z_0}^2+{z_0}^{-2}=a^2+\frac{1}{a^2}$. The derivative of this function is $2a-2a^{-3}$ from which we can conclude that ${z_0}^2+{z_0}^{-2}$ is a monotonically decreasing function for $0<a<1$ and monotonically increasing for $1<a<\infty$. Therefore, it takes its minimum value of 2 at $a=1$.

The expression for *X* given above can be written as,

$$X=[2-({z_0}^2+{z_0}^{-2})]\cos^2(g)+({z_0}^2+{z_0}^{-2}) \tag{29}$$

from which we clearly see that *X* will take values in the interval [2, ${z_0}^2+{z_0}^{-2}$]. The reflectance (*R*) and transmittance (T) in terms of g are shown in the graphs of Figure 17 and Figure 18.

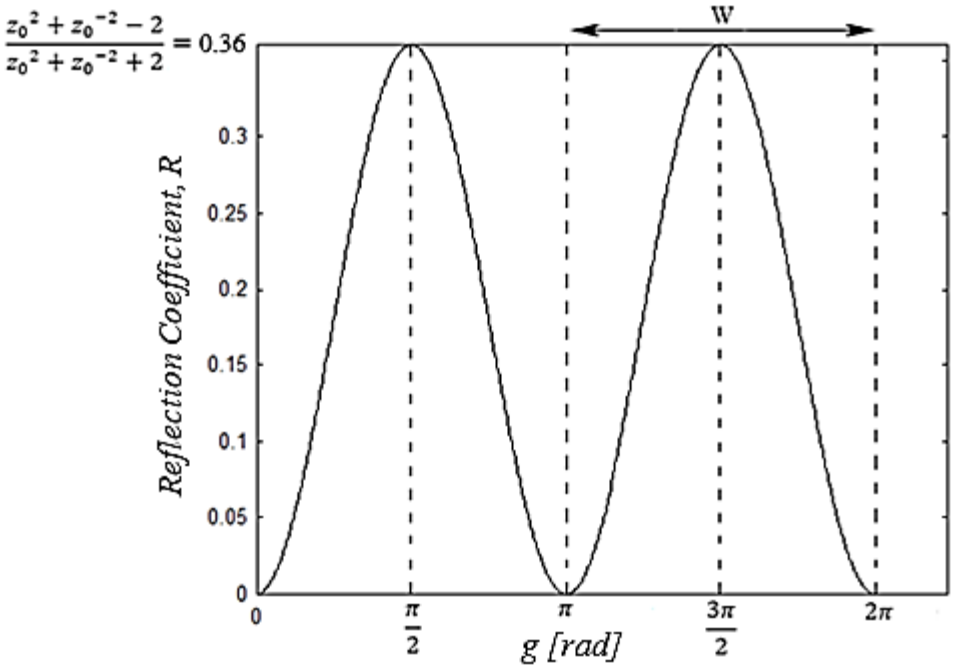

Figure 17 – Graphical representation of the reflectance as function of g for z_0 = 0.5.

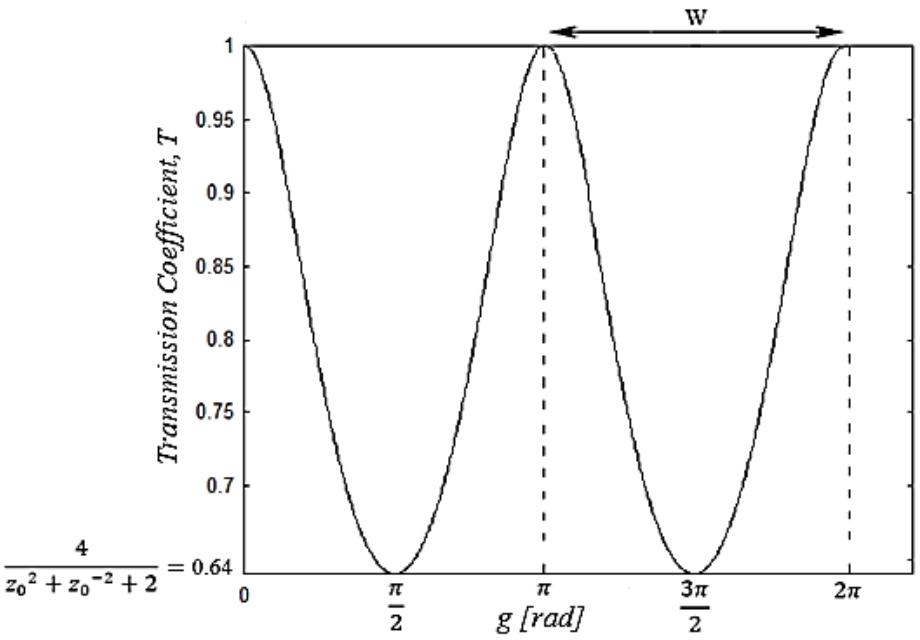

Figure 18 – Graphical representation of the transmittance as function of g for z_0 = 0.5.

From the two figures above, we can draw an important comment that is related to the choice of materials for the slab. The intrinsic impedance of the slab will affect diretly the dynamic range of the transmittance and reflectance because they have respectively minimum and maximum values given by

$$R_{max} = \frac{{z_0}^2 + {z_0}^{-2} - 2}{{z_0}^2 + {z_0}^{-2} + 2}, \qquad T_{min} = \frac{4}{{z_0}^2 + {z_0}^{-2} + 2}. \tag{30}$$

and hence the choice of z_0 is important. We have $Z_0 = Z_1/\sqrt{\varepsilon_r}$ such that the bigger the value of ε_r the bigger the value of ${z_0}^2 + {z_0}^{-2}$ and so more dynamic range, that is, R_{max} approaches 1 and T_{min} approaches 0.

3.2. Wave Propagation Study

In this sub-chapter will be studied the interaction between the electromagnetic waves and the arrays of wires that constitute the sensor. Its operating principle will also be addressed, in order to understand how it will be possible to adjust its sensitivity and selectivity under certain conditions.

In a first approach, we will analyze what happens to the propagation of electromagnetic waves on a dielectric slab and later we will analyze what happens in terms of propagation phenomena when the arrays of wire are added to the slab. This last scenario represents the filter of the proposed THz sensor.

3.2.1 Electromagnetic wave propagation on a dielectric slab

Consider the following example of Figure 19, in which we have a dielectric slab of width *l*, with a characteristic impedance *Z* and a phase velocity *vp*:

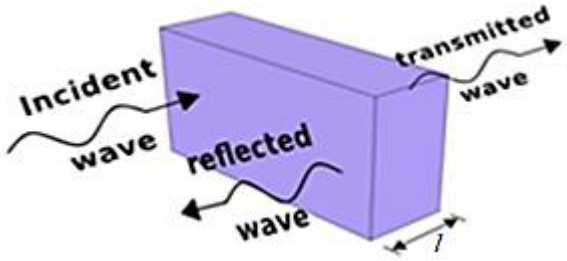

Figure 19 – Dielectric Slab.

A numerical simulation of the dielectric slab can be carried out with the purpose of trying to corroborate the aforementioned theory and to this end a dielectric constant $\varepsilon_r = 5.5$, a magnetic permeability $\mu_r = 1.0$ and a width *l* = 1 cm were assumed as relevant parameters.

Through this procedure we can determine the impedance of the slab, its phase velocity, the reflectance and transmittance. We determined that:

$$Z = Z_0\sqrt{\frac{\mu_r}{\varepsilon_r}} = 160.7\Omega \leftrightarrow z = \frac{Z}{Z_0} = 0.4264, \quad (31)$$

$$vp = \frac{c}{n} \leftrightarrow vp = \frac{c}{\sqrt{\varepsilon_r \mu_r}} = 1.28 \times 10^8 \text{ m/s}, \quad (32)$$

$$R_{max} = \frac{{z_0}^2 + {z_0}^{-2} - 2}{{z_0}^2 + {z_0}^{-2} + 2} \leftrightarrow R_{max} = \frac{X_{man} - 2}{X_{man} + 2} = 0.4793, \quad (33)$$

$$T_{min} = \frac{4}{{z_0}^2 + {z_0}^{-2} + 2} \leftrightarrow T_{min} = \frac{4}{X_{man} + 2} = 0.5207. \quad (34)$$

It is possible to find the expression for the bandwidth of reflection and transmition, assuming that $g = \pi$, and through this assumption we have that:

$$\beta l = \pi \leftrightarrow \frac{w}{vp} l = \pi \leftrightarrow \frac{2\pi f}{vp} l = \pi \leftrightarrow f = \frac{vp}{2l}. \quad (35)$$

As *l* is the width of the slab, the final expression for the bandwidth is:

$$w = \frac{vp}{2l}, \quad (36)$$

and according with our calculations the bandwith *w* will be:

$$w = 6.396 \text{ GHz}. \quad (37)$$

Figure 20 shows the variation of reflectance and transmittance coefficients, which were obtained based on a numerical simulation considering a frequency sweep that starts at 0.005 GHz and finishes at 40 GHz, with a step size of 0.005 GHz.

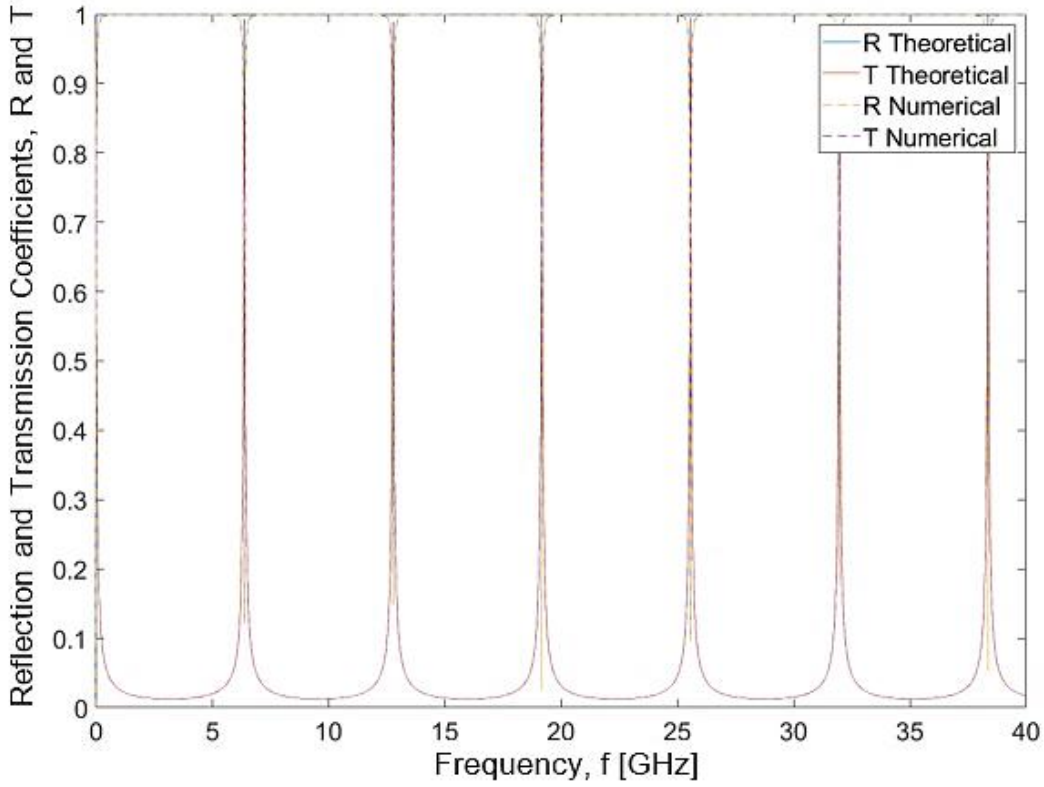

Figure 20 – Evolution of the curves of R and T for propagation in a dielectric slab.

After this simulation was carried out, a clear idea about the effects of each parameter of the slab was obtained:

- Bandwidth control - the larger the width *l*, the smaller the bandwidth *w*. As a consequence of this phenomenon the resonances of *T* and *R* will be closer to each other.
- Dynamic range control - the greater the characteristic impedance of the slab *Z*, the greater the dynamic range that can be achieved. This fact justified to the extent that when *Z* tends to infinity the maximum of reflectance (R_{max}) will tend to 1 and the minimum of the transmittance (T_{min}) will tend to 0.

Comparing the results obtained with the theory, it is shown that it worked well for the case under analysis.

3.2.2 Propagation in a slab loaded with two admittances: the filter of the sensor

Let us now consider the situation shown in Figure 21, where we have the equivalent circuit for a dielectric slab loaded with two arrays of wires. The arrays of wires are described as admittances *Y* in the equivalent circuit and the slab by three sections of transmission line with equal length *l*/3 (total length *l*). In fact, in the following sections we will demonstrate that an array of wires of radius *a* and distance between elements *d* is in fact well described as an admittance *Y* and the expression for *Y* will be given. Our goal is to study this structure with the wire inclusions and demonstrate its working principle. This structure is the main component of the sensor. Once we have an expression for *Y* we will then analyse the structure in order to determine its reflection and transmission frequency response.

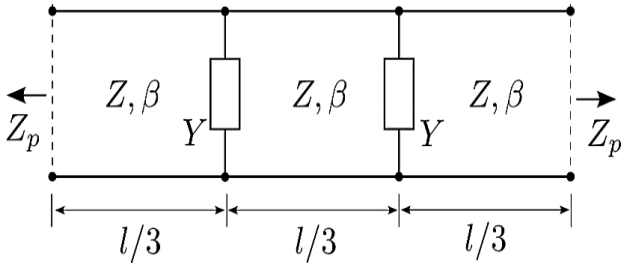

Figure 21 – Equivalent circuit of dielectric slab containing two arrays of wires, which represents the proposed THz filter.

3.2.3 Maxwell's Equations

In all wave propagation problems, where there are transitions between media, we can always identify incident waves at the interface which give rise to reflected and transmitted waves. The major goal of this study is to understand this kind of interactions between electromagnetic waves and the array of wires.

The four fundamental equations that describe the propagation of electromagnetic fields are Maxwell's equations. This is a set of four equations, namely, the Gauss's law, the Faraday's law, the Gauss's law for magnetic fields and the Ampère-Maxwell's law, represented as follows:

$$\nabla \cdot \boldsymbol{E} = \frac{\rho}{\varepsilon_0} \quad (38)$$

$$\nabla \times \boldsymbol{E} = -\frac{\partial \boldsymbol{B}}{\partial t} \quad (39)$$

$$\nabla \cdot \boldsymbol{B} = 0 \quad (40)$$

$$\nabla \times \boldsymbol{B} = \mu_0 \mathbf{J} + \frac{1}{c^2}\frac{\partial \boldsymbol{E}}{\partial t} \quad (41)$$

where **E** is the electric field, **B** is the magnetic field and *c* is the speed of light in vaccum. The remaining quantities, ρ and **J** are the charge density and current density, respectively. In a vacuum, these densities are trivialy zero because there are no free charges and currents. In a more general case, however, the current density depends on the motion of charges, which is determined throught the laws of mechanics. The electromagnetic force on a particle wih charge q is the Lorentz Force,

$$\boldsymbol{F} = q(E + v \times B) \quad (42)$$

It is indeed possible to include the dynamics of charges, thorough the above formula, and for non-relativistic particles, into Maxwell equations but the problem is most often difficult to solve on a microscopic background. However, the relevant range of distances at the experimental level is several orders of magnitude greater than the particles that constitute the materials. We call this the macroscopic perspective and is the approach followed in this dissertation. In this approach we convenientely include the fields of charges and polarized currents in the atoms in two new fields **D** and **H** and, for these fields, the Maxwell's equations can be rewritten as follows:

$$\nabla \cdot \boldsymbol{D} = \rho \quad (43)$$

$$\nabla \times \boldsymbol{E} = -\frac{\partial \boldsymbol{B}}{\partial t} \quad (44)$$

$$\nabla \cdot \boldsymbol{B} = 0 \quad (45)$$

$$\nabla \times \boldsymbol{H} = \mathbf{J}_0 + \frac{\partial \boldsymbol{D}}{\partial t} \quad (46)$$

These equations describe the electric and magnetic phenomena at the macroscopic level and to solve them, it is necessary to relate the electric displacement **D** and the magnetic field **H** with the electric field and the magnetic induction. The simplest case is that of isotropic and linear medium in which the **D** and **H** fields are parallel and directly proportional to **E** and **B**:

$$\boldsymbol{D} = \varepsilon \boldsymbol{E}, \qquad \boldsymbol{B} = \mu \boldsymbol{H}, \quad (47)$$

where ε is the electrical permittivity of the medium and μ is the magnetic permeability.

So far, in discussing Maxwell's Equations and the equations derived from them, we have considered arbitrary time dependence. According to Cheng, the time dependence of the mentioned field quantities depends on the source functions ρ and **J** [44].

Sinusoidal or co-sinusoidal time functions are commonly used in engineering to represent functions and signals as a superposition of basic waves. However, these functions or signals require a specialized treatment for each of its components in the generalization processes. Therefore, Cheng mentions that periodic components can be expanded in Fourier series, whereas non-periodic components should be expressed as Fourier integrals [44]. This principle can also be applied to electromagnetic fields in which for source functions present an arbitrary time dependence, since Maxwell's equations are linear differential equations. This is a great advantage, since a sinusoidal excitation of the source will give rise to a sinusoidal variation of the electromagnetic fields with the same frequency in a steady state [44].

The phasors represent complex quantities, which are characterized by an amplitude, an angular frequency and an initial phase, both time-invariant. Given these features, Cheng states that field vectors can be represented by vector phasors that depend on the coordinates of space, but not in time [44]. For the time-harmonic electromagnetic fields, the convection assumed in this dissertation for general time variation is $e^{j\omega t}$. As an example, the electric field referring to a cosine function can be written in the phasor form as

$$\vec{E}(x,y,z,t) = Re\left[\vec{E}(x,y,z)e^{j\omega t}\right] \quad (48)$$

where $\vec{E}(x,y,z)$ is a vector phasor containing the information about the direction, magnitude and phase. The time rate of change of electric fied vector can also be expressed using this notation as

$$\frac{\partial \vec{E}(x,y,z,t)}{\partial t} = Re\left[j\omega\vec{E}(x,y,z)e^{j\omega t}\right] \quad (49)$$

It is possible to conclude that if the electric field vector $\vec{E}(x,y,z,t)$ is represented in the phasor form as $\vec{E}(x,y,z)$, then the time rate of change of electric fied vector can be represented by the phasor $j\omega\vec{E}(x,y,z)$. On the other hand, the integral of the electric field vector can be represented by the phasor $\frac{\vec{E}(x,y,z)}{j\omega}$. According to Cheng, the higher order

differentiations and integrations with respect to *t* can be represented by multiplications and divisions of the phasor $\vec{E}(x, y, z)$ by higher powers of $j\omega$ [44].

Considering the simplest case of an isotropic and linear medium, in which we have the field phasors $(\vec{E},\ \vec{H})$ and source phasors $(\rho,\ \vec{J})$, Maxwell's equations can be expressed in the phasor form as:

$$\nabla \cdot \vec{E} = \rho \quad (50)$$

$$\nabla \times \vec{E} = -j\omega\mu\vec{H} \quad (51)$$

$$\nabla \cdot \vec{H} = 0 \quad (52)$$

$$\nabla \times \vec{H} = \vec{J} + j\omega\varepsilon\vec{E} \quad (53)$$

In this study we considered the Maxwell's equations to describe the electric and magnetic phenomena at the macroscopic level, without variation on *z* axis and considering that current density $\boldsymbol{J}_0 = 0$ and charge density $\rho = 0$ (no sources). Starting from the vectorial equations, such as:

$$\nabla \cdot \vec{D} = 0 \quad (54)$$

$$\nabla \times \vec{E} = -\frac{\partial \vec{B}}{\partial t} \quad (55)$$

$$\nabla \cdot \vec{B} = 0 \quad (56)$$

$$\nabla \times \vec{H} = \frac{\partial \vec{D}}{\partial t} \quad (57)$$

The fields $\vec{E}$, $\vec{B}$, $\vec{D}$ and $\vec{H}$ generally depend on *x*, *y*, *z* and t variables. In this case, it is assumed that $\frac{\partial}{\partial z} = 0$, which implies that such fields will only depend on *x*, *y* and *t*.

Through these assumptions the Maxwell's equations can be solved and rewritten resulting in two sets of three equations related to the transverse electric (TE) and transverse magnetic (TM) modes, since it is known that **E** and **B** fields are perpendicular between them and they are also perpendicular to the propagation direction.

For the TE mode the following equations were obtained:

$$\frac{\partial E_y}{\partial x} - \frac{\partial E_x}{\partial y} = -\mu\frac{\partial H_z}{\partial t} \quad (58)$$

$$-\frac{\partial H_z}{\partial x} = \varepsilon \frac{\partial E_y}{\partial t} \quad (59)$$

$$\frac{\partial H_z}{\partial y} = \varepsilon \frac{\partial E_x}{\partial t} \quad (60)$$

and for the TM mode the following equations were obtained:

$$\frac{\partial H_y}{\partial x} - \frac{\partial H_x}{\partial y} = \varepsilon \frac{\partial E_z}{\partial t} \quad (61)$$

$$\frac{\partial E_z}{\partial x} = \mu \frac{\partial H_y}{\partial t} \quad (62)$$

$$\frac{\partial E_z}{\partial y} = -\mu \frac{\partial H_x}{\partial t} \quad (63)$$

3.2.4 Wave Equation

Through the two sets of equations referring to the TE and TM modes, it is possible to deduce the wave equation associated with each mode. The wave equation for the TE mode, which only depends on the fields E_x, E_y and H_z, is obtained in terms of H_z as:

$$\frac{\partial^2 H_z}{\partial x^2} + \frac{\partial^2 H_z}{\partial y^2} - \mu\varepsilon \frac{\partial^2 H_z}{\partial t^2} = 0 \quad (64)$$

and the wave equation for the TM mode, which only depends on H_x, H_y and E_z, is:

$$\frac{\partial^2 E_z}{\partial x^2} + \frac{\partial^2 E_z}{\partial y^2} - \mu\varepsilon \frac{\partial^2 E_z}{\partial t^2} = 0 \quad (65)$$

Thus, since both equations have the same form and considering that in a vacuum $\mu = \mu_0$ and $\varepsilon = \varepsilon_0$, the following equation can be written:

$$\frac{\partial^2 \Psi}{\partial x^2} + \frac{\partial^2 \Psi}{\partial y^2} - \frac{1}{c^2} \frac{\partial^2 \Psi}{\partial t^2} = 0 \quad (66)$$

where c is the wave speed, which can be expressed as $c = \frac{1}{\sqrt{\epsilon_0 \mu_0}}$ and Ψ can be E_z or H_z.

3.2.5 Helmoltz Equation

The Helmholtz equation, the time-invariant form of the wave equation, is a partial differential equation that must be solved to obtain the solution of an electromagnetic boundary value problem [44].

Assuming that $\Psi(x, y, t)$ is a time-harmonic solution of the phasor form:

$$\Psi(x, y, t) = \Psi(x, y)e^{j\omega t} \quad (67)$$

It then follows from the wave equation (66) that:

$$\frac{\partial^2 \Psi}{\partial x^2} + \frac{\partial^2 \Psi}{\partial y^2} + k^2 \Psi = 0 \quad (68)$$

which is the Helmoltz equation. In this equation *k* is the magnitude of wave vector, which is given by $k = \frac{\omega}{c}$ and Ψ is a wave function, where $\Psi(x, y)$ is complex-valued.

3.2.6 Periodic Wire Structures

According to Pozar, an infinite transmission line or waveguide periodically loaded with reactive elements can be considered as an example of a periodic structure. Despite the existence of several forms of periodic structures, the loading elements can be modeled as reactances grouped in series in a transmission line, which favors the use of microwave network analysis techniques to analyze the phenomena of propagation in these structures [42]. The author also mentions that periodic structures presents passband and stopband characteristics as in filters, being so common to find them in several applications such as traveling wave tubes, masers, phase shifters and antennas [42].

In this dissertation, a type of periodic wire structures is discussed. In Figure 22, it is possible to observe the diagram of the structure of the array of wires, in which the wires are equally spaced of *d*.

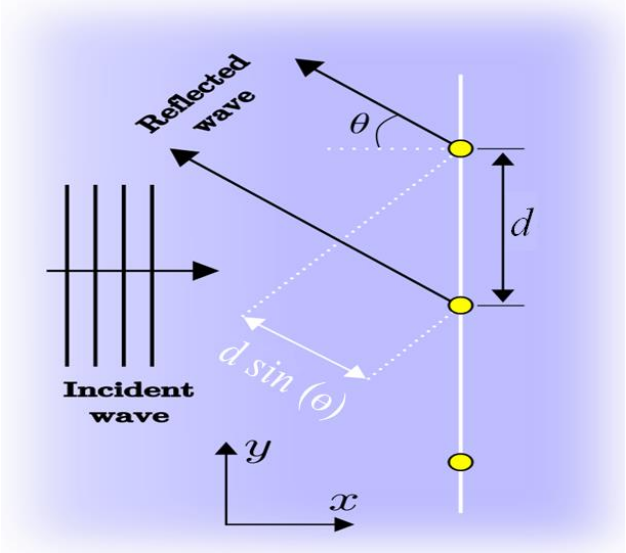

Figure 22 – Diagram of the wire structure.

Let us assume that the solution mentioned in the previous sub-chapter is separable and can be written as

$$\Psi(x,y) = X(x)\,Y(y) \tag{69}$$

then, it follows that

$$\frac{1}{X}\frac{\partial^2 X}{\partial x^2} + \frac{1}{Y}\frac{\partial^2 Y}{\partial y^2} + k^2 = 0. \tag{70}$$

Since there is only propagation in the *xy* plane, *k* can be defined as

$$k^2 = {k_x}^2 + {k_y}^2 \tag{71}$$

in such a way that

$$\left(\frac{1}{X}\frac{\partial^2 X}{\partial x^2} + {k_x}^2\right) + \left(\frac{1}{Y}\frac{\partial^2 Y}{\partial y^2} + {k_y}^2\right) = 0, \tag{72}$$

because the variables are independent. Therefore

$$\frac{1}{X}\frac{\partial^2 X}{\partial x^2} + {k_x}^2 X = 0, \tag{73}$$

and

$$\frac{1}{Y}\frac{\partial^2 Y}{\partial y^2} + {k_y}^2\, Y = 0\ . \tag{74}$$

The solutions for these equations can be written as

$$X(x) = n_1 \mathrm{e}^{\mathrm{j}k_x x} + n_2 \mathrm{e}^{-\mathrm{j}k_x x}, \tag{75}$$

and

$$Y(y) = m_1 \mathrm{e}^{\mathrm{j}k_y y} + m_2 \mathrm{e}^{-\mathrm{j}k_y y}. \tag{76}$$

Since $Y(y)$ is equal to $Y(-y)$, for geometric reasons, it follows that

$$m_1 = m_2, \tag{77}$$

which implies that,

$$Y(y) = c_1 \cos(k_y y). \tag{78}$$

Now, given the periodicity of the structure we have $Y(y+d) = Y(y)$, and it follows that

$$c_1 \cos(k_y y + k_y d) = c_1 \cos(k_y y), \tag{79}$$

and noticing that,

$$\cos(k_y y + k_y d) = \cos(k_y y)\cos(k_y d) - \sin(k_y y)\sin(k_y d). \quad (80)$$

It follows that the above equality is only possible, if:

$$\cos(k_y d) = 1, \sin(k_y d) = 0, \quad (81)$$

which implies that,

$$k_y = \frac{2n\pi}{d}, \quad (82)$$

This restricts the possible solutions for k_y in the Fourier plane k, in the y direction. It is also possible to write:

$$Y(y) = c_1 \cos\left(\frac{2n\pi y}{d}\right). \quad (83)$$

k_x can be represented by the relation between k and k_y, as follows

$$k^2 = {k_x}^2 + {k_y}^2 \leftrightarrow {k_x}^2 = k^2 - \left(\frac{2n\pi}{d}\right)^2. \quad (84)$$

Thus, in a wire we can control the number of modes that propagate through the array, controlling the distance between wires. The constraint that only allows for evanescent wave-type solutions is when the distance between wires (d) is smaller than λ/2. We shall consider this constraint from now on.

In fact, considerig that kind of wave-type solutions, k_y can be expressed assuming that

$$k = \frac{\omega}{c} = \frac{2\pi f}{c} = \frac{2\pi}{\lambda}, \quad (85)$$

as,

$$k_y = \frac{2n\pi}{d} > 2\frac{2n\pi}{\lambda} = \frac{4n\pi}{\lambda} > k, \quad (86)$$

with $n \geq 1$.

For the case when we have $d = \frac{\lambda}{2}$, we may write,

$$k_x = \pm \mathrm{j}\,\alpha_x, \quad (87)$$

from which it follows that:

$$-{\alpha_x}^2 = k^2 - \left(\frac{2n\pi}{d}\right)^2 \leftrightarrow {\alpha_x}^2 = \frac{4n^2\pi^2}{d^2} - k^2. \quad (88)$$

Thus, due to the attenuation, it is possible to admit that all interactions occur around the wires. as described in Figure 23.

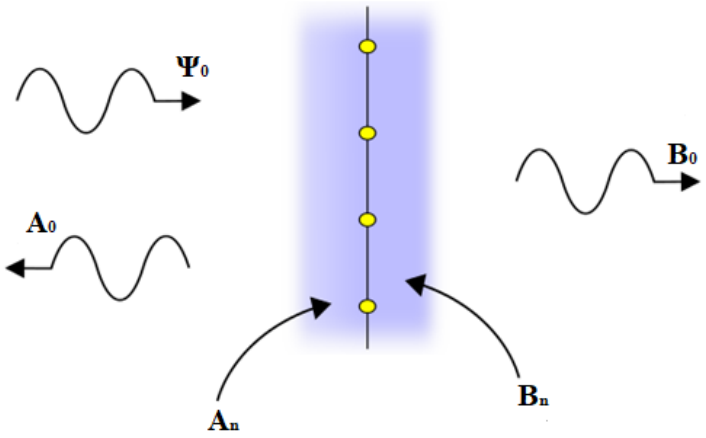

Figure 23 – Representation of the interactions between waves and wires.

These interactions can be described by the following expressions:

$$\Psi(x<0)=\left[\Psi_0 e^{-jkx}+A_0 e^{jkx}+\sum_{n=1}^{\infty}A_n e^{\alpha_x x}\cos\left(\frac{2n\pi y}{d}\right)\right]e^{j\omega t}, \quad (89)$$

and,

$$\Psi(x>0)=\left[B_0 e^{-jkx}+\sum_{n=1}^{\infty}B_n e^{\alpha_x x}\cos\left(\frac{2n\pi y}{d}\right)\right]e^{j\omega t}. \quad (90)$$

α_x can be written more simply if we consider $d \ll \lambda$, which is a reasonable approximation for $d=\frac{\lambda}{2}$:

$$\alpha_x{}^2=\frac{4n^2\pi^2}{d^2}-k^2=\frac{4n^2\pi^2}{d^2}-\frac{4\pi^2}{\lambda^2}, \quad (91)$$

and therefore,

$$\alpha_x \approx \frac{2n\pi}{d} \quad (92)$$

since *k* is negligible under these conditions.

Then, close to the wires we have $|x| \ll \lambda$ which, noting that

$$e^x \approx 1+x \quad (93)$$

results in

$$\Psi(x<0)e^{-j\omega t}=\left[\Psi_0(1-jkx)+A_0(1+jkx)+\sum_{n=1}^{\infty}A_n e^{\frac{2n\pi x}{d}}\cos\left(\frac{2n\pi y}{d}\right)\right], \quad (94)$$

and

$$\Psi(x>0)e^{-j\omega t}=\left[B_0(1-jkx)+\sum_{n=1}^{\infty}B_n e^{-\frac{2n\pi x}{d}}\cos\left(\frac{2n\pi y}{d}\right)\right]. \quad (95)$$

We can impose the continuity of Ψ at $x = 0$,

$$\Psi(x = 0_-) = \Psi(x = 0_+). \quad (96)$$

Using the above equations, we obtain:

$$\Psi_0 + A_0 + \sum_{n=1}^{\infty} A_n \cos\left(\frac{2n\pi y}{d}\right) = B_0 + \sum_{n=1}^{\infty} B_n \cos\left(\frac{2n\pi y}{d}\right), \quad (97)$$

from which it follows that:

$$B_0 = \Psi_0 + A_0, \quad A_n = B_n. \quad (98)$$

With this study, we realized that the wave attenuates as we move away from the structure (according to the direction of the *X* axis).

3.2.7 Circular Wires

It is necessary to find a function that represents the geometric shape of the wires in space and to describe the interaction between the electromagnetic waves and the wires. Taking these facts into account, the following study is presented.

To study the particular case of circular wires, let us consider the following transformation:

$$w = \ln\left(e^{\pi\frac{z}{d}} - e^{-\pi\frac{z}{d}}\right), \quad (99)$$

where $z = x + jy$, $w = u + jv$. So w is now defined as

$$w = u + jv = \ln\left[e^{\pi\frac{z}{d}}\left(1 - e^{-2\pi\frac{z}{d}}\right)\right] = \frac{\pi z}{d} + \ln\left(1 - e^{-2\pi\frac{z}{d}}\right), \quad (100)$$

and if we note that for $|x| \leq 1$:

$$\log(1 - x) = -\sum_{n=1}^{\infty} \frac{x^n}{n}, \quad (101)$$

then, w can also be expressed as:

$$w = \frac{\pi z}{d} - \sum_{n=1}^{\infty} \frac{e^{-\frac{2\pi n z}{d}}}{n} \quad (102)$$

provided that,

$$\left|e^{-\frac{2\pi n z}{d}}\right| \leq 1, \quad (103)$$

which implies that $x \geq 0$, so that u will be equal to the real part of w in these conditions:

$$u = Re\{w\} = \frac{\pi x}{d} - \sum_{n=1}^{\infty} \frac{e^{-\frac{2n\pi x}{d}}}{n} \cos\left(\frac{2n\pi y}{d}\right). \quad (104)$$

It is also possible to predict the expression for the case in which $x \leq 0$, if we consider that $u(x) = u(-x)$ and in this way, it can be written that:

$$u(x \leq 0) = -\frac{\pi x}{d} - \sum_{n=1}^{+\infty} \frac{e^{\frac{2n\pi x}{d}}}{n} \cos\left(\frac{2n\pi y}{d}\right). \quad (105)$$

Let us now consider the situation when the electric field is parallel to the wire arrays. In this case, we have:

$$\Psi = E_z \quad (106)$$

and we note that we must have $\Psi = E_z = 0$ on the surface of the wires. We have seen from equations (94) and (95) that:

$$E_z(x < 0, y)e^{-j\omega t} = \left[\Psi_0(1 - jkx) + A_0(1 + jkx) + A_1 \sum_{n=1}^{\infty} \frac{e^{\frac{2n\pi x}{d}}}{n} \cos\left(\frac{2n\pi y}{d}\right)\right], \quad (107)$$

and

$$E_z(x > 0, y)e^{-j\omega t} = \left[B_0(1 - jkx) + B_1 \sum_{n=1}^{\infty} \frac{e^{-\frac{2n\pi x}{d}}}{n} \cos\left(\frac{2n\pi y}{d}\right)\right], \quad (108)$$

where we made that $A_n = B_n = \frac{A_1}{n}$ for geometric reasons resulting from the transformation introduced earlier. In this way it is possible to introduce the function *u* in the equations that were previously expressed and for this reason we have:

$$E_z(x < 0, y)e^{-j\omega t} \cong (\Psi_0 + A_0) = jkx(\Psi_0 - A_0) - A_1\left(u + \frac{\pi x}{d}\right), \quad (109)$$

and

$$E_z(x > 0, y)e^{-j\omega t} \cong B_0(1 - jkx) - A_1\left(u - \frac{\pi x}{d}\right). \quad (110)$$

To determine A_1 in terms of A_0 and B_0, we will impose the continuity of $\frac{\partial E_z}{\partial x}$ at $x = 0$ and hence it follows that:

$$-jk(\Psi_0 - A_0) - A_1\frac{\partial u}{\partial x} - A_1\frac{\pi}{d} = -jkB_0 - A_1\frac{\partial u}{\partial x} + A_1\frac{\pi}{d} \leftrightarrow$$

$$\leftrightarrow -jk(\Psi_0 - A_0) - A_1\frac{\pi}{d} = -jk(A_0 + \Psi_0) + A_1\frac{\pi}{d} \leftrightarrow \quad (111)$$

$$\leftrightarrow jkA_0 = A_1\frac{\pi}{d} \leftrightarrow A_1 = \frac{jkd}{\pi}A_0,$$

noting that E_z has to be zero on the surface of the wires and considering as an example $(x, y) = (0, a)$, it follows that:

$$\Psi_0 + A_0 - A_1\, u(0,a) = 0 \leftrightarrow \Psi_0 + A_0 - \frac{jkd}{\pi} A_0 u(0,a) = 0 \leftrightarrow$$
$$\Psi_0 + A_0\left(1 - \frac{jkd}{\pi} u(0,a)\right) = 0 \leftrightarrow \frac{A_0}{\Psi_0} = -\frac{1}{1 - \frac{jkd}{\pi} u(0,a)}. \quad (112)$$

Previously we saw that:

$$u(x,y) = \tfrac{1}{2}\ln\left[2\left(\cosh\left(\tfrac{2\pi x}{d}\right) - \cos\left(\tfrac{2\pi y}{d}\right)\right)\right], \quad (113)$$

and therefore,

$$u(0,a) = \tfrac{1}{2}\ln\left[2\left(1 - \cos\left(\tfrac{2\pi a}{d}\right)\right)\right], \quad (114)$$

and then,

$$\frac{A_0}{\Psi_0} = -\frac{1}{1 - \frac{jkd}{2\pi}\ln\left[2\left(1 - \cos\left(\frac{2\pi a}{d}\right)\right)\right]}. \quad (115)$$

Taking into account all these considerations, the equations of the electric field resulting from the interaction between the electromagnetic waves and the wires, which is illustrated in Figure 24 can be determined.

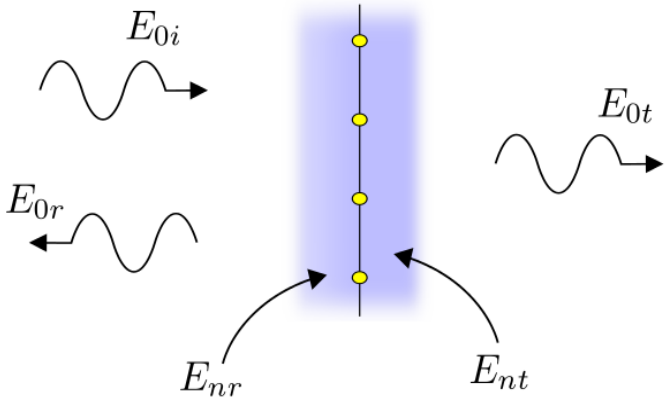

Figure 24 – Representation of the electric field of the waves associated with the interaction with the wires.

After obtaining the expressions of the electric field, it is possible to find an equivalent component to each array of wires, which in this case corresponds to the admittances shown above in Figure 21. Let us focus on the transmission matrix of each admittance present in the structure, which can be expressed as:

$$T = \begin{bmatrix} \mathrm{A} = 1 & \mathrm{B} = 0 \\ \mathrm{C} = \mathrm{Y} & \mathrm{D} = 1 \end{bmatrix}. \quad (116)$$

Applying the formulas of conversion between two-port network parameters, S_{11} will be equal to:

$$S_{11} = \frac{A+\frac{B}{Z_0}-CZ_0-D}{A+\frac{B}{Z_0}+CZ_0+D} = -\frac{YZ_0}{2+YZ_0}, \quad (117)$$

where $Y = G + jB$ and Z_0 is the characteristic impedance of the dielectric slab material. The real part of the admittance Y corresponds to the conductance, while the imaginary part is associated to the susceptance (both quantities are measured in siemens, in SI units). Thus, S_{11} can also be written as:

$$S_{11} = -\frac{1}{1+\frac{2}{YZ_0}}, \quad (118)$$

and therefore, the wires are in fact an admittance, where:

$$\frac{2}{YZ_0} = -\frac{jkd}{2\pi}\ln\left[2\left(1-\cos\left(\frac{2\pi a}{d}\right)\right)\right], \quad (119)$$

where $k = \frac{2\pi}{\lambda}$ and so that, the admittance can be expressed as follows:

$$Y = j\ \frac{\lambda}{d}\ \frac{2{Z_0}^{-1}}{\ln\left[2\left(1-\cos\left(\frac{2\pi a}{d}\right)\right)\right]}. \quad (120)$$

If we consider that

$$\frac{1}{2}\ln\left[2\left(1-\cos\left(\frac{2\pi a}{d}\right)\right)\right] \cong \ln\left(\frac{2\pi a}{d}\right), \quad (121)$$

the expression of admittance shall become:

$$Y = j\frac{\lambda}{d}\ \frac{{Z_0}^{-1}}{\ln\left(\frac{2\pi a}{d}\right)}. \quad (122)$$

With this expression we conclude that the value of the conductance (G) is equal to 0, while the value of the susceptance (B) will be

$$B = \frac{\lambda}{d}\ \frac{{Z_0}^{-1}}{\ln\left(\frac{2\pi a}{d}\right)}. \quad (123)$$

For negative values of B is associated with an inductive behaviour, whereas for positive values of B we will have a capacitive behaviour. No restriction has been imposed on the relationship between a and d, although it is reasonable to assume that $a \ll d$.

A numerical simulation of the dielectric slab with the arrays of wires, as described in Figure 21, can be performed in order to corroborate the aforementioned theory and to understand how the parameters can influentiate the reflectance and the transmittance of the structure. In addition to the parameters considered for the case of the dielectric slab, in the present case a distance between wires $d = l/35$ and a radius $a = d/100$ were also considered as relevant parameters associated with the wires.

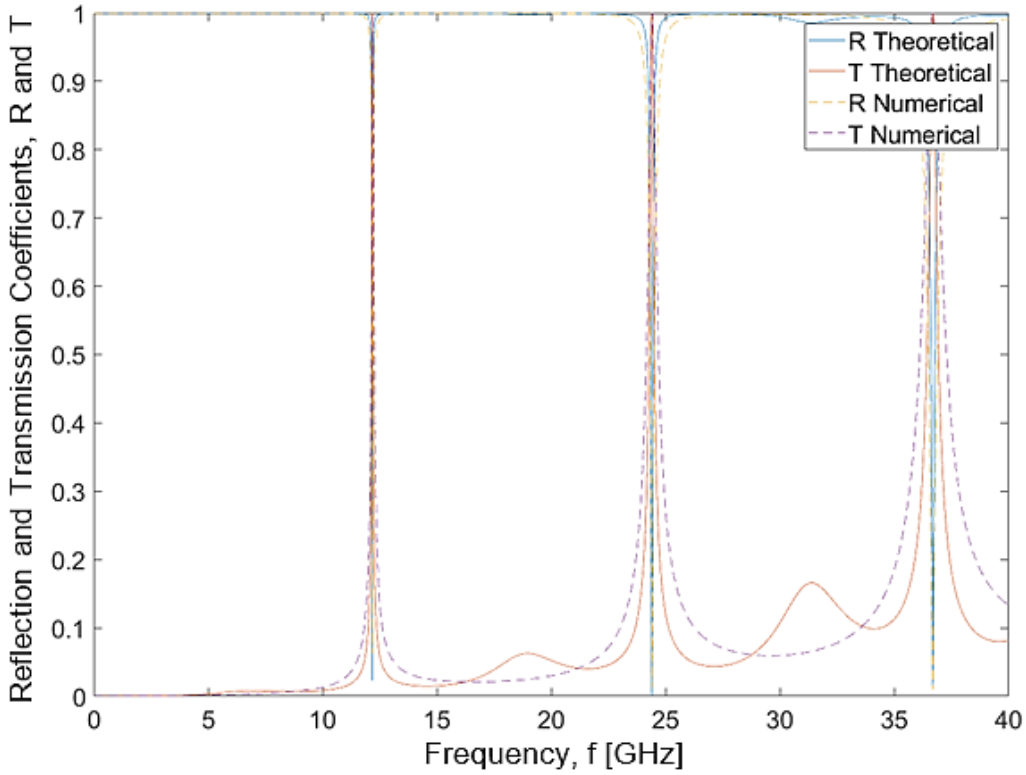

Figure 25 – Evolution of the curves of R and T for propagation in a dielectric slab with the arrays of wires.

Figure 25 shows the variation of reflectance and transmittance coefficients, which were obtained based on the numerical simulation considering a frequency sweep that starts at 0.005 GHz and finishes at 40 GHz, with a step size of 0.005 GHz. Through the simulation, it was discovered that admittance exhibits an inductive behaviour (Y is always negative along the frequency sweep) and causes the cancellation of the odd harmonics, leaving only the even harmonics when compared to Figure 20.

It has also been found that by decreasing the value of the distance between wires, the bandwidth of the curves decreases in such a way that a greater frequency selectivity is observed at the even harmonics. If we consider *d=l/50*, the variation of the reflectance and transmittance coefficients will be the one observed in Figure 26.

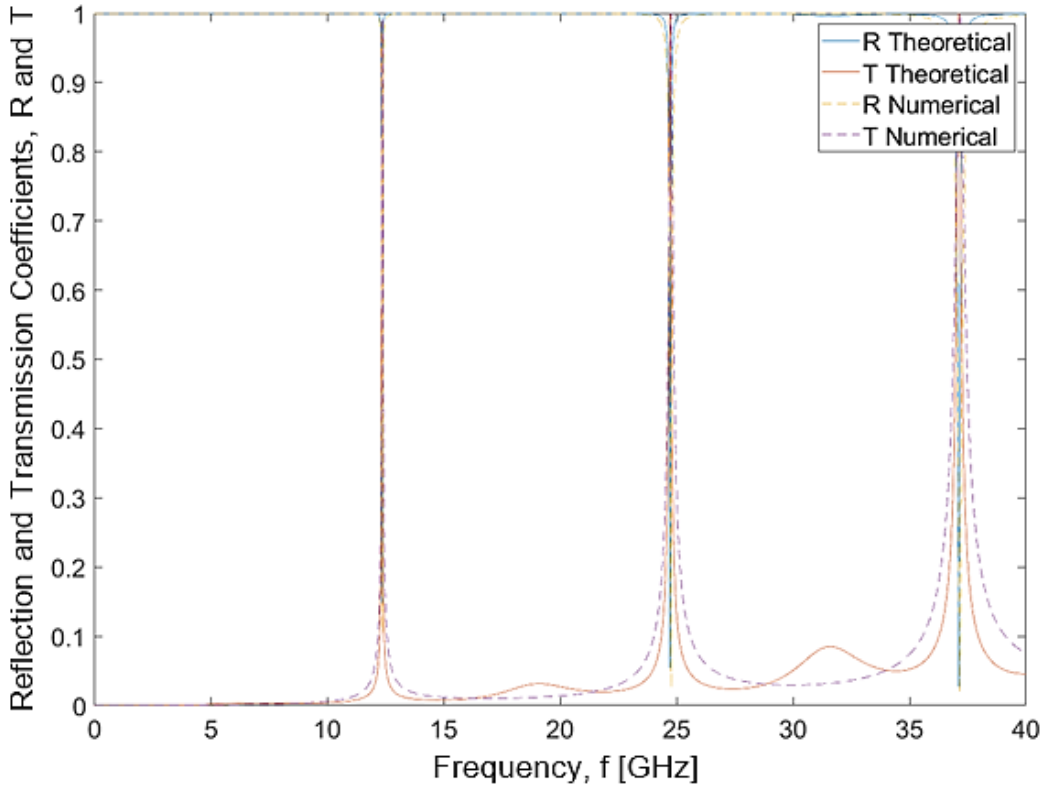

Figure 26 – Evolution of the curves of R and T for propagation in a dielectric slab with the arrays of wires, with d=l/50.

By decreasing the radius of the wires having a *d=l/35*, the bandwidth of the curves increases in such a way that a lower frequency selectivity is observed at the even harmonics. If we consider *a=d/200*, the variation of the reflectance and transmittance coefficients will be the one observed in Figure 27.

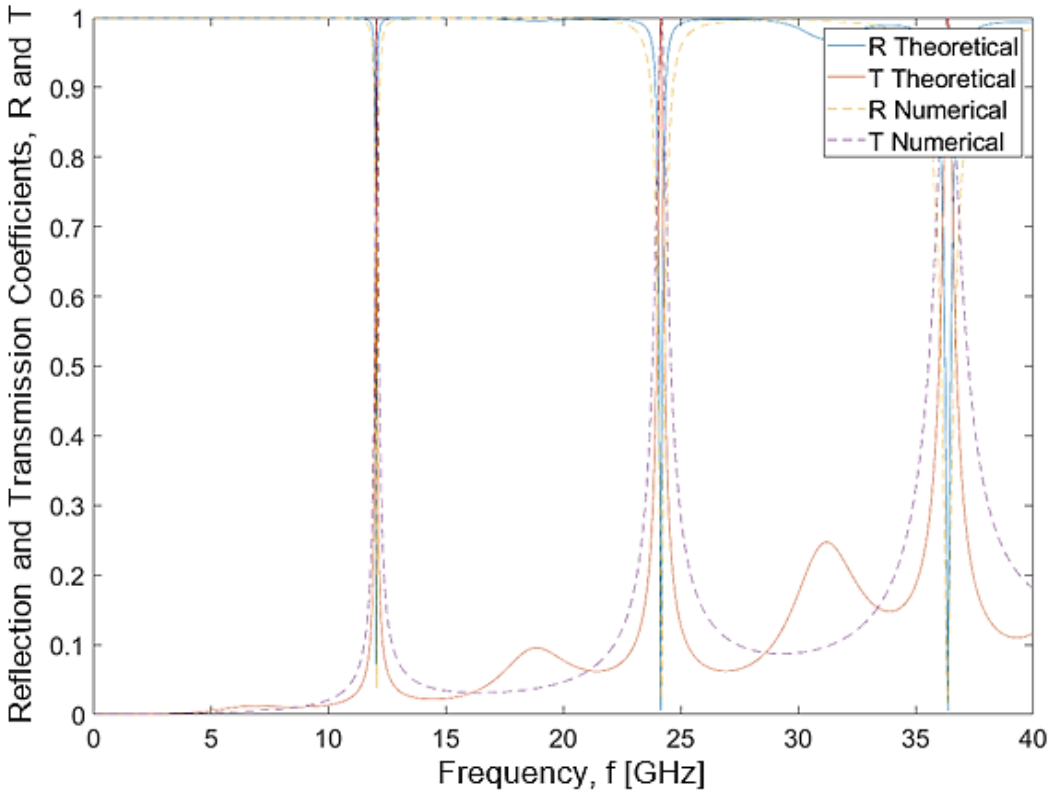

Figure 27 – Evolution of the curves of R and T for propagation in a dielectric slab with the arrays of wires, with a=d/200.

When a comparison is established between Figure 26 and Figure 27 having as a reference Figure 25, it is understood that these results corroborate the aforementioned theory on the control of admittance behaviour. In addition, the spacing between harmonics is influenced by the length of the structure (*l*), since the larger the parameter l the smaller the spacing between harmonics. The length of the structure of Figure 25 was *l=1cm*, but if the value of *l* doubled, the following behaviour will be observed in Figure 28.

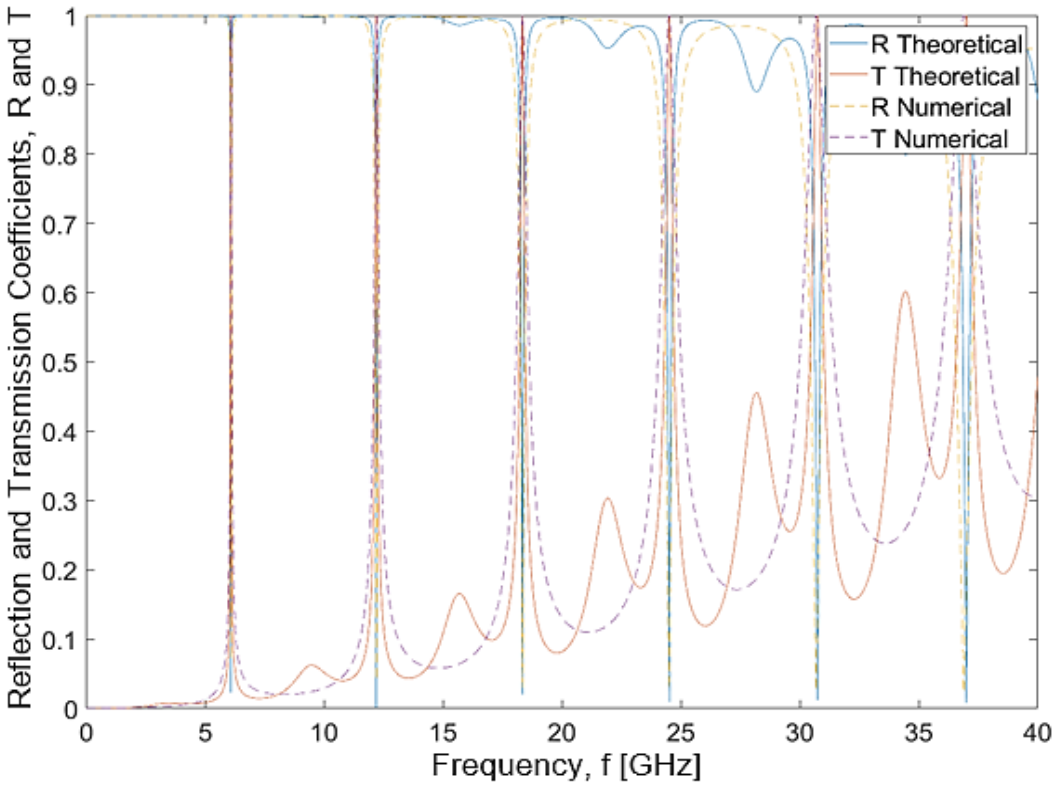

Figure 28 – Evolution of the curves of R and T for propagation in a dielectric slab with the arrays of wires, with l=2cm.

The spacing between harmonics of Figure 28 is about half compared to that in Figure 25. Furthermore, it can also be observed twice the number of harmonics, as compared to the previous figure. As a conclusion, these observations reinforce the theory that to control the admittance, only the parameters *a*, *d* and *l* are influential.

3.3. Design and Simulation

In this sub-chapter will be addressed some topics related to the design, the numerical simulation method and the sensor modeling procedure of the sensor incorporating the proposed filter.

3.3.1 Sensor Design

After studying the operating principle of our THz filter and characterizing the role of the resonant effects in the THz radiation detection mechanism, we will propose a new type of stress sensor based on our filter.

In Figure 29, it is possible to observe the schematic of the proposed sensor. The sensor is composed by two square shaped cells of plastic material with 70 wires each. This

number was chosen due to the limitation of the software to generate mechanical models with more than 70 wires for the design shown below.

The remaining specifications were selected based in order to provide the desired characteristics for the sensor. To accomplish this selection, it was necessary to perform several electromagnetic and mechanical simulations that later led to the following characteristics:

- Distance between wires (d) ∈ [0.015; 0.02] mm, with a step size of 0.0005 mm;
- Length (l) =1.42 mm;
- Radius of the wires (a) = 0.0002 mm;
- Resonance frequency target (F_{target}) defined based on the theoretical model, considering the relative permittivity (ε_r) and permeability (μ_r) of plastic material used in both cells and a frequency range between 395 and 455 GHz;

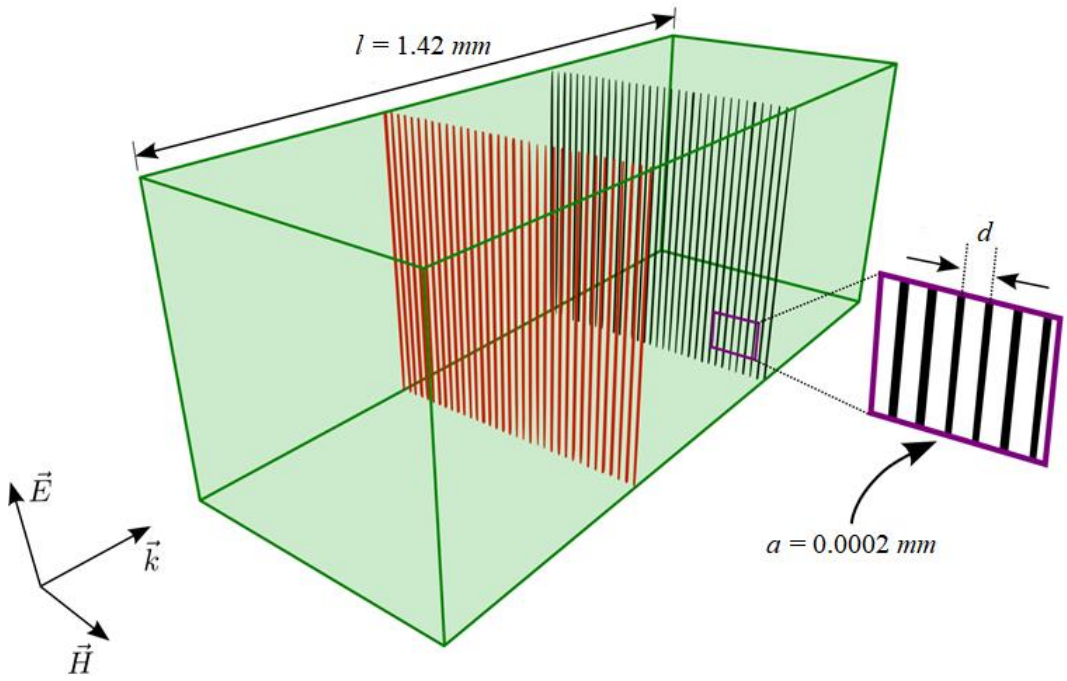

Figure 29 – THz sensor scheme.

The distance between the wires is expressed as a range, since the previous study proved that it is possible to control the behaviour of the filter by varying this parameter. In this way, we consider submitting the sensor at a compression rate of up to 25% and figuring out what will be the sensor response as a result of this action.

To create a sensor with some robustness and a good degree of sensitivity, one should consider materials that will be relatively easy to compress and exhibit low losses and low

dispersion in the region of THz frequency. According to M. Naftaly and R. Dudley from National Physical Laboratory, silicon and plastics such as HDPE and PTFE are key terahetz materials, once they fulfill the requirements mentioned above for the frequency band over 0.1 to 5 THz [45].

Naturally, the plastic materials will be the most suitable solution, since other materials, like silicon, will require a substantially higher level of compression compared to the plastics. However, it is important to consider other physical and mechanical characteristics that can be useful for the design and assembly of the sensor.

HDPE owns a linear structure with few branches lending to its optimal strength/density ratio. In fact, this thermoplastic presents some interesting mechanical characteristics, such as: the sensitivity to stress cracking, the possibility to customize their physical properties through the moulding process that is used in its manufacturing and it can handle temperatures from 173.15 to 353.15 Kelvin degrees (K) [46, 47].

PTFE has also some special features, such as: weatherability and the capability of maintaining high strength, toughness and self-lubrication at low temperatures down to 5 K, and good flexibility at temperatures above 194 K [48, 49].

It should be noted that both HDPE and PTFE can be manufactured using 3D printing. Considering their properties and the fact that both have low losses and low dispersion in the frequency region of the THz, as mentioned above, these materials prove to be in fact good candidates to be used in the assembly of the proposed sensor.

Despite the constraint imposed by the software that generates the mechanical model of the device, the methodology used in the mechanical simulations will allow us to understand what happens to the required force to compress the sensor as the number of wires increases.

In chapter 4, the results will be presented and analyses from an electromagnetic and mechanical point of view will be performed with the aim of showing the sensor behaviour, as compression is applied.

3.3.2 Simulation Method

The finite element method was used to predict and confirm the behaviour of the sensor, according to its working principle. This solving technique is an efficient method that can provide approximated solutions to partial differential equations and it is widely used to solve problems of several areas, such as: electromagnetics, fluid flow, heat transfer, structural analysis, among others [50].

First, the structure under study is designed in a 3-dimensional model and boundary conditions are imposed on each domain of the structure, based on the theoretical study. Then, the finite element method will subdivide the structure into many smaller, simpler elements in a process known as meshing. It is always possible to connect adjacent elements at nodes, as shown in Figure 30 and the number of elements produced can be adjusted to achieve the desired solution resolution, which process is known as refinement [51].

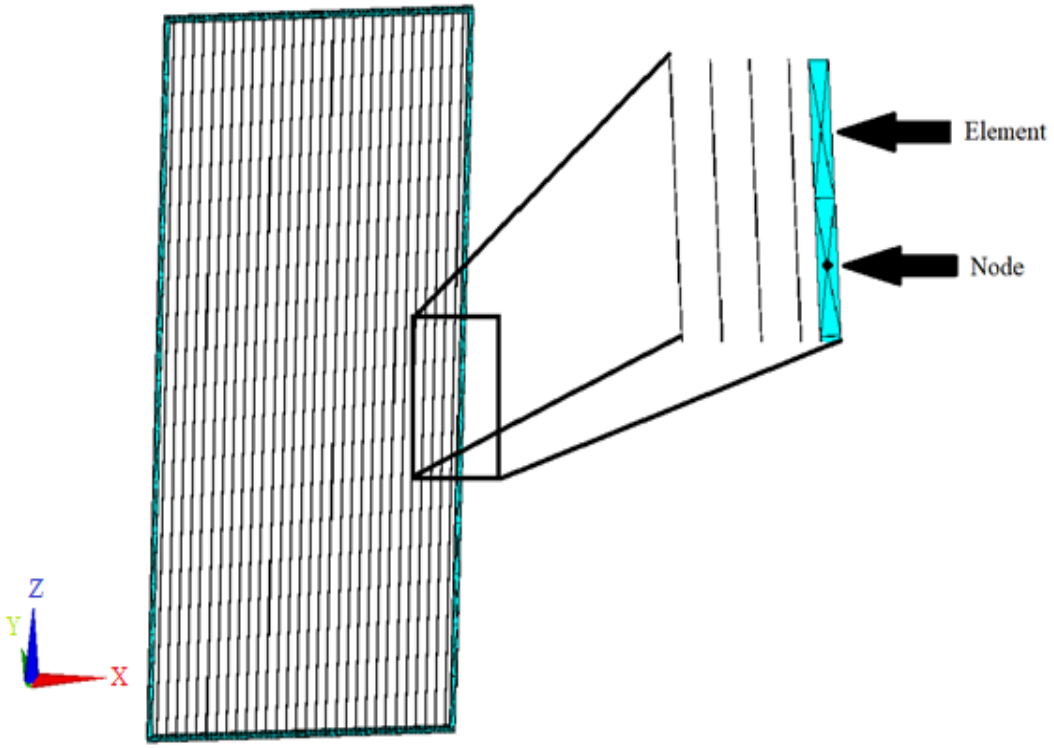

Figure 30 – The finite element method process of meshing of the structure under study.

The desired field expressions to be approximated are interpolated by polynomials over each element and from this interpolation results a large set of algebraic equations to be solved simultaneously [51]. Unknown field quantities over each element can be found

from the approximated solutions and using these solutions, many other field quantities may also be calculated such as energy, power, among others [50].

3.3.3 Sensor Modeling

As mentioned in the previous topic, it is necessary to study and analyze the sensor response, from the electromagnetic and mechanical point of view, when compression is applied.

Since the sensor response analysis is confined to two components, electromagnetically and mechanically, respectively, it will be necessary to model the sensor in this context. To satisfy this requirement, the following softwares were used to model the sensor:

- ANSYS HFSS;
- Gmsh;
- Elmer solver;

Using ANSYS HFSS it is possible to model and perform electromagnetic simulation. In this software the sensor can be represented by a "box" and two "cylinders" placed as suggested in the equivalent circuit of Figure 21. The “box” represents the plastic material and the "cylinders"represents the gold wire arrays. Naturally, there will be more than one wire in each array, however this software assumes this considering the infinite replication in the direction of propagation.

After specifying all parameters associated with the structure to be simulated, the boundary conditions are imposed and excitations are defined. Before executing the simulation, an analysis is performed to verify the existence of problems with the model of the structure under study. The creation of the mesh and its refinement takes place automatically after being indicated by the user to execute the simulation. In Figure 31, it is possible to observe the model that was used in this simulation:

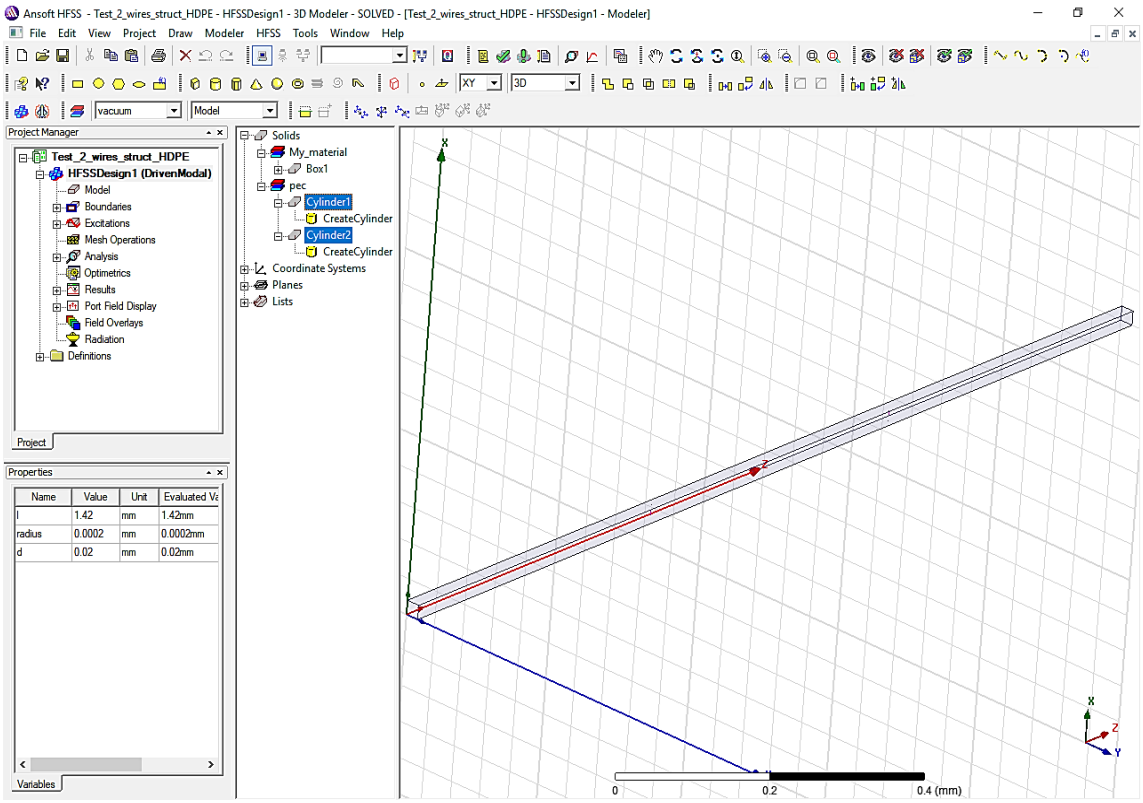

Figure 31 – Model of THz sensor on ANSYS HFSS.

Through the Gmsh it was possible to design the sensor and generate its corresponding mesh. However as drawing through software becomes increasingly complex and difficult as new elements are added, we have chosen to write a program in C++ that makes the drawing in an automated way and easily scalable to the specifications imposed by the user.

The 3D mesh generation process is performed manually by the user in the software and all mesh optimization operations are performed automatically by the software. In Figure 32, it is possible to observe the model that was automatically generated by the program developed in C++ before the process of mesh generation.

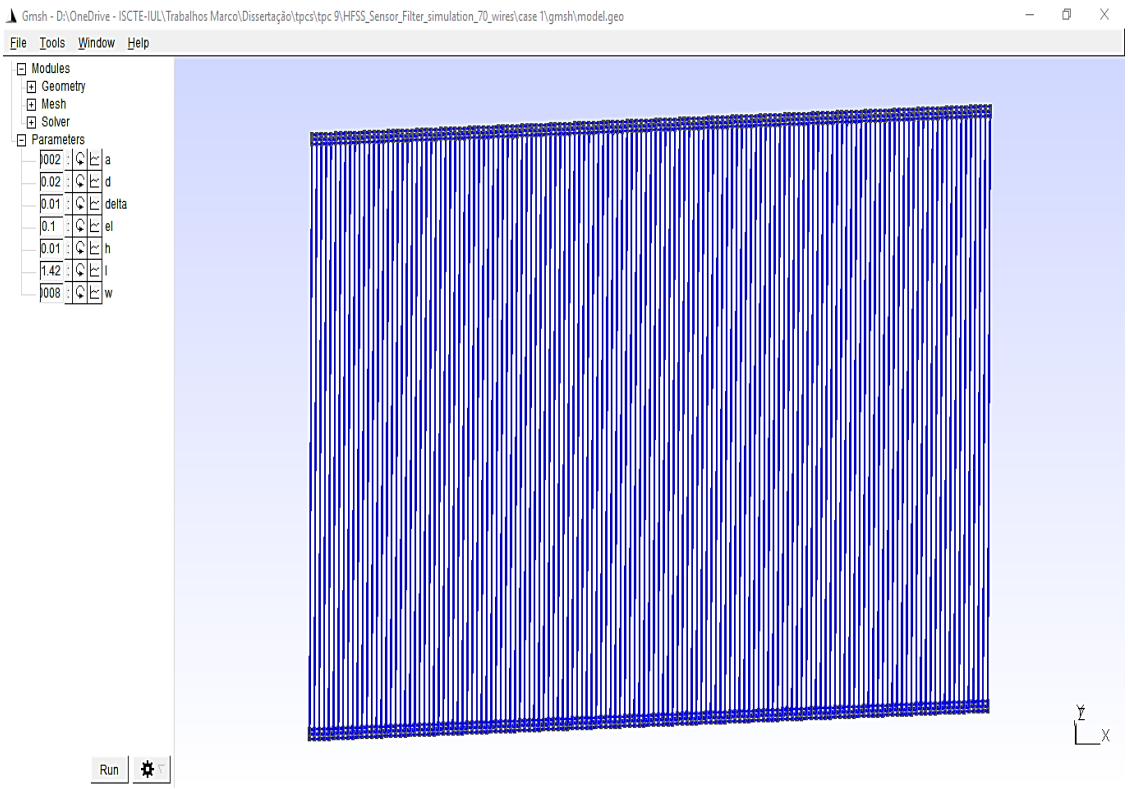

Figure 32 – Model of THz sensor on Gmsh.

After generating the sensor mesh, the same was imported into the Elmer solver, as shown in Figure 33. In Elmer, the materials and boundary conditions associated with the elasticity equation were defined. Knowing the range to be tested for the wire spacing and the lateral section area of the sensor, it is possible to determine the level of compression needed to obtain the required wire spacing.

This is possible to achieve, knowing that the materials have certain physical properties, some of which are especially important for calculating the displacement derived from the compression applied to the sensor, among them: density, young's modulus and poisson's ratio. Elmer has a library with some materials, but if we need to simulate with materials other than those available, it will be necessary to consult some specific datasheets to know the values associated with each parameter, which allows us to define the material in study.

Acording to Walter Lewin, it is possible to establish a relation between the displacement and the force applied to a material, by the following equation [52]:

$$\frac{F}{A} = E \times \frac{\Delta l}{l} \quad (124)$$

where *F/A* is the force applied per unit area (stress), *E* is the young's modulus and $\Delta l/l$ is the extension per unit length (strain).

The stress is expressed in Newton per square meter (Nm^{-2}) or in Pascal (Pa) and the young's modulus is usually expressed in GPa. Last but not least, the strain has no units, because it is just a ratio between the extension and the original length of the object.

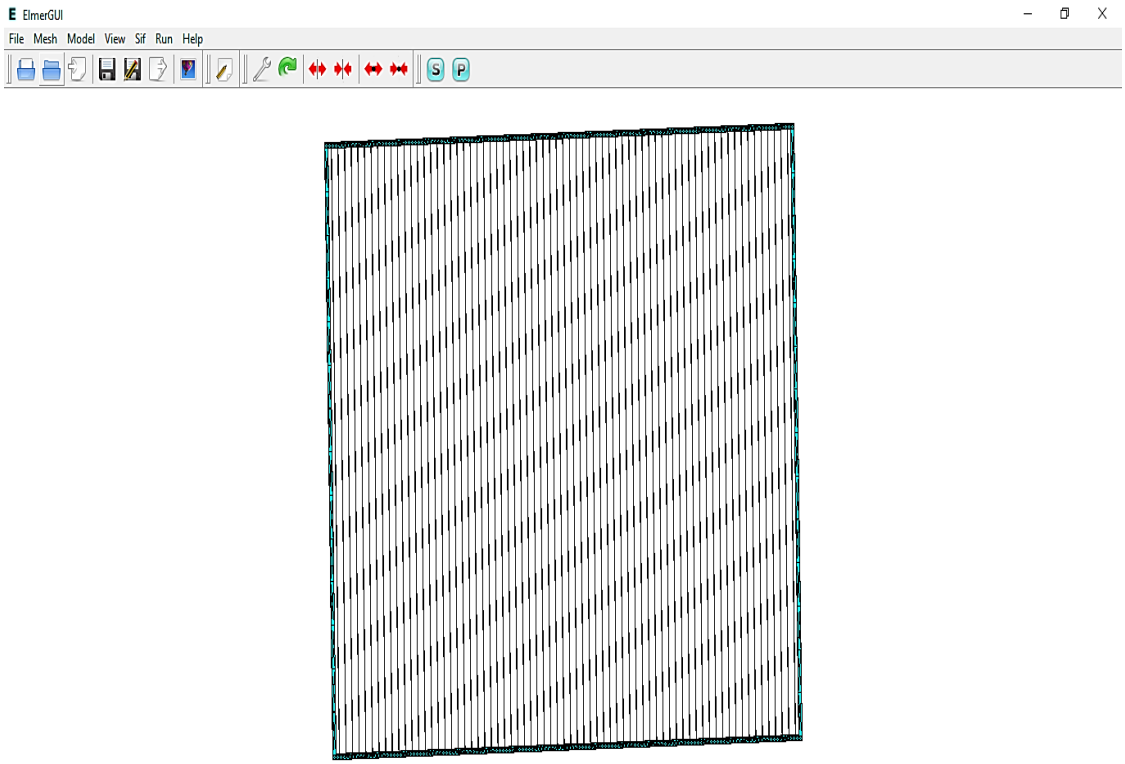

Figure 33 – Model of THz sensor on Elmer.

Finally, having determined the compression levels to be applied, it became necessary to find out which current would be required to provide a force, previously discovered through Elmer, to compress the sensor.

Finite Element Method Magnetics (FEMM) is a software used to solve 2D planar and axisymmetric problems in low frequency magnetics and electrostatics. This powerful tool is composed of an interactive shell that includes pre and post-processing graphics, a mesh generator, and several solvers [53].

A plunger was designed in the FEMM software with the specific dimensions to perform the compression of the lateral section of the sensor and, for each case of compression, the required current can be determined. In Figure 34, the model of the plunger to be used for applying compression to the sensor is shown. However, since this

magnetostatic problem is of the axisymmetric type, it allows us to simplify the design of the model to be simulated. In fact, instead of drawing the object in its entirety, it is enough to draw only half of the same considering the existence of an axis of symmetry. According to the software documentation, FEMM assumes by convention that the $r = 0$ axis is understood to run vertically, and the problem domain is restricted to the region where $r \geq 0$. In addition, by considering this convention the software developers mention that the positive-valued currents will flow in the into-the-page direction of the model [53].

After drawing the plunger and the coil, the materials used for each of the components of the model are specified and the boundaries are drawn for the solution region. The square of Figure 34 represents the boundary condition that was used in the simulation. Setting a boundary condition is not required, since the program runs anyway. However, it is strongly suggested for better results according to the developers [53].

The plunger is composed of iron and the coil consists of 1000 turns of copper wire. With all necessary conditions being met, the solver will mesh and find a solution over a finite region of space that contains the objects of interest.

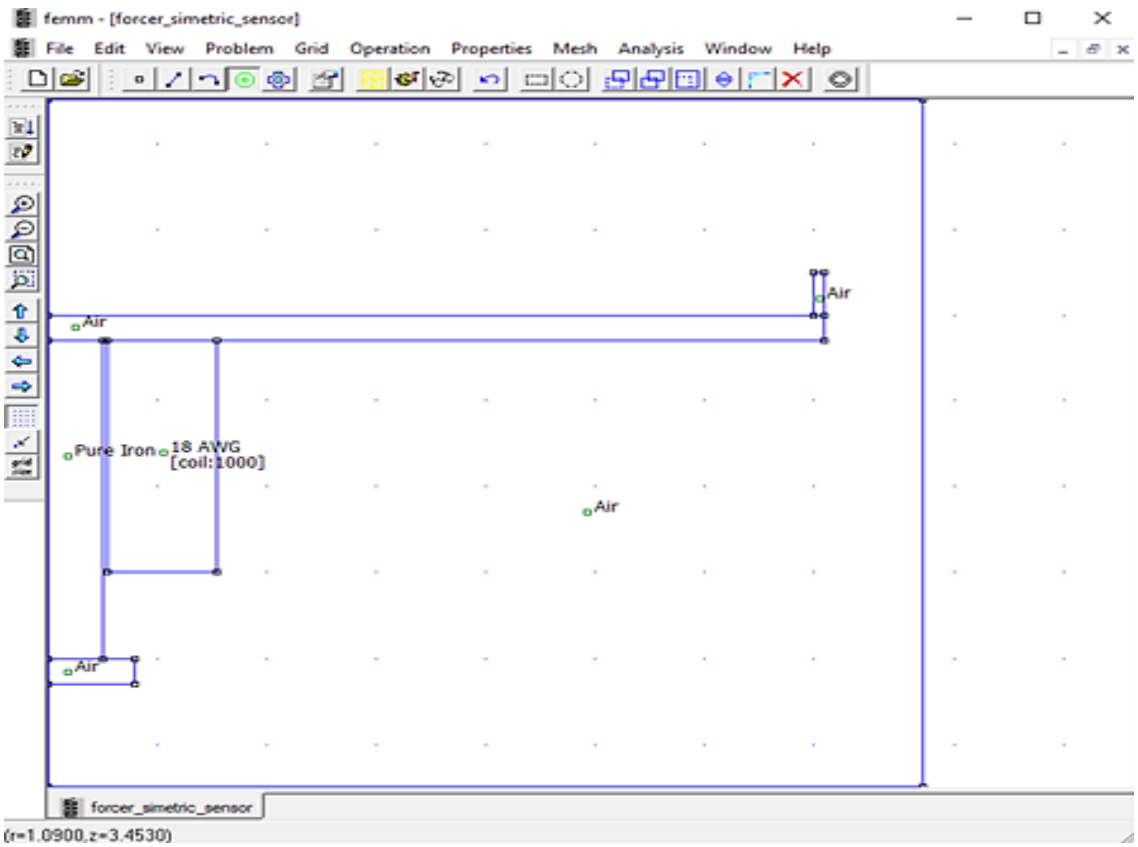

Figure 34 – Model of plunger on FEMM.

Chapter 4 – Analysis and discussion of results

In this chapter, the results of the design of the sensors will be presented for each assembling hypothesis. Two analyses of its behaviour from the electromagnetic and mechanical point of view will be carried out. Two methodologies will also be discussed in this chapter, which will allow to apply the desired compression level to the sensors and thus control the distance between wires.

4.1. Electromagnetic Analysis

After the conception and presentation of the theoretical study of the operating principle of the filter, it is necessary to demonstrate the accuracy of the theoretical model in relation to the simulation case and analyse some relevant issues such as: the dynamic range, the periodicity of the shifts of resonance frequency and the evolution of the bandwidth of the sensor, under compression.

Knowing that HDPE and PTFE have different relative permittivity and permeability constants, it will be necessary to analyze the sensor design for both assembling cases. It should be noted that each simulation has an average duration of 30 minutes and that 11 simulations were performed for each of the cases under analysis.

4.1.1. HDPE

The accuracy of the theoretical model in relation to the simulation case can be demonstrated first by proceeding to the simulation of the normal case, where there is no compression of the sensor at the predefined operating frequency, as shown in Figure Figure 35:

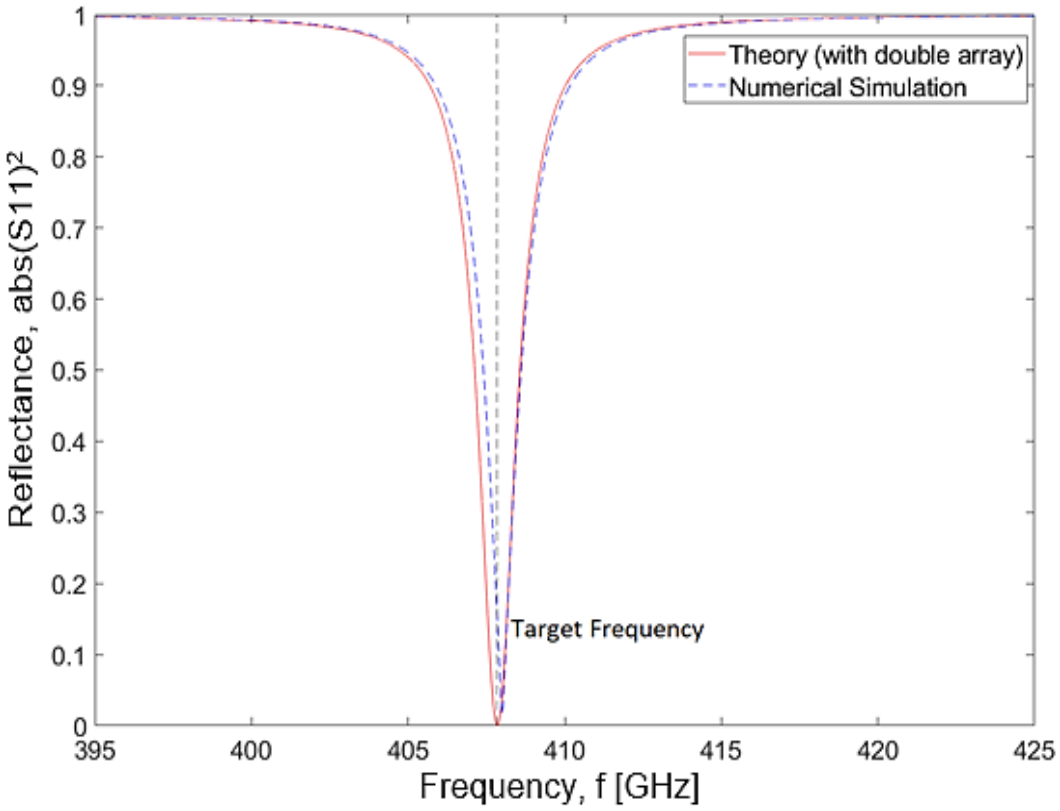

Figure 35 – Curves of the relation between frequency and reflectance, based the theoretical study and numerical simulation.

A relative permittivity (ε_r) of 2.3 and a relative permeability(μ_r) of 1.0 were considered and as can be seen in Figure 35, the simulation corroborates the theoretical model and it is what is expected as the distance between wires decreases by the effect of compression. A frequency shift is observed for the resonance frequency target (407.8 GHz).

After the electromagnetic simulations were carried out for each case of compression on ANSYS HFSS, the graphs presented in Figure 36 and Figure 39 were elaborated to demonstrate the variation of the sensor response in terms of reflectance and transmittance, as compression is applied.

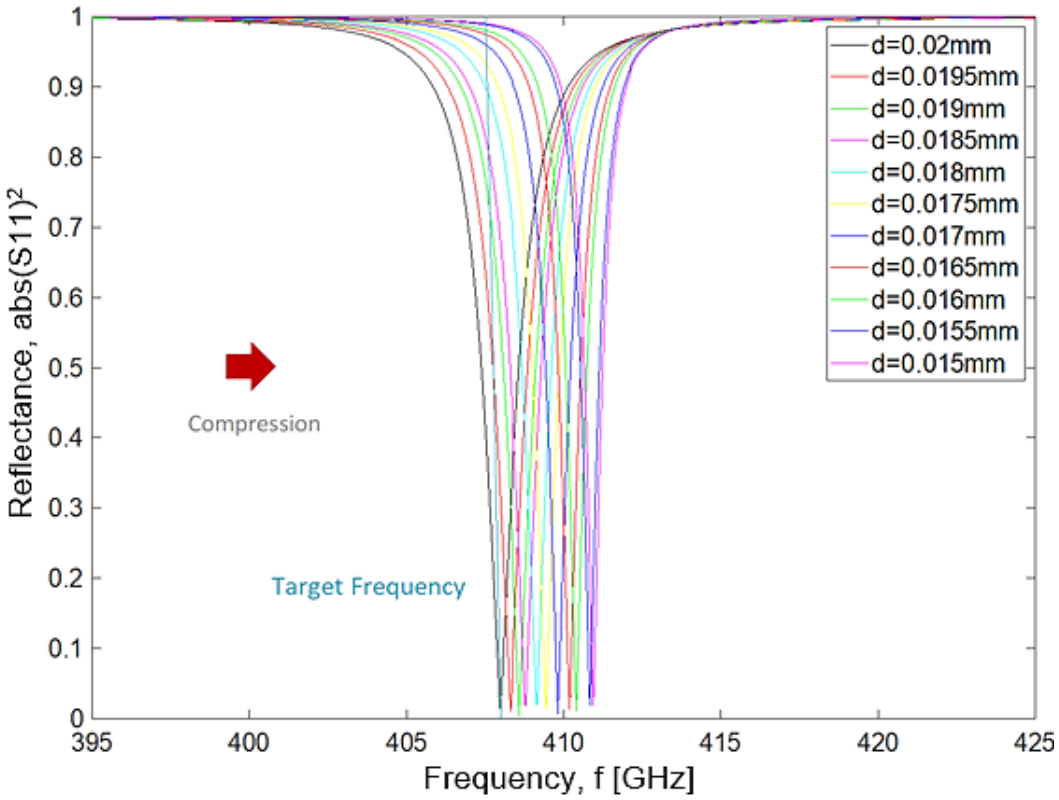

Figure 36 – Curves of the relation between frequency and reflectance, for each case of compression.

By analyzing the data, which originated Figure 36, it was verified that an average displacement in the frequency of 278MHz occurs at each reduction of 0.0005mm of the distance between the wires. This shift of the resonance frequency is approximately constant as compression is applied and consequently the distance between wires is reduced, as shown in Figure 37. Between the normal state in the absence of compression and the state in which there is a compression of 25%, there was a shift in the frequency of 2.9 GHz compared to the initial frequency.

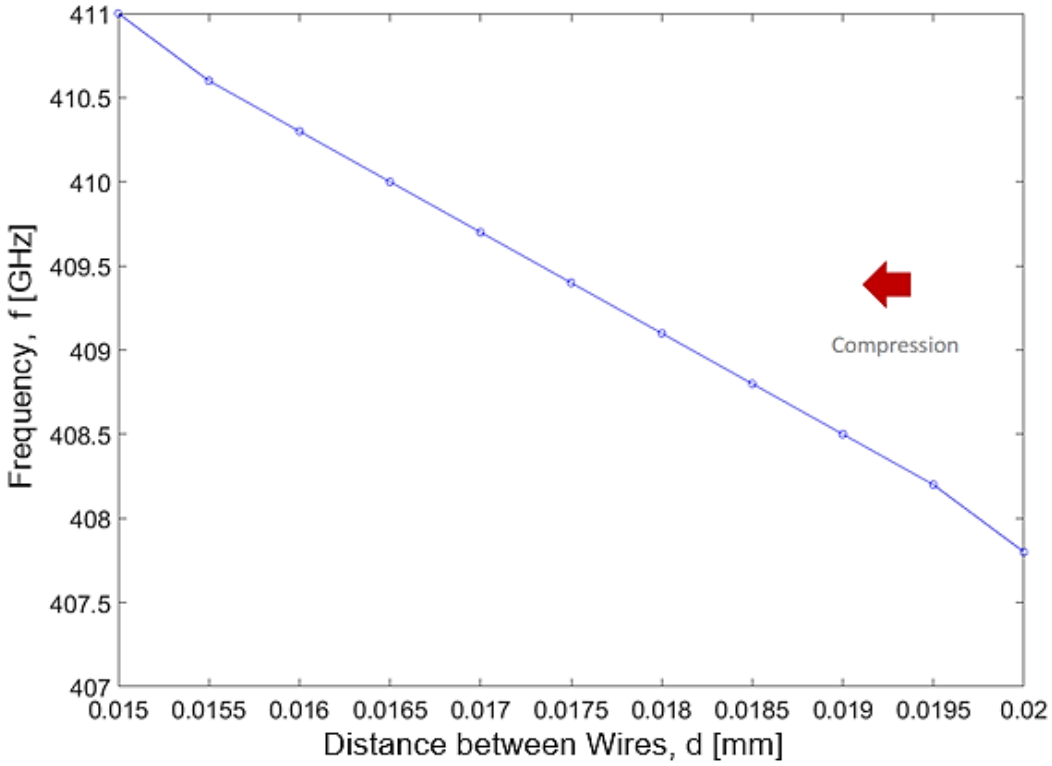

Figure 37 – Curve of the relation between the distance between wires and the resonance frequency, for each case of compression.

Also through Figure 36 the good dynamic range of the sensor is notorious, however, as the resonance frequency changes, because of the reduction of the distance between wires, some saturation is observed when the maximum compression state is approached, as shown in Figure 38.

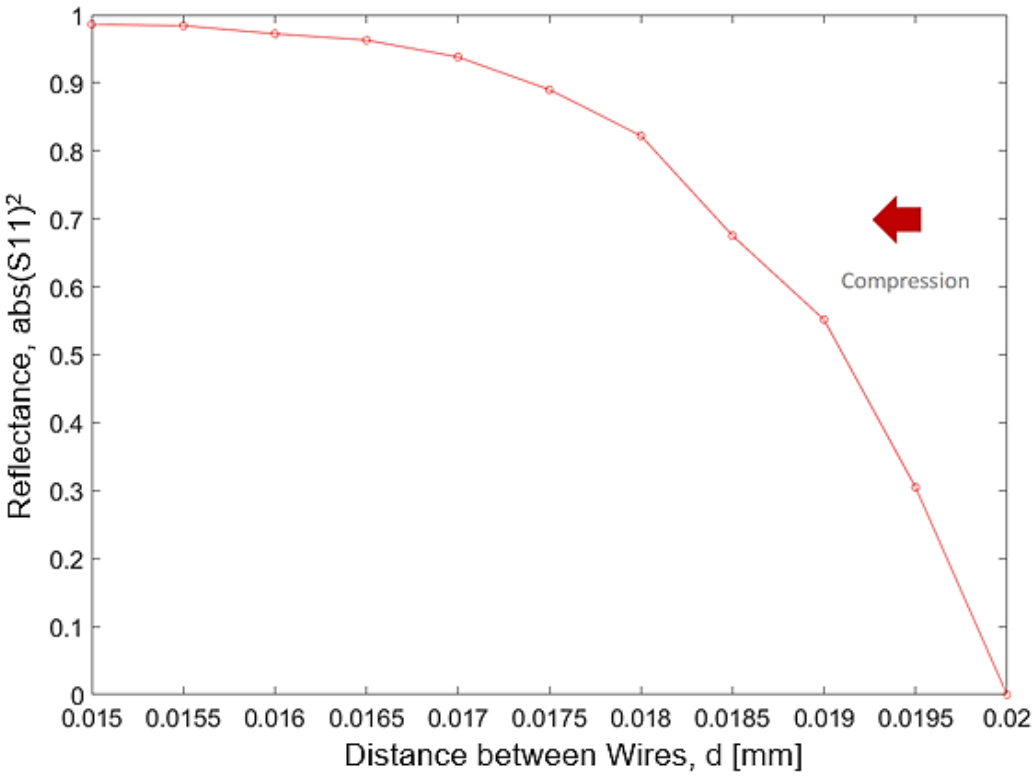

Figure 38 – Curve of the relation between the distance between wires and the reflectance, for each case of compression.

Analyzing Figure 36 and Figure 39, it is also possible to conclude that, as the compression is applied to the sensor and the frequency shift is verified, that the bandwidth decreases and the selectivity increases. In this way, it can be assumed that the higher the compression level applied, the greater the selectivity. Between the normal state in the absence of compression and the state in which there is a compression of 25%, there was a reduction of the bandwidth at 3dB from 2.1 GHz to 1 GHz, which represents a decrease of more than 50% compared to the initial case. The average bandwidth reduction value at 3dB is approximately 111MHz, as compression is applied to the sensor.

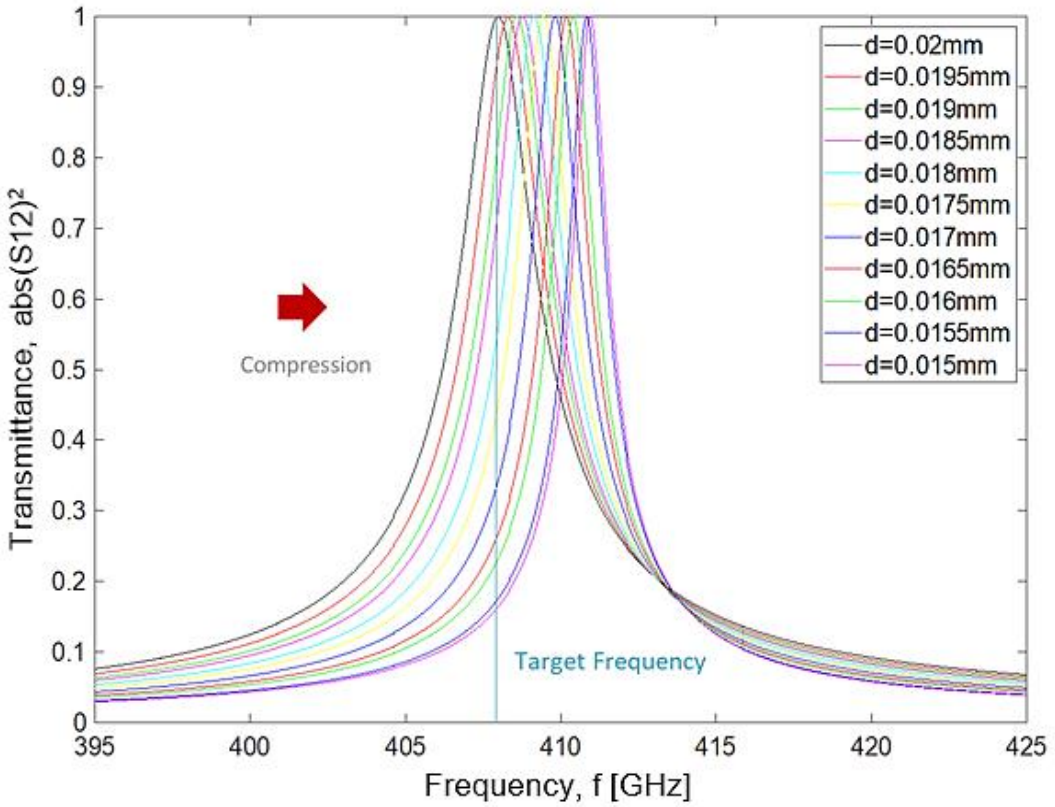

Figure 39 – Curves of the relation between frequency and transmittance, for each case of compression.

4.1.2. PTFE

Similarly, to the previous case, the accuracy of the theoretical model in relation to the simulation case can be demonstrated using the same methodology. After performing the simulation of the normal case, where there is no compression of the sensor at the predefined operating frequency, the following figure was obtained:

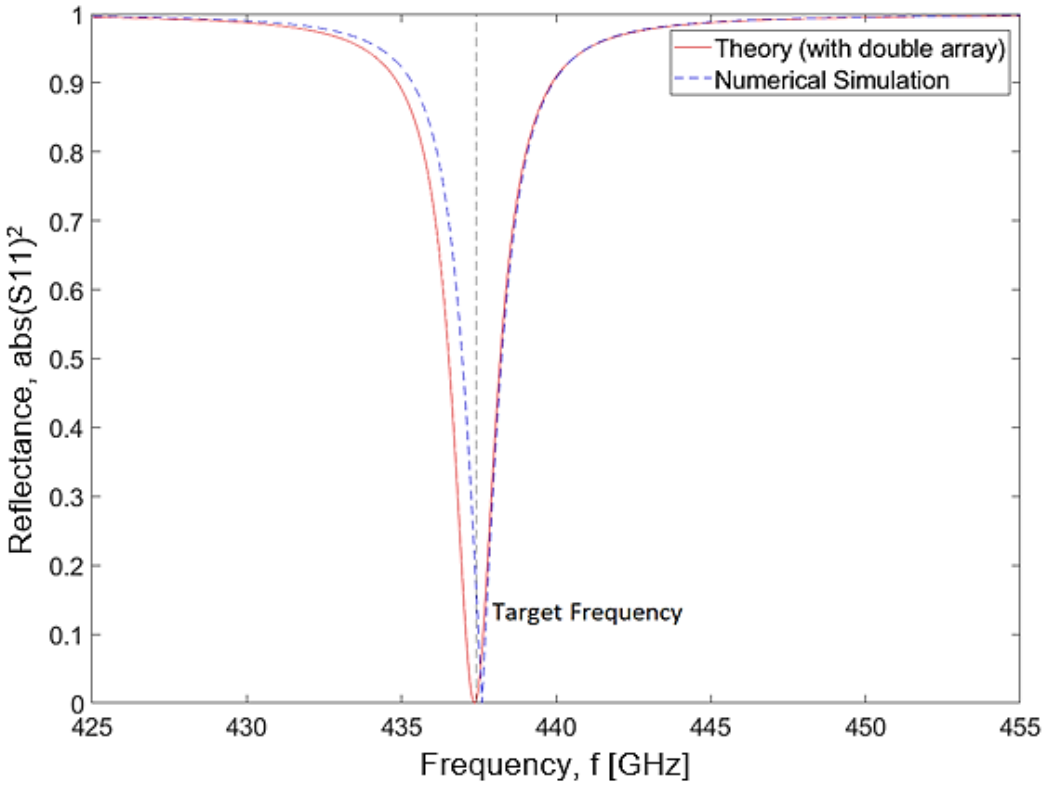

Figure 40 – Curves of the relation between frequency and reflectance, based the theoretical study and numerical simulation.

A relative permittivity (ε_r) of 2.0 and a relative permeability(μ_r) of 1.0 were considered and as can be seen in Figure 40, the simulation corroborates the theoretical model and it is what is expected as the distance between wires decreases by the effect of compression. A frequency shift is observed for the resonance frequency target (437.4 GHz).

After the electromagnetic simulations were carried out for each case of compression on ANSYS HFSS, the graphs presented in Figure 41 and Figure 44 were elaborated to demonstrate the variation of the sensor response in terms of reflectance and transmittance, as compression is applied.

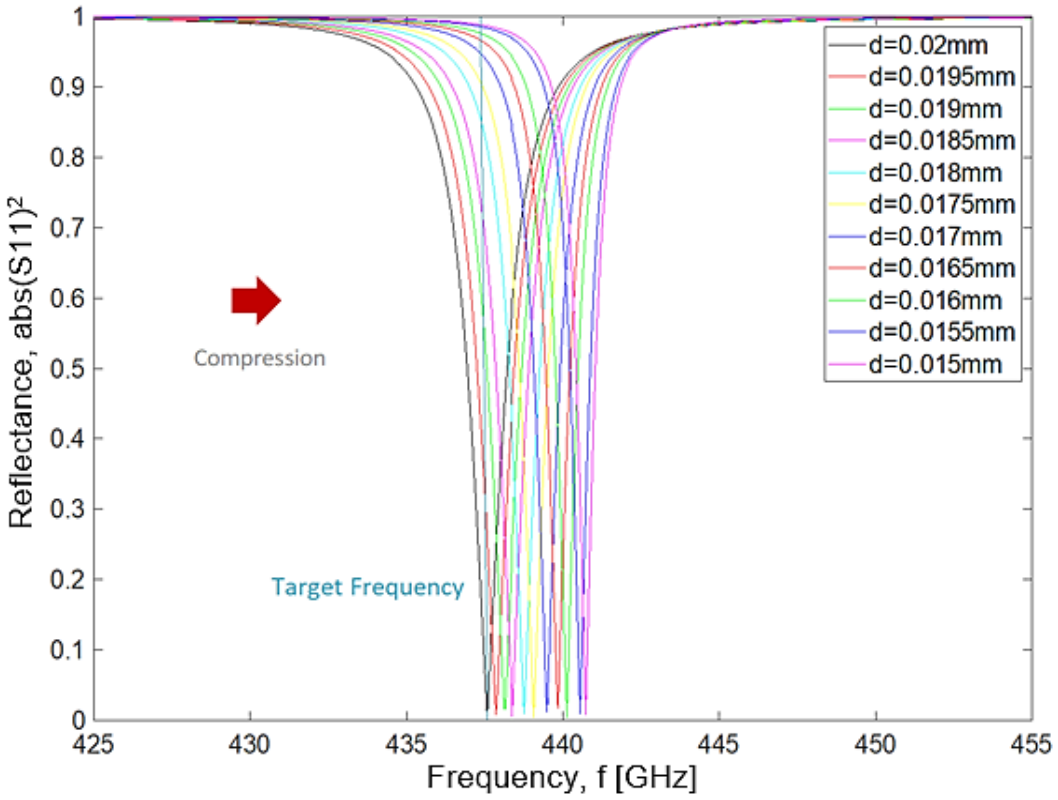

Figure 41 – Curves of the relation between frequency and reflectance, for each case of compression.

By analyzing the data, which originated Figure 41, it was verified that an average displacement in the frequency of 300MHz occurs at each reduction of 0.0005mm of the distance between the wires. This shift of the resonance frequency is approximately constant as compression is applied and consequently the distance between wires is reduced, as shown in Figure 42.However, when compared with Figure 38, it appears that this appears to be a less linear frequency shift.

Between the normal state in the absence of compression and the state in which there is a compression of 25%, there was a shift in the frequency of 3.1 GHz compared to the initial resonance frequency.

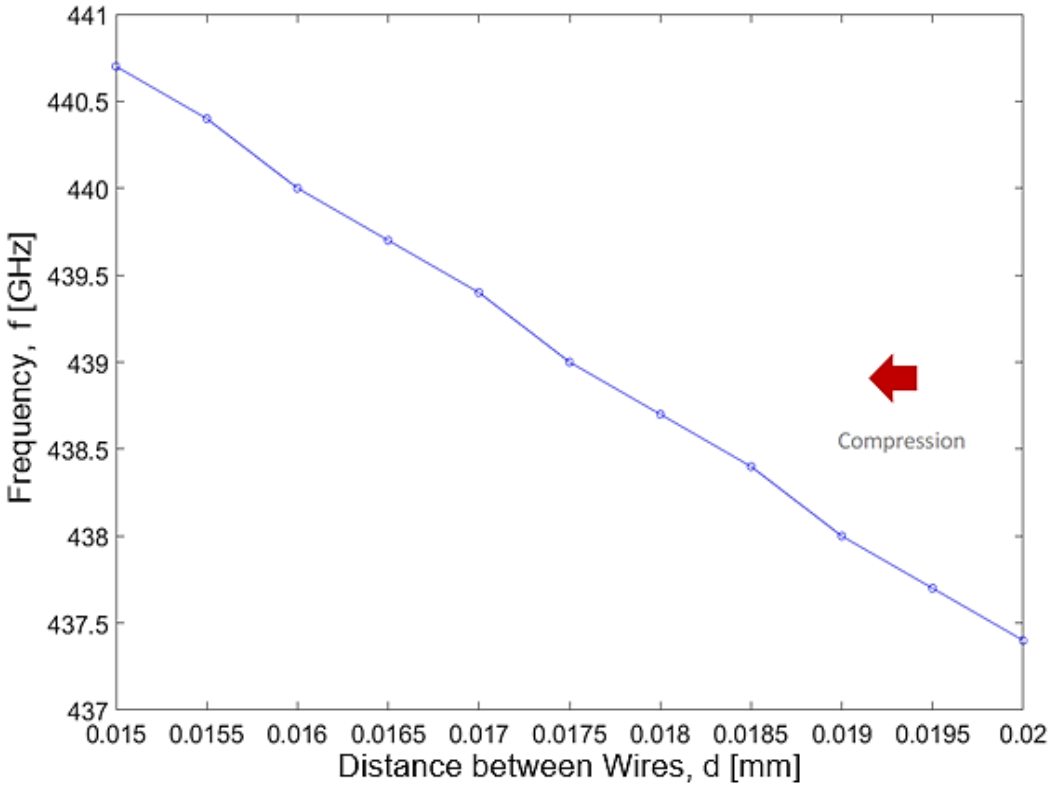

Figure 42 – Curve of the relation between the distance between wires and the resonance frequency, for each case of compression.

Also through Figure 41, the good dynamic range of the sensor is notorious, which is slightly better when compared to the case of the sensor assembled with HDPE. This is assumed, since the reflectance values for each case of compression in this typology are relatively smaller when compared with the values obtained for the previous case.

Here, as the resonance frequency changes, because of the reduction of the distance between wires, some saturation is also observed when the maximum compression state is approached, as shown in Figure 43.

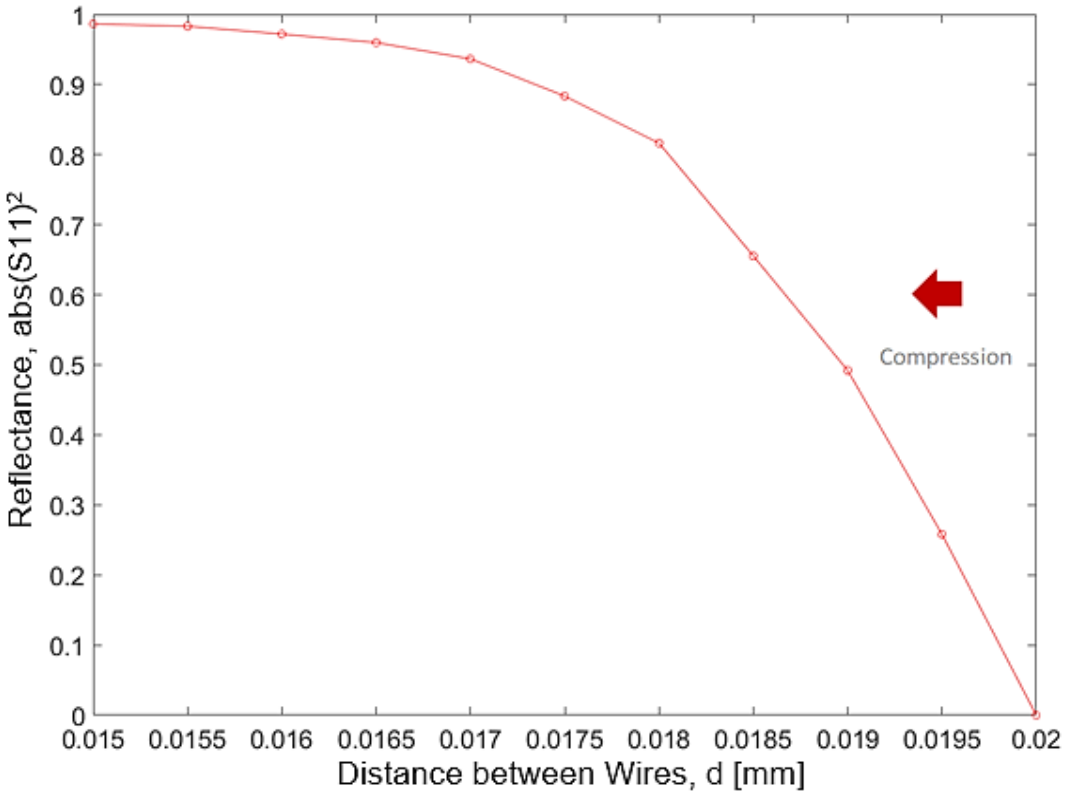

Figure 43 – Curve of the relation between the distance between wires and the reflectance, for each case of compression.

By analyzing Figure 41 and Figure 44, it is also possible to conclude that, as the compression is applied to the sensor and the frequency shift is verified, that the bandwidth decreases and the selectivity increases. In this way, it can be assumed that the higher the compression level applied, the greater the selectivity. Between the normal state in the absence of compression and the state in which there is a compression of 25%, there was a reduction of the bandwidth at 3dB from 2.2 GHz to 1.1 GHz, which represents a decrease of 50% compared to the initial case. The average bandwidth reduction value at 3dB is approximately 122MHz, as compression is applied to the sensor.

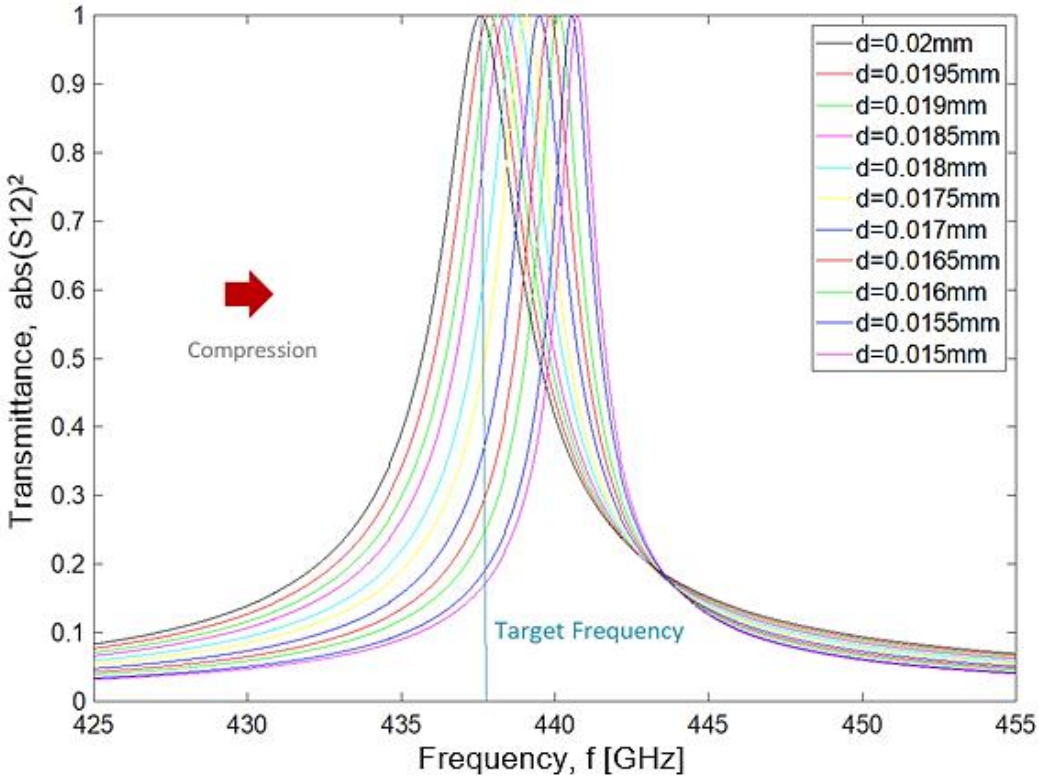

Figure 44 – Curves of the relation between frequency and transmittance, for each case of compression.

From the theoretical study and simulations, it was possible to understand that the parameters associated with the wires can be adjusted to allow only evanescent modes at the ends of the structure, allowing us to control the transmission and reflection of the energy.

Besides that, through the appropriate choice of the radius of the wire and the distance between the wires *d*, it is also possible to control the position of the modes in the diagram, which allows the design of highly selective filters with a high Q factor.

In addition, the inclusion of the proposed filter causes the transition of the sensor from situations in which it has a total opacity ($|S_{11}| = 1$) for situations in which it exhibits total transparency ($|S_{11}| = 0$).

4.2. Mechanical Analysis

After performing the analysis of the electromagnetic component, we will focus on the analysis of the mechanical component. In this section, we intend to understand how much force it will take to make the sensor compress up to 25% according to the materials used for its assembling and knowing that there is a possibility of controlling the compression magnetically, what will be the current required to reach each level of compression.

Although, Elmer contains in its library the material used for the wires (gold), unfortunately it does not have any of the other materials used as hypothesis in the sensor assembling. This implies a more careful search for the datasheets of each material in order to find the values associated to each of the parameters described in section 3.5.2 that characterize the materials in Elmer.

The gold in Elmer is characterized by the following properties, which can be observed in Table 1:

Table 1 – Simulation Parameters of gold in Elmer.

Density	19300 Kg/m^3
Young's Modulus	78 GPa
Poisson's Ratio	0.44

In the next topics, the values associated with each parameter, which characterize the other materials, will be presented, similarly to what was shown in Table 1. The results will be presented for each of the assembling hypothesis. It should be noted that each simulation has an average duration of 20 minutes and that 11 simulations were performed for each of the cases under analysis.

4.2.1 HDPE

Through the datasheets of the Engineering ToolBox website [54], the values associated with the following physical properties of the HDPE, which are present in Table 2, were discovered. This website is widely used to learn how to work with some tools and to obtain basic information for design, engineering and construction of technical applications, constituting a good reference for knowledge acquisition.

Table 2 – Simulation Parameters of HDPE in Elmer, taken from [54].

Density	641 Kg/m^3
Young's Modulus	0.8 GPa
Poisson's Ratio	0.46

Simulations were performed for a sensor with the characteristics mentioned in sub-chapter 3.3.1, which is composed of two square shaped cells of HDPE with gold wires. It was decided to observe the behaviour of the sensor as the number of wires in each array was increased and thus several scenarios with 10, 30, 50 and 70 wires per array were tested.

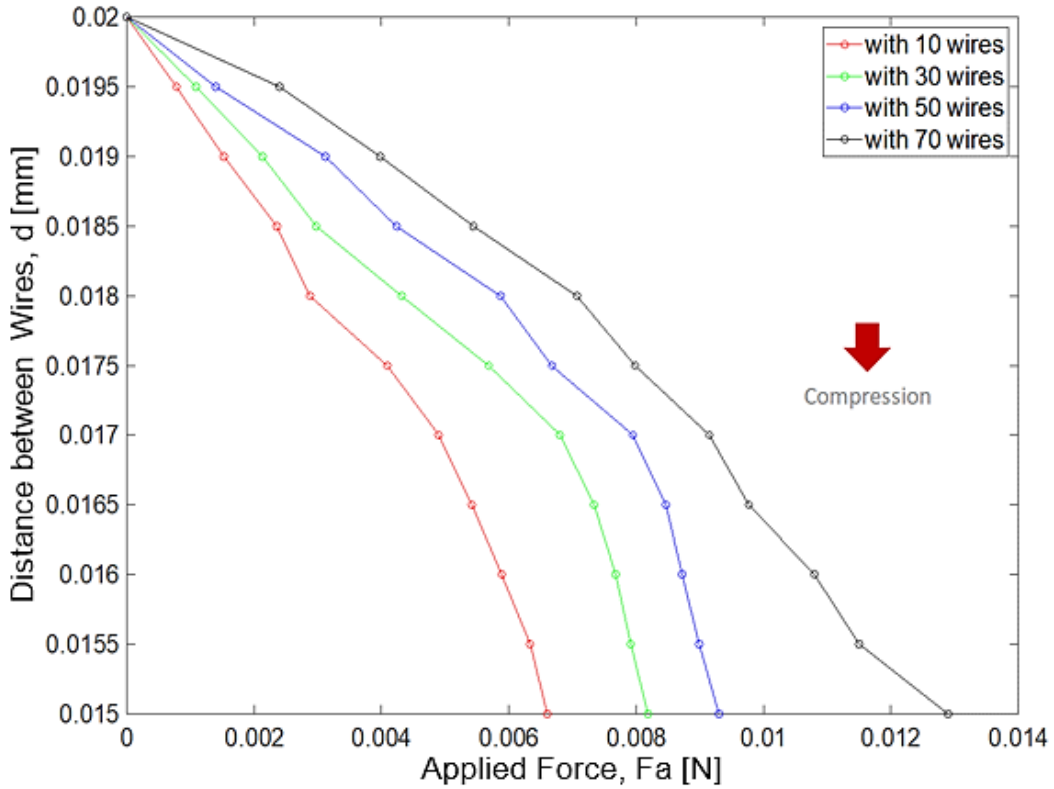

Figure 45 – Curves of the relation between the applied force and the distance between wires, for each case of compression

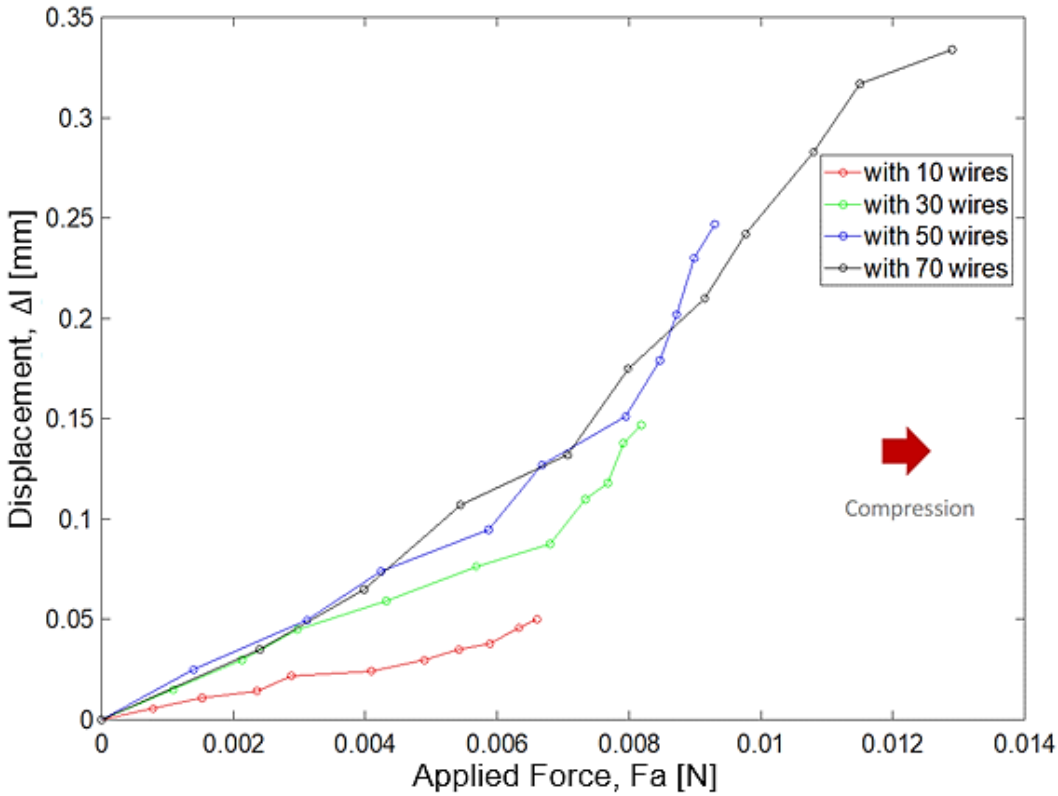

Figure 46 – Curves of the relation between the applied force and the displacement, for each case of compression.

Analyzing the Figure 45 and Figure 46, it is noted that for each scenario according to the number of wires, that the smaller the distance between wires the greater the force to be applied to reach the desired compression state. In addition, the higher the force applied the greater the displacement that will be required to obtain the intended wire spacing.

This behaviour is consistent with the values obtained for reflectance and transmittance as the wire spacing decreases, since the greater the force the greater / lesser the value associated with each of these magnitudes, as shown in Figure 47 and Figure 48. Here, the curves of the relationship between the applied force and the reflectance, as well as the relation between the applied force and the transmittance are presented.

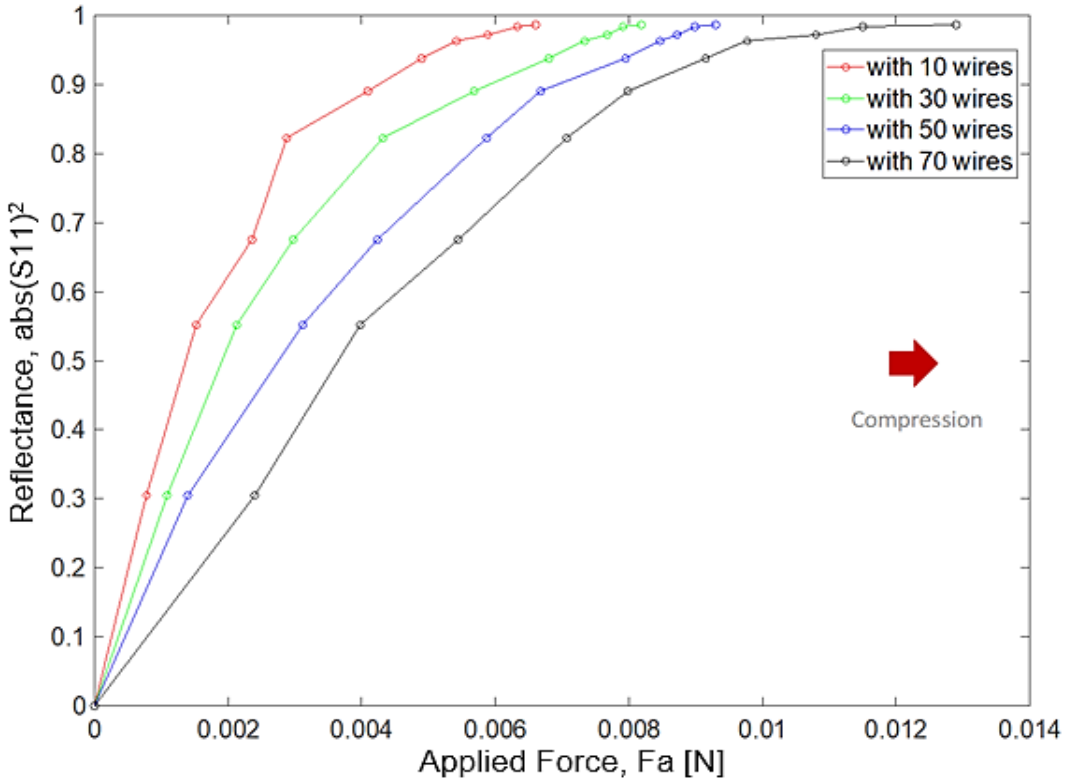

Figure 47 – Curves of the relation between the applied force and the reflectance, for each case of compression.

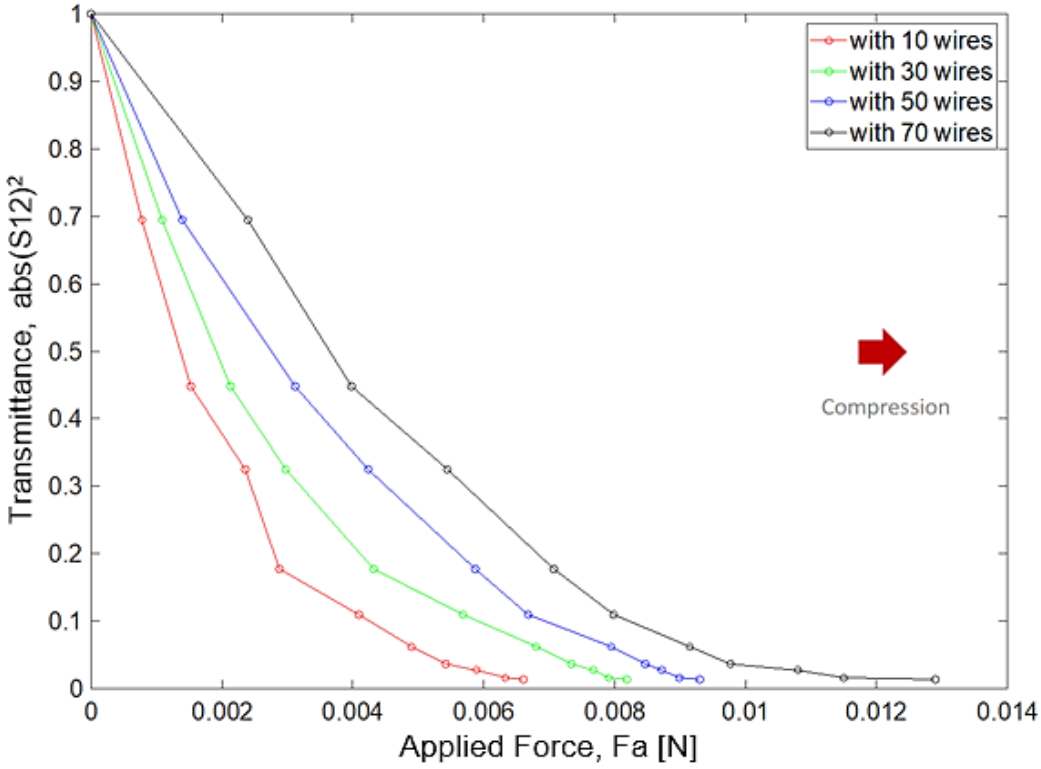

Figure 48 – Curves of the relation between the applied force and the transmittance, for each case of compression.

4.2.2 PTFE

Similarly, to the previous section, it was considered the same methodology to find the values associated to the following physical properties of PTFE. After consulting the datasheet of the Engineering ToolBox website [54], we figured out that PTFE is characterized by the properties which are present in Table 3:

Table 3 – Simulation Parameters of PTFE in Elmer, taken from [54].

Density	2100 Kg/m^3
Young's Modulus	0.3 GPa
Poisson's Ratio	0.46

The mechanical simulations were performed for a sensor with the characteristics mentioned in sub-chapter 3.3.1 and used in the previous section. It was also decided to observe the behaviour of the sensor as the number of wires in each array was increased and thus several scenarios with 10, 30, 50 and 70 wires per array were tested.

Analyzing Figure 49 and Figure 50, it is noted that for each scenario according to the number of wires, the smaller the distance between wires the greater the force to be applied to reach the desired compression state. In addition, the higher the force applied the greater the displacement that will be required to obtain the intended wire spacing.

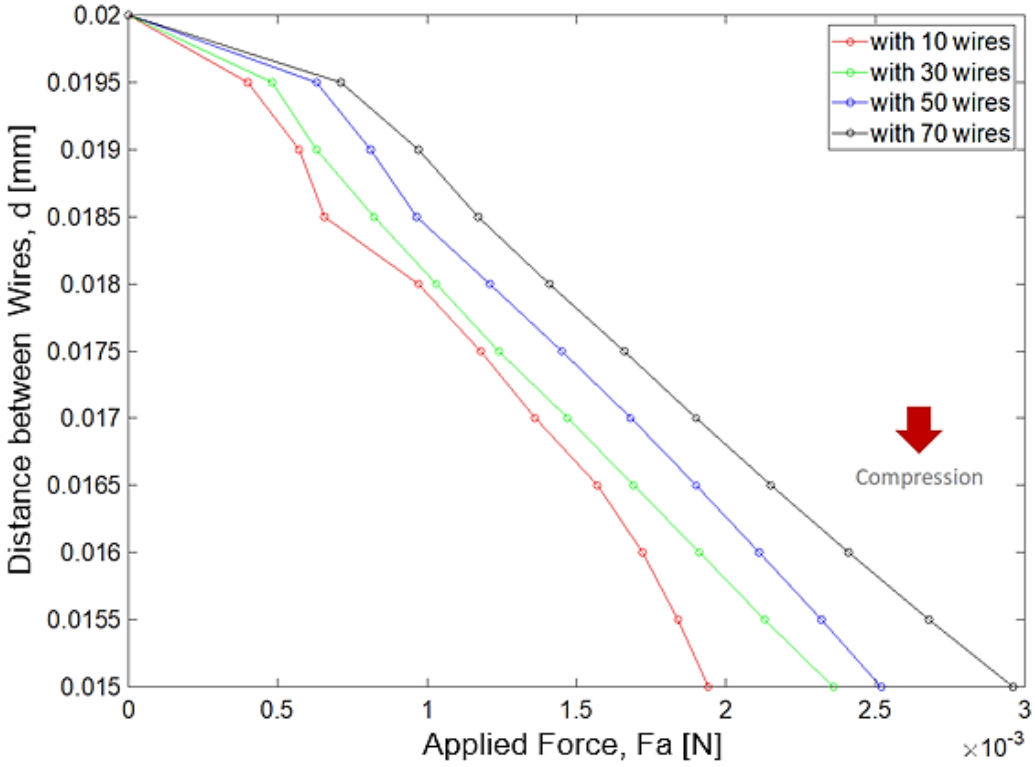

Figure 49 – Curves of the relation between the applied force and the distance between wires, for each case of compression.

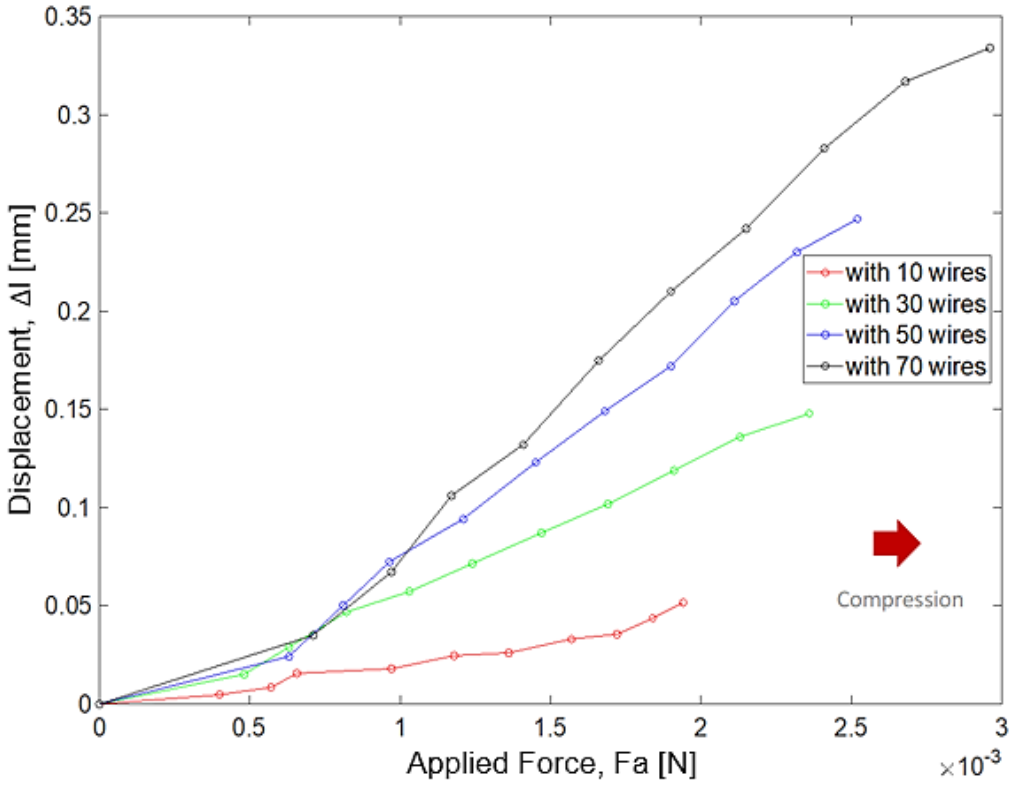

Figure 50 – Curves of the relation between the applied force and the displacement, for each case of compression.

This behaviour is consistent with the values obtained for reflectance and transmittance as the wire spacing decreases, since the greater the force the greater / lesser the value associated with each of these magnitudes, as shown in Figure 51 and Figure 52. Here, the curves of the relationship between the applied force and the reflectance, as well as the relation between the applied force and the transmittance are presented.

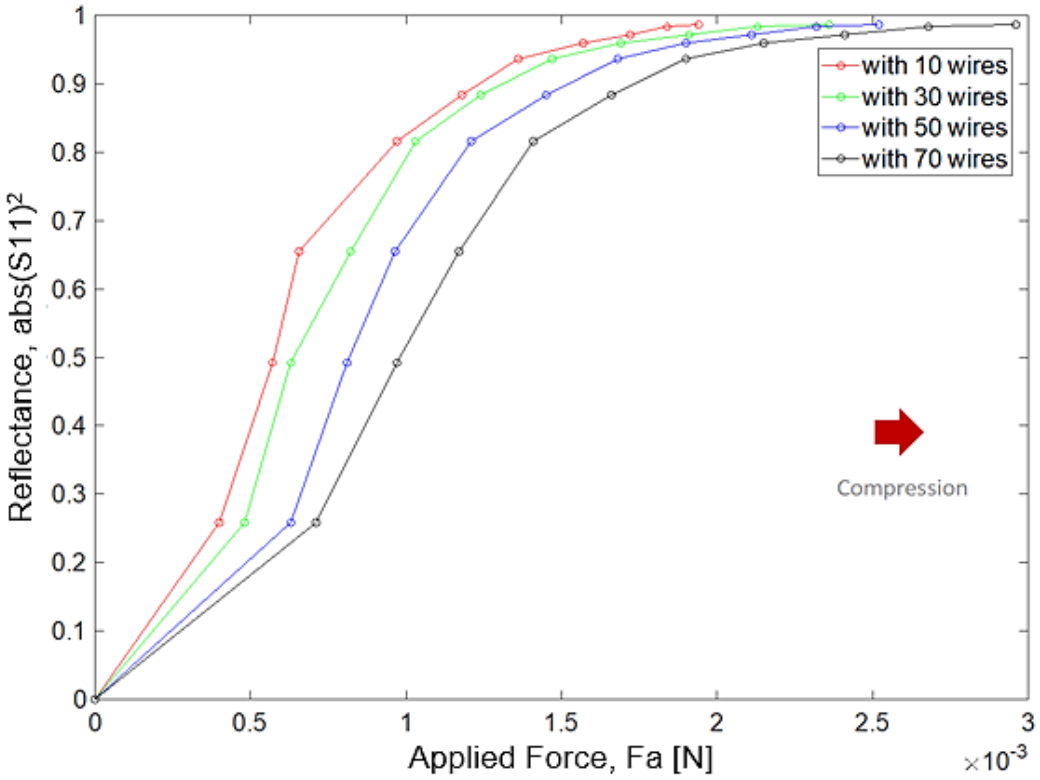

Figure 51 – Curves of the relation between the applied force and the reflectance, for each case of compression.

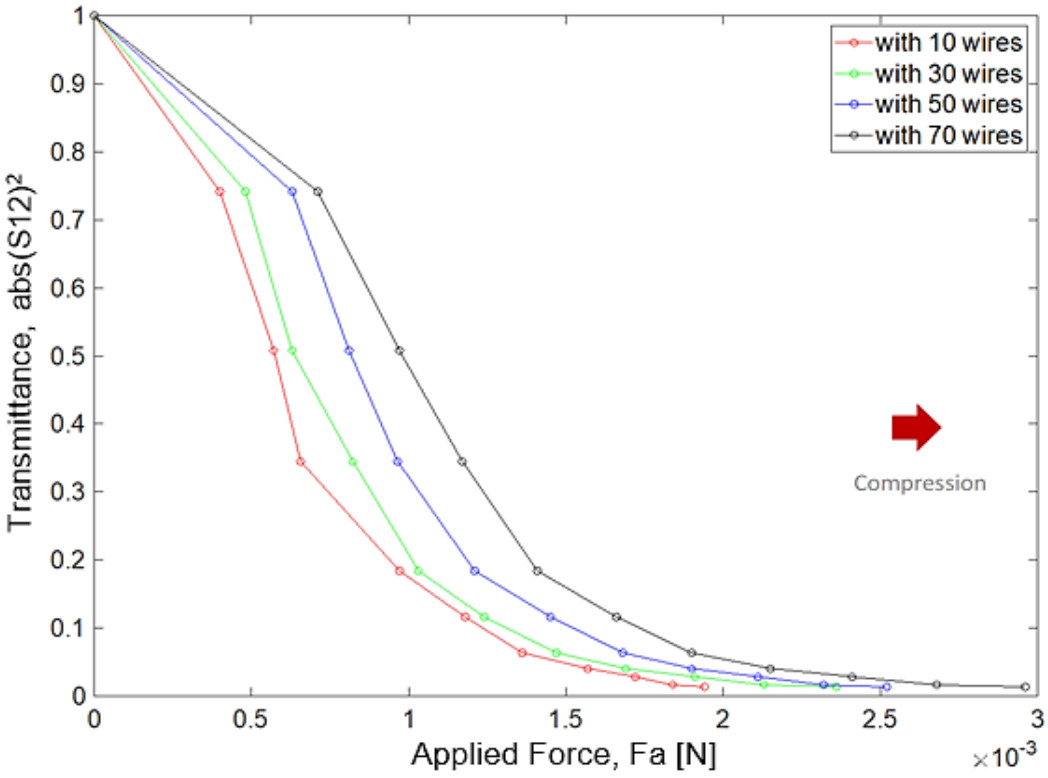

Figure 52 – Curves of the relation between the applied force and the transmittance, for each case of compression.

However, given that PTFE is a more flexible material than HDPE, it is notable that less force will be required to achieve the same wire spacing compared to the case of the sensor assembled with gold wire and HDPE. In the next sub-chapter, it will be discussed how the range of forces to compress the sensor will be generated, for each one of the analyzed cases.

4.2.3 Force Generation

The levels of the range of forces necessary to compress the sensors, which were mentioned in the previous sub-chapters, can be generated by passing an electric current through a coil placed on the plunger.

The plunger was simulated according to the description in section 3.3.3, considering the data in Table 4, in order to obtain the levels of the force range for each material.

Table 4 – Simulation Parameters of the Magnetic Circuit in FEMM.

Coil Length	0.27 mm
Coil Width	0.1 mm
Electric Condutivity of Iron	10.44 MS/m (default value at FEMM material's library)
Gap between Coil and Plunger	0.05 mm
Number of Turns of the Coil	1000
Plunger Length	0.37 mm
Plunger Width	0.05 mm

From the theory presented by Fitzgerald et all, the magnetic circuit associated with the problem at hand can be obtained using the following equation [55]:

$$\oint_C \boldsymbol{H} d\boldsymbol{l} = \sum_i^n R_i \Phi_i = M = NI \quad (125)$$

Where:

- C is the contour of integration;
- $\boldsymbol{H}$ is the magnetic field along the contour of integration;
- R is the reluctance, which can be defined as the ratio of the magnetic potential difference to the corresponding flux. Reluctance is obtained by dividing the length of the magnetic path $\boldsymbol{l_i}$ by the permeability of the cross-sectional area $\boldsymbol{A_i}$;
- M is the magnetomotive force, which is equal to the sum of the current concatenated to the contour of integration, being equivalent to the number of turns of the coil multiplied by the current.

The relation between the magnetic induction **B** and the magnetic field **H** depends on the material used and can be expressed as [55]

$$\mathbf{B} = \mu \mathbf{H}. \quad (126)$$

In the case of ferromagnetic materials this relation is non-linear, but in the case of the air it is linear. Furthermore $\mu_{air} \ll \mu_{iron}$.

The magnetic flux can be defined as [55]

$$\Phi_{\boldsymbol{B}} = \int \mathbf{B} \cdot \mathrm{d}\mathbf{A} = \mathrm{BAcos}(\theta), \quad (127)$$

where **A** is the area vector; its magnitude is the area of the loop, and its direction is perpendicular to the area of the loop, and ϴ is the angle between **A** and the magnetic induction field **B**. It is also important to point out that the magnetic flux ($\Phi_{\boldsymbol{B}}$) is only equal to $\mathrm{BAcos}(\theta)$**,** when the field is uniform over the entire loop. Finally, the energy stored in the material is equal to [55]

$$E = \frac{1}{2} l_i A_i \mu_{air} H^2. \quad (128)$$

Assuming $\mu_{air} \ll \mu_{iron}$, then all the magnetic energy stored in the system should be concentrated in the air gap. However, this is not what it is observed in the simulations. Despite that, this simplifying assumption is fine for getting a rough idea of how the system behaves.

After running the simulations, the results of each case show the same type of pattern, which is observed in Figure 53 and Figure 54:

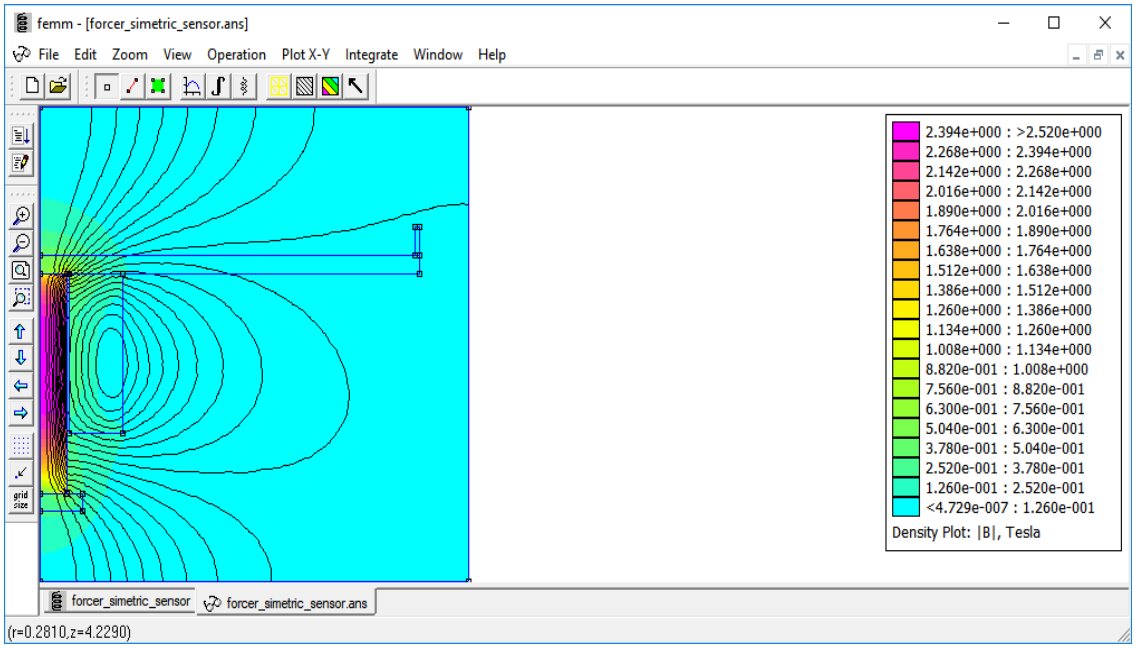

Figure 53 – Magnetic Induction Field Density with a current of 0.117461 A on FEMM.

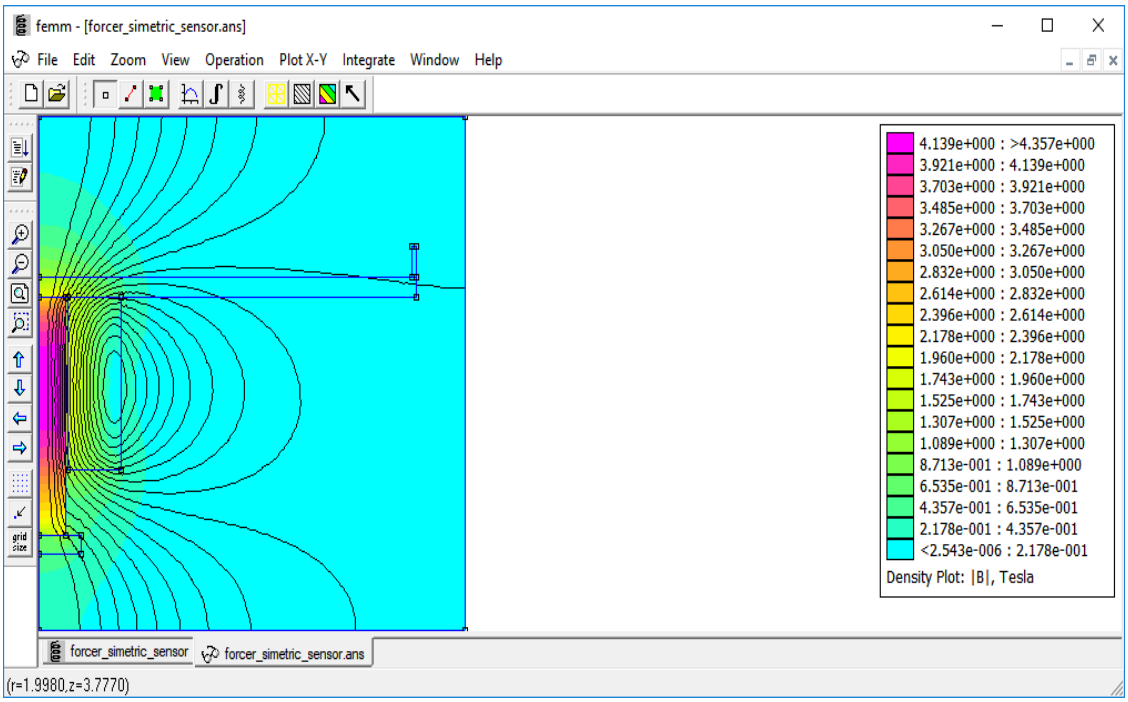

Figure 54 – Magnetic Induction Field Density with a current of 0.585975 A on FEMM.

It is possible to observe a leakage flux, since some flux lines are outside the magnetic element and the induction field is stronger in the iron element, becoming weaker near the air gap. The flux was calculated by using the integration function of FEMM.

Another interesting fact that is also observed in both figures is the variation of the magnetic induction field as the electric current increases. This happens because a changing current creates a changing magnetic field. The coil produces magnetic flux through itself and any change in the current in the coil gives originates a variation of this flux, producing electromotive force induced in the coil, which opposes the current change.

According to Biot-Savart's law, the magnetic field produced by the coil is directly proportional to the current. Then, the magnetic flux produced by the coil on itself is proportional to the current [56]

$$\Phi_B = LI, \tag{129}$$

where L is the inductance of the coil and I is the current. The electromotive force induced in the coil can be expressed as [56]

$$\varepsilon = -L\frac{\partial I}{\partial t}, \tag{130}$$

which is equivalent to the product of the number of turns of wire by the rate of change of the magnetic flux in a single loop.

The greater the area of the turns in the coil, the greater its inductance. The larger the number of turns, *n*, in the coil, the greater will be the area crossed by the field lines and the larger the magnetic field itself. Thus, the inductance of a coil with *n* turns is directly proportional to n^2 [56]. The inductance of the coil is n^2 times the inductance of each of its turns separately:

$$L_{coil} = n^2 L_{turn}. \quad (131)$$

After the simulation, FEMM is also able to define the relation between the magnetic induction B and the magnetic field H, based on the characteristics of the ferromagnetic material, which can be observed in Figure 55:

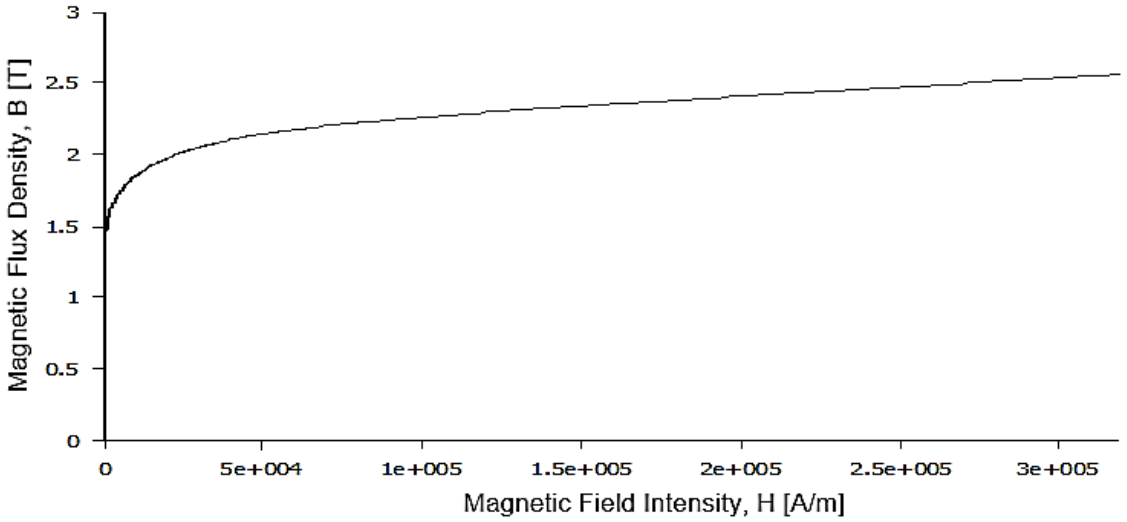

Figure 55 – Curve of the relation between B and H obtained on FEMM.

As it is possible to see in Figure 55, the curve of the relation between B and H is non-linear, which will imply a considerable variation of the behaviour of the magnetic flux since the magnetic induction field is not uniform.

So far, in discussing the theory related with the magnetic circuit, we understood some particularities of ferromagnetic materials such as iron and how to introduce a variation of the magnetic induction field through electric current. Now let us discuss the theory behind the generation of the magnetic force acting on the plunger.

Assuming the only paths of reluctance are the plunger and the air gap, as it is suggested by Slocum, we can determine both expressions and establish a relation with the magnetic flux of the circuit [57]. The reluctance of the air gap is given by

$$R_{air} = \frac{g}{\mu_0 A_p}, \quad (132)$$

where g is the length of the gap, A_p is the sectional area of the plunger and μ_0 is the magnetic permeability of free space, $4\pi\times10^{-7}$ H/m. The reluctance of the plunger, in turn, is given by [57]

$$R_{plunger} = \frac{l-g}{\mu A_p}, \quad (133)$$

where l is the length of the coil. Considering that the magnetomotive force is NI, and the reluctances are placed in series, so the magnetic flux can be expressed as

$$\Phi_B = \frac{NI}{\frac{g}{\mu_0 A_p}+\frac{l-g}{\mu A_p}} \approx \frac{\mu_0 N I A_p}{g}. \quad (134)$$

Through this information the magnetic force that the system can generate, which is acting on the plunger, can be expressed as [57]

$$F = \frac{{\Phi_B}^2}{2\mu_0 A_p} = \frac{\mu_0 A_p N^2 I^2}{2g^2}. \quad (135)$$

Through this formula it is possible to know that the force is proportional to the current and to the number of turns and inversely proportional to the gap length. However, this does not mean that it is easy to predict the strength analytically, since the relation between B and H is not linear. This problem can be solved using numerical simulation and thus the force can be calculated.

The range of forces to be applied to compress the sensor was found through numerical simulation on Elmer, so the goal will be to find out the intensity of current to provide to the circuit to achieve the desired compression. The simulations on FEMM were performed for both cases, being the results shown in Figure 56 and Figure 57:

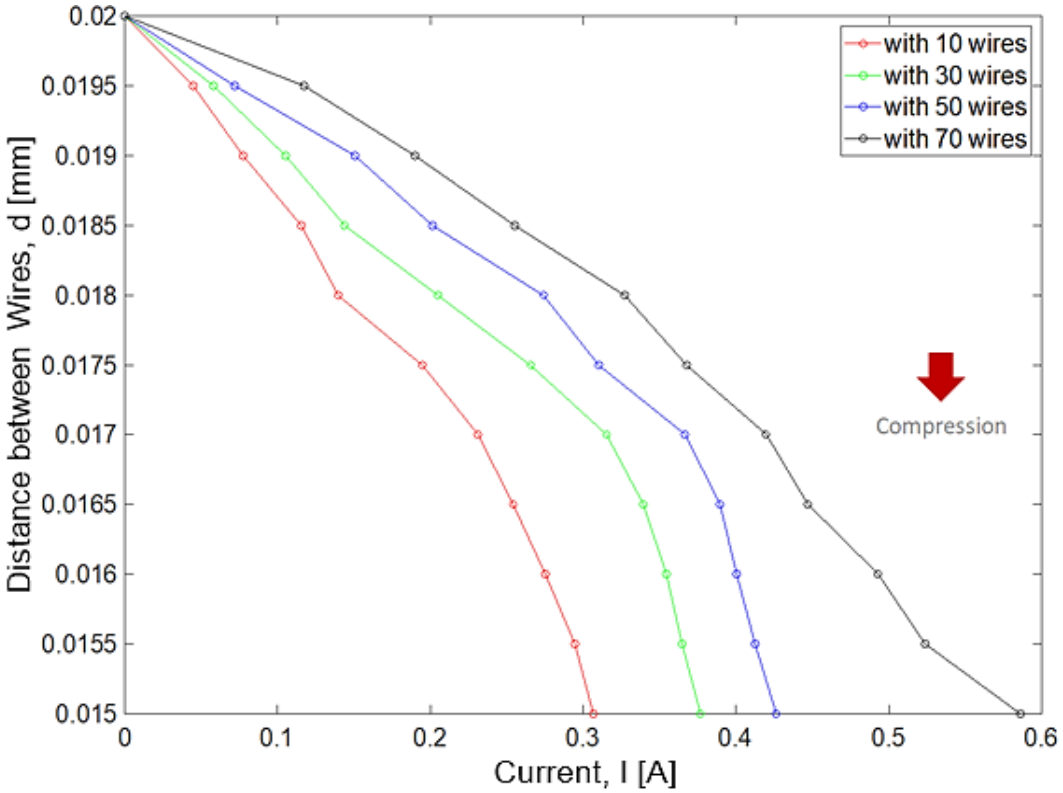

Figure 56 – Required Current for each level of compression for a sensor assembled with gold wires and HDPE.

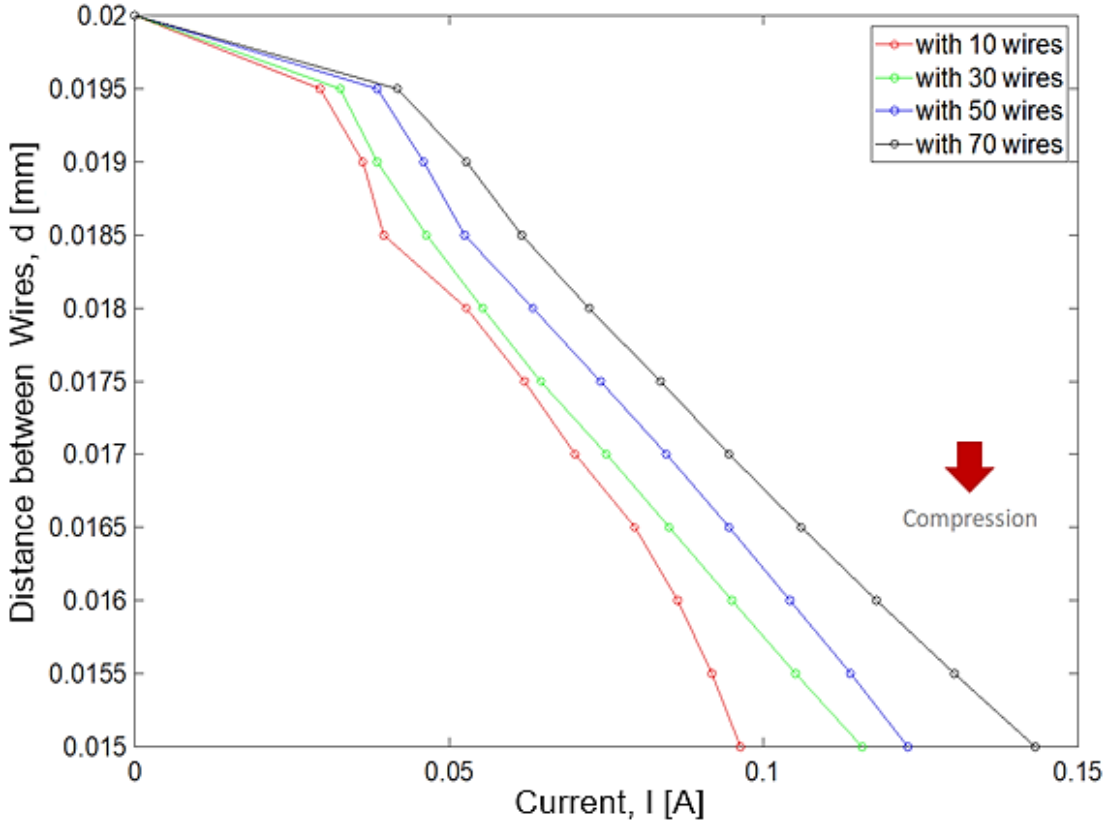

Figure 57 – Required Current for each level of compression for a sensor assembled with gold wires and PTFE.

Similarly, to the behaviour described in the graphs of the applied force vs distance between wires with the data from Elmer, it is observed that the higher the force to be applied to the sensor the higher the current to be supplied to the magnetic circuit. As the range of forces to be applied for each typology is different, also the range of currents to be supplied will be different.

Another interesting fact affecting the force produced by the plunger is the positioning of the coil therein. According to the geometry of the problem, the movement of the coil will cause a change to the magnetic induction field, which in turn will cause the generated force to be smaller. Since the plunger has a length of 0.37mm and that the coil has a length of 0.27mm, only 0.1mm remains to allow the coil to be displaced.

In Figure 58 and Figure 59, the evolution of the force generated by the plunger is shown as the coil is moved from its initial position. For the elaboration of these graphs, it was considered a situation in which the sensor presents 70 wires in each array and the supply of a constant electric current, according to the typology of the sensor.

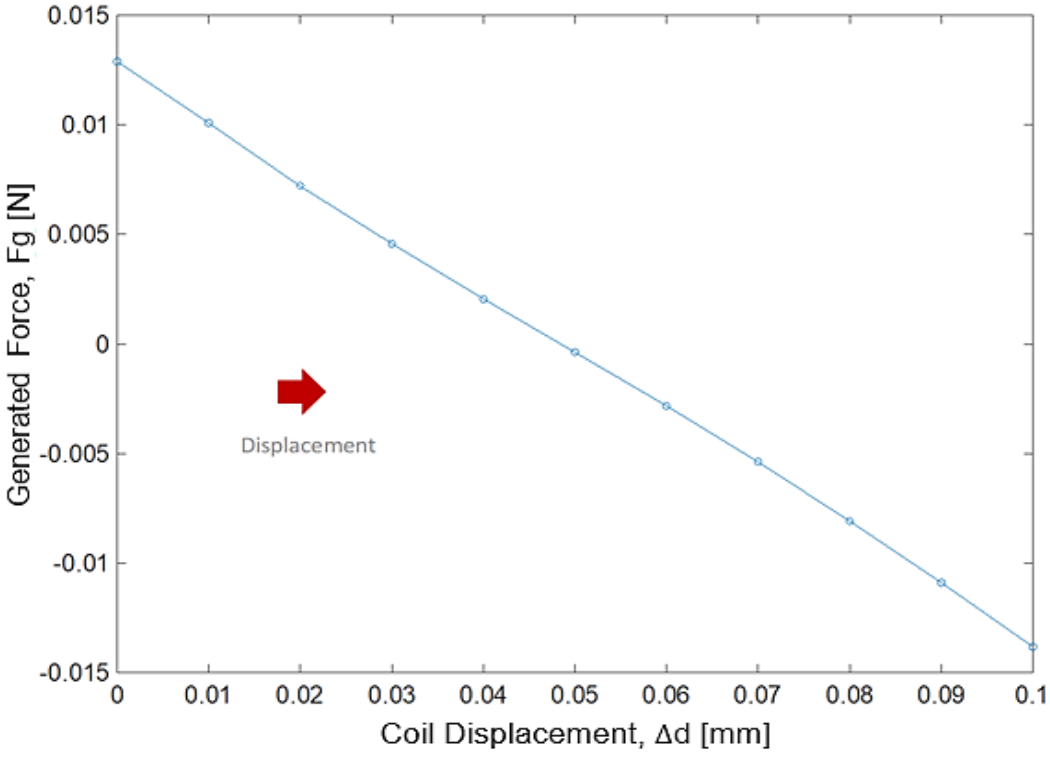

Figure 58 – Curve of the relation between the generated force and the displacement of the coil, with a current of 0.585975 A (sensor assembled with gold wires and HDPE).

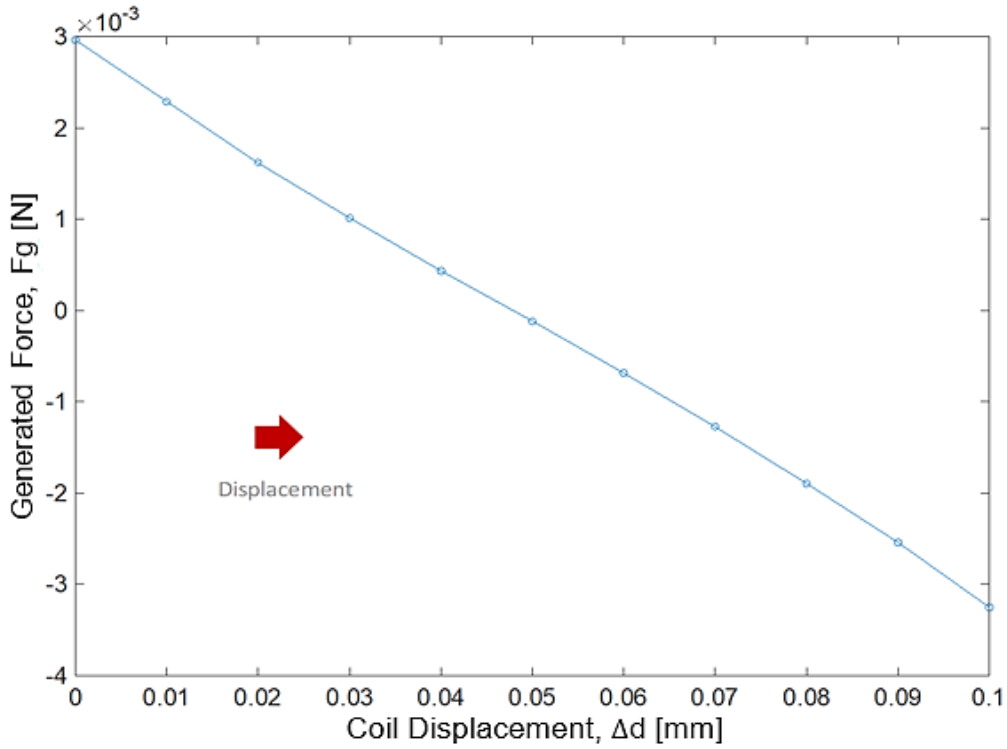

Figure 59 – Curve of the relation between the generated force and the displacement of the coil, with a current of 0.117461 A (sensor assembled with gold wires and PTFE).

In both cases, the force reduction as the coil moves away from the initial position of the circuit is approximately linear, however the force is only compressive for coil displacements up to 0.0478mm, according with the analysis of the data and simulations. From this situation the force acquires a direction contrary to that of the desired movement, that is, instead of the force pushing the plunger outwards it retracts it.

The force generated for the 0.01, 0.02, 0.03 and 0.04mm coil displacement positions correspond to the following distances between wires, which can be seen in Table 5 and Table 6:

Table 5 – Calculation of the displacement of the sensor and the distance between wires, for a sensor assembled with gold wires and HDPE.

Displacement of the Coil [mm]	**Generated Force [N]**	**Displacement of the sensor [mm]**	**Distance between wires [mm]**
0	0.0129	0.334	0.015
0.01	0.0100944	0.2564	0.01633714
0.02	0.00722222	0.1393	0.01801
0.03	0.00456238	0.08359	0.018806
0.04	0.00203236	0.03074	0.01956085

Table 6 – Calculation of the displacement of the sensor and the distance between wires, for a sensor assembled with gold wires and PTFE.

Displacement of the Coil [mm]	Generated Force [N]	Displacement of the sensor [mm]	Distance between wires [mm]
0	0.00296	0.334	0.015
0.01	0.00228718	0.267	0.01618571
0.02	0.0016215	0.16655	0.01762071
0.03	0.00101231	0.07582	0.01891686
0.04	0.000432117	0.01995	0.019715

The displacement of the sensor and the distance between wires were calculated based on the data obtained for the graphs of Figure 45, Figure 46, Figure 49 and Figure 50. For a coil displacement position of 0.0478mm, the force generated is so small that when applied to the sensor it will lead to a very small displacement. The magnitude of this displacement is so small that it can be considered to be practically 0, which means that the distance between wires is practically the same as that of the case without compression.

The curve of the relation between the generated force and the displacement of the sensor for both typologies, can be seen in Figure 60 and Figure 61. The first point of both graphs corresponds to the state without compression and the last point corresponds to the state with 25% of compression.

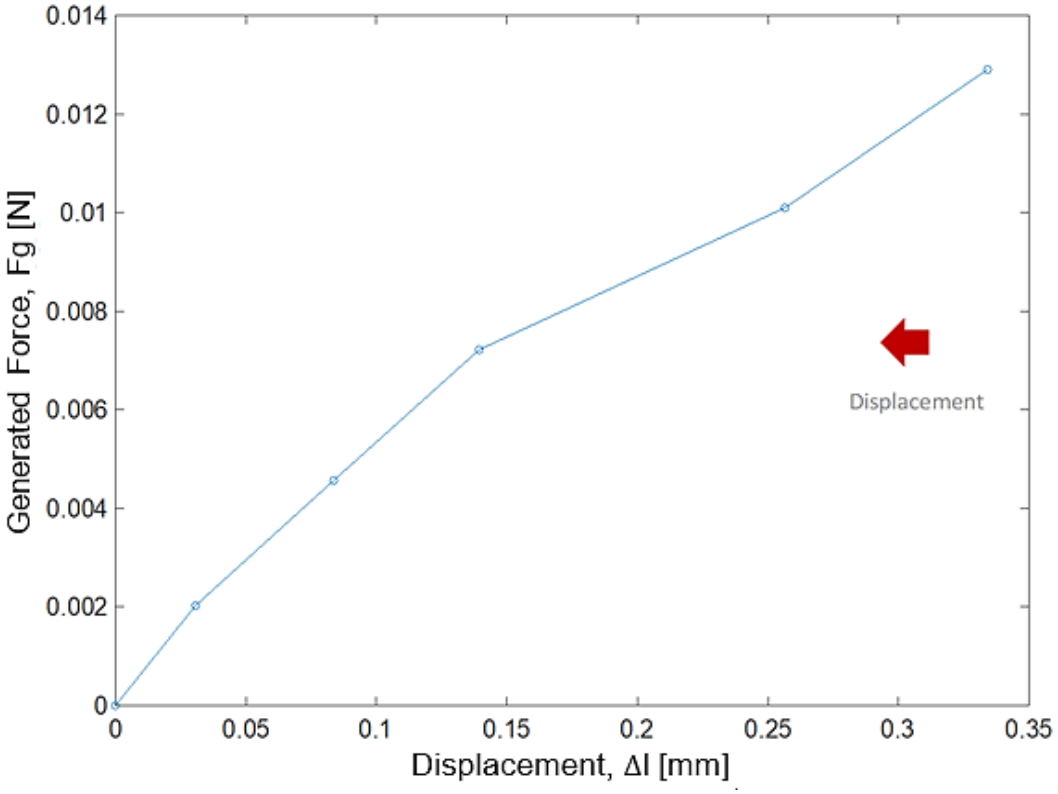

Figure 60 – Curve of the relation between the generated force and the displacement of the sensor, with a current of 0.585975 A (sensor assembled with gold wires and HDPE).

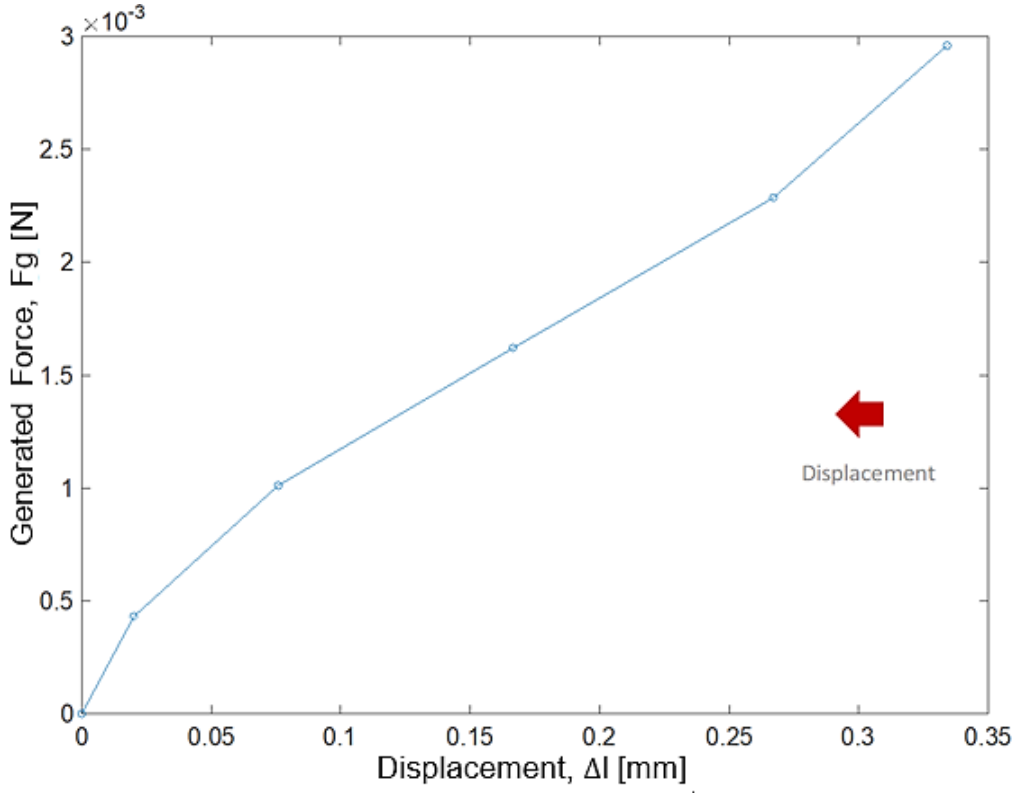

Figure 61 – Curve of the relation between the generated force and the displacement of the sensor, with a current of 0.117461 A (sensor assembled with gold wires and PTFE).

Therefore, it is concluded that in addition to being able to control the level of force applied to the sensor through the current supplied to the circuit, it is also possible to carry out this control by displacing the coil. In this particular case, the displacement of the coil led to the discovery of new force levels to be applied within each of the ranges under study. The behaviour observed in the graphs would be exactly the same for cases with 10, 30 and 50 wires. For the same number of wires, if we reduce the current supplied to the circuit it is expected that the applied compression level will be lower and therefore the coil displacement will also be smaller.

After performing this study of characterization, design and simulation of the devices, its working principle was confirmed and the evolution of its response, when different levels of compression are applied, was verified. This type of sensors is also more sensitive and selective in frequency compared to the stress sensors described in section 2.3.5. In fact, they can even be easily integrated on the structures to detect cracks and other damages based on the variation of the transmittance of the sensors.

Capitulo 5 – Conclusions and Future Work

In this chapter, a brief summary containing the main conclusions and possible work extensions are presented.

5.1. Summary

This dissertation aimed to study THz filter design, using frequency selective structures based on metamaterial resonators, so that it can be used in the development of sensors for SHM applications.

THz filters can be designed resorting to the microwave network analysis theory, since its characteristics allow to modify the original problem or to combine several elements together in order to find the desired filter response without having to reanalyze the behaviour of each element in combination with its neighbors.

In a first approach, we began by studying the propagation electromagnetic waves on a dielectric slab and, later, we studied the propagation phenomena when the arrays of wire are added to the slab. Through this study, we proved that each array of wires behaves as an admittance and we found its corresponding expression. We also discovered that its introduction into a dielectric slab causes the cancellation of the odd harmonics, leaving only the harmonic pairs. In addition, the parameters that influenced the behaviour of admittance were found, which include: the radius of the wires (a), the distance between wires(d) and the length of the structure (l).

The proposed filter is composed by two arrays of wires to provide greater cancellation of harmonics, higher dynamic range and enhanced frequency selectivity. The resonant effects result from careful tuning the wire radius and the distance between wires, which can be altered so that only evanescent modes exist in the vicinity of the structure, allowing us to control the energy transmission and reflection. Due to its simplicity, this filter design is especially suited for the implementation of reconfigurable THz filters and optical modulators, since it transits from situations in which it presents a full opacity ($|S_{11}| = 1$) for a full transparency ($|S_{11}| = 0$).

A sensor was designed for SHM applications, encompassing the proposed THz filter. Two assembling hypotheses with different thermoplastic polymers materials were analyzed. The choice of these materials was essentially due to the fact that they are

relatively easy to compress and exhibit low losses and low dispersion for the frequency band over 0.1 to 5 THz.

The variation of the electromagnetic and mechanical response of the sensor was studied as compression was applied to reduce the distance between wires using the finite element method.

From the electromagnetic point of view, for both cases, it was verified that the behaviour of the sensor according to the theoretical model versus the result of the simulation corresponded to what was expected, although the resonance frequencies were different for each case. In this analysis, the variation of the reflection and transmission coefficients, the dynamic range and the bandwidth at 3dB were also discussed, as the applied compression caused a shift in the resonance frequency.

From the mechanical point of view, the force level to be applied and the displacement reached for each distance between wires were analyzed for both cases. As with the previous analysis and considering the characteristics of thermoplastic polymers materials, the amount of force to be applied for the same compression level will also be different. Force generation applied to the sensors can be accomplished using a magnetic circuit containing a controlled plunger and a coil. It was observed that two methodologies can be adopted to control the level of force applied to the sensors, namely, through the variation of the electric current and the positioning of the coil on the plunger.

5.2. Future Work

By following this dissertation research, some improvements can be made to add even more value to this work:

- Design and simulation of sensors using other thermoplastic polymer materials that can be used in the THz domain;
- Creation of new designs for the sensor, encompassing the THz filter in the structure;
- Selection of the best manufacturing technique for the production of the sensor for later measurements and its comparison to the results obtained from the simulations.

References

[1] F. Yuan, Structural Health Monitoring (SHM) in Aerospace Structures, Woodhead Publishing Series in Composite Sciences and Engineering, 2016.

[2] B. Ferguson and X. Zhang, "Materials for terahertz science and technology," *Nature Materials,* vol. 1, nº 1, pp. 26-33, 2002.

[3] S. Lucyszyn, Advanced RF MEMS, Cambridge University Press, 2010.

[4] B. A. Munk, Frequency selective surfaces theory and design, John Wiley & Sons, 2000.

[5] J. S. Fernandez, "Frequency selective surfaces for Terahertz applications, PhD Thesis," The University of Edinburgh, 2012.

[6] D. Wang, P. Zhao and C. Chan, "Design and Analysis of a High-Selectivity Frequency-Selective Surface at 60 GHz," *IEEE Transactions on Microwave Theory and Techniques,* vol. 64, nº 6, pp. 1694-1703, 2016.

[7] A. Melo, A. Gobbi, M. Piazzetta and A. da Silva, "Cross-Shaped Terahertz Metal Mesh Filters: Historical Review and Results," *Advances in Optical Technologies,* vol. 2012, pp. 1-12, 2012.

[8] B. Hooberman, "Everything You Ever Wanted to Know About Frequency-Selective Surface Filters but Were Afraid to Ask," University of Illinois Urbana-Champaign, 2005.

[9] F. Costa, A. Monorchio and G. Manara, "An Overview of Equivalent Circuit Modeling Techniques of Frequency Selective Surfaces and Metasurfaces," *Applied Computational Electromagnetics Society Journal,* vol. 29, nº 12, pp. 960-976, 2014.

[10] L. B. Whitbourn and R. C. Compton, "Equivalent circuit formulas for metal grid reflectors at a dielectric boundary," *Applied Optics,* vol. 24, pp. 217-220, 1985.

[11] S. T. Shanahan and N. R. Heckenberg, "Transmission line model of substrate effects on capacitive mesh couplers," *Applied Optics,* vol. 20 , pp. 4019-4023, 1981.

[12] T. Timusk and P. L. Richards, "Near millimeter wave bandpass filters," *Applied Optics,* vol. 20, pp. 1356-1359, 1981.

[13] W. Hayt, J. Kemmerly and S. Durbin, Engineering circuit analysis, McGraw-Hill, 2012.

[14] A. K. Rashid, B. Li and Z. Shen, "An overview of three-dimensional frequency-selective structures," *IEEE Antennas and Propagation Magazine,* vol. 56, nº 3, pp. 43-67, 2014.

[15] R. Mittra, C. Chan and T. Cwik, "Techniques for analyzing frequency selective surfaces - A review," *Proceedings of the IEEE,* vol. 76, nº 12, pp. 1593-1615, 1988.

[16] R. Anwar, L. Mao and H. Ning, "Frequency Selective Surfaces: A Review," *Applied Sciences,* vol. 8, nº 9, pp. 1689-1736, 2018.

[17] A. Güemes, "SHM Technologies and Applications in Aircraft Structures," *Proceedings of the 5th International Symposium on NDT in Aerospace,* p. 13–15, 2013.

[18] S. Dwivedi, M. Vishwakarma and P. Soni, "Advances and Researches on Non-Destructive Testing: A Review," *Materials Today: Proceedings,* vol. 5, nº 2, pp. 3690-3698, 2018.

[19] S. Doebling, C. Farrar and M. Prime, "A Summary Review of Vibration-Based Damage Identification Methods," *The Shock and Vibration Digest,* vol. 30, nº 2, pp. 91-105, 1998.

[20] A. Plankis, "Structural health monitoring mems sensors using elasticity-based beam vibrations, MSc Thesis," Colorado State University, 2012.

[21] W. Fan and P. Qiao, "Vibration-based Damage Identification Methods: A Review and Comparative Study," *Structural Health Monitoring: An International Journal,* vol. 10, nº 1, pp. 83-111, 2010.

[22] P. Chang, A. Flatau and S. Liu, "Review Paper: Health Monitoring of Civil Infrastructure," *Structural Health Monitoring: An International Journal,* vol. 2, nº 3, pp. 257-267, 2003.

[23] A. Alvandi and C. Cremona, "Assessment of vibration-based damage identification techniques," *Journal of Sound and Vibration,* vol. 292, nº 1, pp. 179-202, 2006.

[24] P. Cawley, "Structural health monitoring: Closing the gap between research and industrial deployment," *Structural Health Monitoring,* vol. 17, nº 5, pp. 1225-1244, 2018.

[25] L. Kang, D. Kim and J. Han, "Estimation of dynamic structural displacements using fiber Bragg grating strain sensors," *Journal of Sound and Vibration,* vol. 305, nº 3, pp. 534-542, 2007.

[26] N. Nor, Structural health monitoring through acoustic emission, Woodhead Publishing Series in Civil and Structural Engineering, 2018, pp. 123-146.

[27] A. Moreno-Gomez, C. A. Perez-Ramirez, A. Dominguez-Gonzalez, M. Valtierra-Rodriguez, O. Chavez-Alegria and J. P. Amezquita-Sanchez, "Sensors Used in Structural Health Monitoring," *Archives of Computational Methods in Engineering,* 2017.

[28] L. Zhu, Y. Fu, R. Chow, B. Spencer, J. Park and K. Mechitov, "Development of a High-Sensitivity Wireless Accelerometer for Structural Health Monitoring," *Sensors,* vol. 18, pp. 262-278, 2018.

[29] G. Manthei, "Characterisation of Acoustic Emission Sensors," *Proceedings of EWGAE 2010 - The 29th European Conference on Acoustic Emission Testing,* 2010.

[30] J. Li, H. Hao, K. Fan and J. Brownjohn, "Development and application of a relative displacement sensor for structural health monitoring of composite bridges," *Structural Control and Health Monitoring,* vol. 22, nº 4, pp. 726-742, 2014.

[31] J. Lopez-Higuera, L. Rodriguez Cobo, A. Quintela Incera and A. Cobo, "Fiber Optic Sensors in Structural Health Monitoring," *Journal of Lightwave Technology,* vol. 29, nº 4, pp. 587-608, 2011.

[32] M. Krüger, C. Grosse and P. Marrón, "Wireless Structural Health Monitoring Using MEMS," *Key Engineering Materials,* vol. 293, pp. 625-634, 2005.

[33] A. Thomas, K. Pradeep and P. Mathew, "Structural Health Monitoring Using Piezoelectric Sensors and Actuators," *Applied Mechanics and Materials,* vol. 857, pp. 255-260, 2016.

[34] H. Choi, S. Choi and H. Cha, "Structural Health Monitoring system based on strain gauge enabled wireless sensor nodes," *Proceedings of The 5th International Conference on Networked Sensing Systems,* 2008.

[35] D. Ha, H. Park, S. Choi and Y. Kim, "A Wireless MEMS-Based Inclinometer Sensor Node for Structural Health Monitoring," *Sensors,* vol. 13, nº 12, pp. 16090-16104, 2013.

[36] C. Duque and M. A. Ribeiro, "Effect of uniaxial compression in the transmittance of a novel terahertz sensor for structural health monitoring," *Proceedings of Loughborough Antennas and Propagation Conference,* 2013.

[37] J. P. Pavia, W. J. Otter, S. Lucyszyn and M. A. Ribeiro, "Design of a THz-MEMS Frequency Selective Surface for Structural Health Monitoring," *Proceedings of META'16 - The 7th International Conference on Metamaterials, Photonic Crystals and Plasmonics,* 2016.

[38] B. Ozbey, E. Unal, H. Ertugrul, O. Kurc, C. Puttlitz, V. Erturk, A. Altintas and H. Demir, "Wireless Displacement Sensing Enabled by Metamaterial Probes for Remote Structural Health Monitoring," *Sensors,* vol. 14, nº 1, pp. 1691-1704, 2014.

[39] S. Fan, T. Li, J. Zhou, X. Liu, H. Qi and Z. Mu, "Terahertz non-destructive imaging of cracks and cracking in structures of cement-based materials," *AIP Advances,* vol. 7, nº 11, pp. 115202-115213, 2017.

[40] J. Chen and S. Kaushik, "Terahertz Interferometer That Senses Vibrations Behind Barriers," *IEEE Photonics Technology Letters,* vol. 19, nº 7, pp. 486-488, 2007.

[41] W. J. Otter, F. Hu, S. Hanham, A. S. Holmes, W. T. Pike, N. Klein, M. A. Ribeiro and S. Lucyszyn, "Terahertz Metamaterial Devices," *Proceedings of The International Conference on Semiconductor Mid-IR and THZ Materials and Optics – SMMO 2016,* 2016.

[42] D. M. Pozar, Microwave Engineering, John Wiley & Sons, 2012.

[43] K. W. Whites, "EE 481/581 - Microwave Engineering," South Dakota School of Mines & Technology, 2014.

[44] D. Cheng, Field and Wave Electromagnetics, Addison-Wesley, 1989.

[45] M. Naftaly and R. Dudley, "Introduction to Terahertz - The ins and outs of terahertz technologies, what it does and how it works.," NPL - National Physics Laboratory, 2017.

[46] N. Hinsley, “The History of Polyethylene,” Globalplasticsheeting.com, 2018. [Online]. Available: https://www.globalplasticsheeting.com/our-blog-resource-library/bid/23095/The-History-of-Polyethylene. [Accessed on 25 July 2018].

[47] “HDPE | Plastics International,” Plasticsintl.com, 18. [Online]. Available: http://www.plasticsintl.com/hdpe.htm. [Accessed on 25 July 2018].

[48] “Roy J. Plunkett,” Science History Institute, 2018. [Online]. Available: https://www.sciencehistory.org/historical-profile/roy-j-plunkett. [Accessed on 25 July 2018].

[49] “PTFE | Plastics International,” Plasticsintl.com, 2018. [Online]. Available: http://www.plasticsintl.com/ptfe.html. [Accessed on 25 July 2018].

[50] J. Reddy, An introduction to the finite element method, McGraw-Hill, 2006.

[51] J. Jin, The Finite Element Method in Electromagnetics, John Wiley & Sons, 2002.

[52] W. H. G. Lewin, “Elasticity - Young's Modulus,” MIT - Massachusetts Institute of Technology, 2008.

[53] “Finite Element Method Magnetics: FEMM 4.2 Magnetostatic Tutorial,” Femm.info, 2018. [Online]. Available: http://www.femm.info/wiki/magneticstutorial. [Accessed on 25 July 2018].

[54] “Engineering ToolBox - Resources, Tools and Basic Information for Engineering and Design of Technical Applications,” The Engineering Toolbox, 2018. [Online]. Available: https://www.engineeringtoolbox.com. [Accessed on 25 July 2018].

[55] A. E. Fitzgerald, C. Kingsley and S. Umans, Electric Machinery, McGraw-Hill, 2003.

[56] J. Villate, Electromagnetismo, McGraw-Hill, 1999.

[57] A. H. Slocum, “Power Systems,” MIT - Massachusetts Institute of Technology, 2008.

Annexes

Annex A

In this annex the C++ code for the sensor design to be used in Gmsh and Elmer, is presented, as follows:

```
#include < iostream >
#include < string >
#include < fstream >
#include < vector >
int N = 70;
//struct 1
void  create_base_points(std::string  z_pos,  int  base_number,
std::ofstream * outfile) {
  int point = base_number;
  // create first section
  ( * outfile) << "Point(" << point << ") = { 0, 0, " << z_pos
<< ", el};" << std::endl;
  point = point + 1;
  ( * outfile) << "Point(" << point << ") = { 0, w, " << z_pos
<< ", el};" << std::endl;
  point = point + 1;
  // create wires
  for (int i = 1; i < N + 1; i++) {
    //(*outfile) << i << std::endl;
    ( * outfile) << "Point(" << point << ") = { delta + " << i
- 1 << "*d, 0, " << z_pos << ", el};" << std::endl;
    point = point + 1;
    ( * outfile) << "Point(" << point << ") = { delta + " << i
- 1 << "*d, w, " << z_pos << ", el};" << std::endl;
    point = point + 1;
    ( * outfile) << "Point(" << point << ") = { delta + " << i
- 1 << "*d + d/2, 0, " << z_pos << ", el};" << std::endl;
    point = point + 1;
    ( * outfile) << "Point(" << point << ") = { delta + " << i
- 1 << "*d + d/2, w, " << z_pos << ", el};" << std::endl;
    point = point + 1;
    ( * outfile) << "Point(" << point << ") = { delta + " << i
- 1 << "*d + d/2, w/2, " << z_pos << ", el};" << std::endl;
    point = point + 1;
    ( * outfile) << "Point(" << point << ") = { delta + " << i
- 1 << "*d + d/2 - a, w/2, " << z_pos << ", el};" << std::endl;
    point = point + 1;
    ( * outfile) << "Point(" << point << ") = { delta + " << i
- 1 << "*d + d/2 + a, w/2, " << z_pos << ", el};" << std::endl;
    point = point + 1;
    ( * outfile) << "Point(" << point << ") = { delta + " << i
- 1 << "*d + d/2, w/2 + a, " << z_pos << ", el};" << std::endl;
    point = point + 1;
    ( * outfile) << "Point(" << point << ") = { delta + " << i
- 1 << "*d + d/2, w/2 - a, " << z_pos << ", el};" << std::endl;
    point = point + 1;
  }
```

```
  // create final section
  ( * outfile) << "Point(" << point << ") = { delta + " << N <<
" *d, 0, " << z_pos << ", el};" << std::endl;
  point = point + 1;
  ( * outfile) << "Point(" << point << ") = { delta + " << N <<
" *d, w, " << z_pos << ", el};" << std::endl;
  point = point + 1;
  ( * outfile) << "Point(" << point << ") = { 2*delta + " << N
<< " *d, 0, " << z_pos << ", el};" << std::endl;
  point = point + 1;
  ( * outfile) << "Point(" << point << ") = { 2*delta + " << N
<< " *d, w, " << z_pos << ", el};" << std::endl;
  point = point + 1;
}
void      create_base_lines(std::string      z_pos,      int
base_number_points,  int  base_number_lines,  std::ofstream  *
outfile) {
  int line = base_number_lines;
  int start = base_number_points - 1;
  // create first section
  ( * outfile) << "Line(" << line << ") = {" << start + 1 << ",
" << start + 2 << "};" << std::endl;
  line = line + 1;
  ( * outfile) << "Line(" << line << ") = {" << start + 1 << ",
" << start + 3 << "};" << std::endl;
  line = line + 1;
  ( * outfile) << "Line(" << line << ") = {" << start + 3 << ",
" << start + 4 << "};" << std::endl;
  line = line + 1;
  ( * outfile) << "Line(" << line << ") = {" << start + 2 << ",
" << start + 4 << "};" << std::endl;
  line = line + 1;
  for (int i = 0; i < N; i++) {
    ( * outfile) << "Line(" << line << ") = {" << start + 3 + i
* 9 << ", " << start + 8 + i * 9 << "};" << std::endl;
    line = line + 1;
    ( * outfile) << "Line(" << line << ") = {" << start + 4 + i
* 9 << ", " << start + 8 + i * 9 << "};" << std::endl;
    line = line + 1;
    ( * outfile) << "Line(" << line << ") = {" << start + 6 + i
* 9 << ", " << start + 10 + i * 9 << "};" << std::endl;
    line = line + 1;
    ( * outfile) << "Line(" << line << ") = {" << start + 5 + i
* 9 << ", " << start + 11 + i * 9 << "};" << std::endl;
    line = line + 1;
    ( * outfile) << "Line(" << line << ") = {" << start + 9 + i
* 9 << ", " << start + 12 + i * 9 << "};" << std::endl;
    line = line + 1;
    ( * outfile) << "Line(" << line << ") = {" << start + 9 + i
* 9 << ", " << start + 13 + i * 9 << "};" << std::endl;
    line = line + 1;
    ( * outfile) << "Line(" << line << ") = {" << start + 3 + i
* 9 << ", " << start + 5 + i * 9 << "};" << std::endl;
    line = line + 1;
    ( * outfile) << "Line(" << line << ") = {" << start + 5 + i
```

```
* 9 << ", " << start + 12 + i * 9 << "};" << std::endl;
    line = line + 1;
    ( * outfile) << "Line(" << line << ") = {" << start + 4 + i
* 9 << ", " << start + 6 + i * 9 << "};" << std::endl;
    line = line + 1;
    ( * outfile) << "Line(" << line << ") = {" << start + 6 + i
* 9 << ", " << start + 13 + i * 9 << "};" << std::endl;
    line = line + 1;
    ( * outfile) << "Circle(" << line << ") = {" << start + 8 +
i * 9 << ", " << start + 7 + i * 9 << ", " << start + 10 + i *
9 << "};" << std::endl;
    line = line + 1;
    ( * outfile) << "Circle(" << line << ") = {" << start + 10
+ i * 9 << ", " << start + 7 + i * 9 << ", " << start + 9 + i *
9 << "};" << std::endl;
    line = line + 1;
    ( * outfile) << "Circle(" << line << ") = {" << start + 9 +
i * 9 << ", " << start + 7 + i * 9 << ", " << start + 11 + i *
9 << "};" << std::endl;
    line = line + 1;
    ( * outfile) << "Circle(" << line << ") = {" << start + 11
+ i * 9 << ", " << start + 7 + i * 9 << ", " << start + 8 + i *
9 << "};" << std::endl;
    line = line + 1;
  }
  // create final section
  int point = 3 + 9 * N;
  ( * outfile) << "Line(" << line << ") = {" << start + point
<< ", " << start + point + 1 << "};" << std::endl;
  line = line + 1;
  ( * outfile) << "Line(" << line << ") = {" << start + point
<< ", " << start + point + 2 << "};" << std::endl;
  line = line + 1;
  ( * outfile) << "Line(" << line << ") = {" << start + point +
2 << ", " << start + point + 3 << "};" << std::endl;
  line = line + 1;
  ( * outfile) << "Line(" << line << ") = {" << start + point +
1 << ", " << start + point + 3 << "};" << std::endl;
  line = line + 1;
}
void   create_parameter(std::string   name,   float   value,
std::ofstream * outfile) {
  ( * outfile) << name << " = DefineNumber[" << value << ", Name
\"Parameters/" << name << "\" ];" << std::endl;
}
void connect(int lower_point, int upper_point, int base_line,
std::ofstream * outfile) {
  int line = base_line;
  // first section
  for (int i = 0; i < 4; i++) {
    ( * outfile) << "Line(" << line << ") = {" << lower_point +
i << ", " << upper_point + i << "};" << std::endl;
    line = line + 1;
  }
  // wires section
```

```
  for (int i = 0; i < N; i++) {
    ( * outfile) << "Line(" << line << ") = {" << (lower_point
- 1) + 8 + i * 9 << ", " << (upper_point - 1) + 8 + i * 9 <<
"};" << std::endl;
    line = line + 1;
    ( * outfile) << "Line(" << line << ") = {" << (lower_point
- 1) + 10 + i * 9 << ", " << (upper_point - 1) + 10 + i * 9 <<
"};" << std::endl;
    line = line + 1;
    ( * outfile) << "Line(" << line << ") = {" << (lower_point
- 1) + 11 + i * 9 << ", " << (upper_point - 1) + 11 + i * 9 <<
"};" << std::endl;
    line = line + 1;
    ( * outfile) << "Line(" << line << ") = {" << (lower_point
- 1) + 9 + i * 9 << ", " << (upper_point - 1) + 9 + i * 9 <<
"};" << std::endl;
    line = line + 1;
    ( * outfile) << "Line(" << line << ") = {" << (lower_point
- 1) + 5 + i * 9 << ", " << (upper_point - 1) + 5 + i * 9 <<
"};" << std::endl;
    line = line + 1;
    ( * outfile) << "Line(" << line << ") = {" << (lower_point
- 1) + 6 + i * 9 << ", " << (upper_point - 1) + 6 + i * 9 <<
"};" << std::endl;
    line = line + 1;
    ( * outfile) << "Line(" << line << ") = {" << (lower_point
- 1) + 12 + i * 9 << ", " << (upper_point - 1) + 12 + i * 9 <<
"};" << std::endl;
    line = line + 1;
    ( * outfile) << "Line(" << line << ") = {" << (lower_point
- 1) + 13 + i * 9 << ", " << (upper_point - 1) + 13 + i * 9 <<
"};" << std::endl;
    line = line + 1;
  }
  // final section
  ( * outfile) << "Line(" << line << ") = {" << (lower_point -
1) + 5 + N * 9 << ", " << (upper_point - 1) + 5 + N * 9 << "};"
<< std::endl;
  line = line + 1;
  ( * outfile) << "Line(" << line << ") = {" << (lower_point -
1) + 6 + N * 9 << ", " << (upper_point - 1) + 6 + N * 9 << "};"
<< std::endl;
  line = line + 1;
}
void   create_base_surface(int   base_line,   int   loop_index,
std::ofstream * outfile) {
  int index = loop_index;
  // first section
  int start = base_line - 1;
  ( * outfile) << "Line Loop(" << index << ") = {" << start + 1
<< ", " << start + 4 << ", -" << start + 3 << ", -" << start +
2 << "};" << std::endl;
  ( * outfile) << "Plane Surface(" << index << ") = {" << index
<< "};" << std::endl;
  index = index + 1;
```

```
  ( * outfile) << "Line Loop(" << index << ") = {" << start + 3
<< ", " << start + 6 << ", -" << start + 5 << "};" << std::endl;
  ( * outfile) << "Plane Surface(" << index << ") = {" << index
<< "};" << std::endl;
  index = index + 1;
  // wire section
  for (int i = 0; i < N; i++) {
    ( * outfile) << "Line Loop(" << index << ") = {" << start +
15 + 14 * i << ", " << start + 16 + 14 * i << ", " << start +
17 + 14 * i << ", " << start + 18 + 14 * i << "};" << std::endl;
    ( * outfile) << "Plane Surface(" << index << ") = {" <<
index << "};" << std::endl;
    index = index + 1;
    ( * outfile) << "Line Loop(" << index << ") = {" << start +
5 + 14 * i << ", -" << start + 18 + 14 * i << ", -" << start +
8 + 14 * i << ", -" << start + 11 + 14 * i << "};" << std::endl;
    ( * outfile) << "Surface(" << index << ") = {" << index <<
"};" << std::endl;
    index = index + 1;
    ( * outfile) << "Line Loop(" << index << ") = {" << start +
8 + 14 * i << ", -" << start + 17 + 14 * i << ", " << start + 9
+ 14 * i << ", -" << start + 12 + 14 * i << "};" << std::endl;
    ( * outfile) << "Surface(" << index << ") = {" << index <<
"};" << std::endl;
    index = index + 1;
    ( * outfile) << "Line Loop(" << index << ") = {" << start +
16 + 14 * i << ", " << start + 10 + 14 * i << ", -" << start +
14 + 14 * i << ", " << start + 7 + 14 * i << "};" << std::endl;
    ( * outfile) << "Surface(" << index << ") = {" << index <<
"};" << std::endl;
    index = index + 1;
    ( * outfile) << "Line Loop(" << index << ") = {" << start +
15 + 14 * i << ", -" << start + 7 + 14 * i << ", -" << start +
13 + 14 * i << ", " << start + 6 + 14 * i << "};" << std::endl;
    ( * outfile) << "Surface(" << index << ") = {" << index <<
"};" << std::endl;
    index = index + 1;
  }
  for (int i = 0; i < N - 1; i++) {
    ( * outfile) << "Line Loop(" << index << ") = {" << start +
10 + 14 * i << ", " << start + 20 + 14 * i << ", -" << start +
19 + 14 * i << ", -" << start + 9 + 14 * i << "};" << std::endl;
    ( * outfile) << "Surface(" << index << ") = {" << index <<
"};" << std::endl;
    index = index + 1;
  }
  // final section
  int base = (base_line - 1) + 4 + 14 * N + 1;
  ( * outfile) << "Line Loop(" << index << ") = {" << base + 2
<< ", -" << base + 3 << ", -" << base << ", " << base + 1 <<
"};" << std::endl;
  ( * outfile) << "Plane Surface(" << index << ") = {" << index
<< "};" << std::endl;
  index = index + 1;
  ( * outfile) << "Line Loop(" << index << ") = {" << base <<
```

```
", -" << base - 9 << ", " << base - 10 << "};" << std::endl;
  ( * outfile) << "Plane Surface(" << index << ") = {" << index
<< "};" << std::endl;
  index = index + 1;
}
void create_lateral_surface(int lower_index, int upper_index,
int lateral_index, int line_index, std::ofstream * outfile) {
  // first section
  int index = line_index;
  lower_index--;
  upper_index--;
  lateral_index--;
  ( * outfile) << "Line Loop(" << index << ") = {" << lower_index
+ 1 << ", " << lateral_index + 2 << ", -" << upper_index + 1 <<
", -" << lateral_index + 1 << "};" << std::endl;
  ( * outfile) << "Plane Surface(" << index << ") = {" << index
<< "};" << std::endl;
  index = index + 1;
  ( * outfile) << "Line Loop(" << index << ") = {" << lower_index
+ 2 << ", " << lateral_index + 3 << ", -" << upper_index + 2 <<
", -" << lateral_index + 1 << "};" << std::endl;
  ( * outfile) << "Plane Surface(" << index << ") = {" << index
<< "};" << std::endl;
  index = index + 1;
  ( * outfile) << "Line Loop(" << index << ") = {" << lower_index
+ 4 << ", " << lateral_index + 4 << ", -" << upper_index + 4 <<
", -" << lateral_index + 2 << "};" << std::endl;
  ( * outfile) << "Plane Surface(" << index << ") = {" << index
<< "};" << std::endl;
  index = index + 1;
  ( * outfile) << "Line Loop(" << index << ") = {" << lower_index
+ 3 << ", " << lateral_index + 4 << ", -" << upper_index + 3 <<
", -" << lateral_index + 3 << "};" << std::endl;
  ( * outfile) << "Plane Surface(" << index << ") = {" << index
<< "};" << std::endl;
  index = index + 1;
  // wire section
  for (int i = 0; i < N; i++) {
    ( * outfile) << "Line Loop(" << index << ") = {" <<
lower_index + 18 + 14 * i << ", " << lateral_index + 5 + 8 * i
<< ", -" << upper_index + 18 + 14 * i << ", -" << lateral_index
+ 7 + 8 * i << "};" << std::endl;
    ( * outfile) << "Surface(" << index << ") = {" << index <<
"};" << std::endl;
    index = index + 1;
    ( * outfile) << "Line Loop(" << index << ") = {" <<
lower_index + 17 + 14 * i << ", " << lateral_index + 7 + 8 * i
<< ", -" << upper_index + 17 + 14 * i << ", -" << lateral_index
+ 8 + 8 * i << "};" << std::endl;
    ( * outfile) << "Surface(" << index << ") = {" << index <<
"};" << std::endl;
    index = index + 1;
    ( * outfile) << "Line Loop(" << index << ") = {" <<
lower_index + 15 + 14 * i << ", " << lateral_index + 6 + 8 * i
<< ", -" << upper_index + 15 + 14 * i << ", -" << lateral_index
```

```
+ 5 + 8 * i << "};" << std::endl;
    ( * outfile) << "Surface(" << index << ") = {" << index <<
"};" << std::endl;
    index = index + 1;
    ( * outfile) << "Line Loop(" << index << ") = {" <<
lower_index + 16 + 14 * i << ", " << lateral_index + 8 + 8 * i
<< ", -" << upper_index + 16 + 14 * i << ", -" << lateral_index
+ 6 + 8 * i << "};" << std::endl;
    ( * outfile) << "Surface(" << index << ") = {" << index <<
"};" << std::endl;
    index = index + 1;
    ( * outfile) << "Line Loop(" << index << ") = {" <<
lower_index + 5 + 14 * i << ", " << lateral_index + 5 + 8 * i
<< ", -" << upper_index + 5 + 14 * i << ", -" << lateral_index
+ 3 + 8 * i << "};" << std::endl;
    ( * outfile) << "Plane Surface(" << index << ") = {" <<
index << "};" << std::endl;
    index = index + 1;
    ( * outfile) << "Line Loop(" << index << ") = {" <<
lower_index + 6 + 14 * i << ", " << lateral_index + 5 + 8 * i
<< ", -" << upper_index + 6 + 14 * i << ", -" << lateral_index
+ 4 + 8 * i << "};" << std::endl;
    ( * outfile) << "Plane Surface(" << index << ") = {" <<
index << "};" << std::endl;
    index = index + 1;
    ( * outfile) << "Line Loop(" << index << ") = {" <<
lower_index + 11 + 14 * i << ", " << lateral_index + 9 + 8 * i
<< ", -" << upper_index + 11 + 14 * i << ", -" << lateral_index
+ 3 + 8 * i << "};" << std::endl;
    ( * outfile) << "Plane Surface(" << index << ") = {" <<
index << "};" << std::endl;
    index = index + 1;
    ( * outfile) << "Line Loop(" << index << ") = {" <<
lower_index + 8 + 14 * i << ", " << lateral_index + 7 + 8 * i
<< ", -" << upper_index + 8 + 14 * i << ", -" << lateral_index
+ 9 + 8 * i << "};" << std::endl;
    ( * outfile) << "Plane Surface(" << index << ") = {" <<
index << "};" << std::endl;
    index = index + 1;
    ( * outfile) << "Line Loop(" << index << ") = {" <<
lower_index + 12 + 14 * i << ", " << lateral_index + 11 + 8 * i
<< ", -" << upper_index + 12 + 14 * i << ", -" << lateral_index
+ 9 + 8 * i << "};" << std::endl;
    ( * outfile) << "Plane Surface(" << index << ") = {" <<
index << "};" << std::endl;
    index = index + 1;
    ( * outfile) << "Line Loop(" << index << ") = {" <<
lower_index + 9 + 14 * i << ", " << lateral_index + 11 + 8 * i
<< ", -" << upper_index + 9 + 14 * i << ", -" << lateral_index
+ 8 + 8 * i << "};" << std::endl;
    ( * outfile) << "Plane Surface(" << index << ") = {" <<
index << "};" << std::endl;
    index = index + 1;
    ( * outfile) << "Line Loop(" << index << ") = {" <<
lower_index + 10 + 14 * i << ", " << lateral_index + 12 + 8 * i
```

```
<< ", -" << upper_index + 10 + 14 * i << ", -" << lateral_index
+ 8 + 8 * i << "};" << std::endl;
    ( * outfile) << "Plane Surface(" << index << ") = {" <<
index << "};" << std::endl;
    index = index + 1;
    ( * outfile) << "Line Loop(" << index << ") = {" <<
lower_index + 7 + 14 * i << ", " << lateral_index + 6 + 8 * i
<< ", -" << upper_index + 7 + 14 * i << ", -" << lateral_index
+ 10 + 8 * i << "};" << std::endl;
    ( * outfile) << "Plane Surface(" << index << ") = {" <<
index << "};" << std::endl;
    index = index + 1;
    ( * outfile) << "Line Loop(" << index << ") = {" <<
lower_index + 14 + 14 * i << ", " << lateral_index + 12 + 8 * i
<< ", -" << upper_index + 14 + 14 * i << ", -" << lateral_index
+ 10 + 8 * i << "};" << std::endl;
    ( * outfile) << "Plane Surface(" << index << ") = {" <<
index << "};" << std::endl;
    index = index + 1;
    ( * outfile) << "Line Loop(" << index << ") = {" <<
lower_index + 13 + 14 * i << ", " << lateral_index + 10 + 8 * i
<< ", -" << upper_index + 13 + 14 * i << ", -" << lateral_index
+ 4 + 8 * i << "};" << std::endl;
    ( * outfile) << "Plane Surface(" << index << ") = {" <<
index << "};" << std::endl;
    index = index + 1;
  }
  // last section
  ( * outfile) << "Line Loop(" << index << ") = {" << lower_index
+ 5 + 14 * N << ", " << lateral_index + 4 + 8 * N << ", -" <<
upper_index + 5 + 14 * N << ", -" << lateral_index + 3 + 8 * N
<< "};" << std::endl;
  ( * outfile) << "Plane Surface(" << index << ") = {" << index
<< "};" << std::endl;
  index = index + 1;
  ( * outfile) << "Line Loop(" << index << ") = {" << lower_index
+ 5 + 14 * N + 2 << ", " << lateral_index + 4 + 8 * N + 2 << ",
-" << upper_index + 5 + 14 * N + 2 << ", -" << lateral_index +
3 + 8 * N + 2 << "};" << std::endl;
  ( * outfile) << "Plane Surface(" << index << ") = {" << index
<< "};" << std::endl;
  index = index + 1;
  ( * outfile) << "Line Loop(" << index << ") = {" << lower_index
+ 5 + 14 * N + 1 << ", " << lateral_index + 4 + 8 * N + 1 << ",
-" << upper_index + 5 + 14 * N + 1 << ", -" << lateral_index +
3 + 8 * N << "};" << std::endl;
  ( * outfile) << "Plane Surface(" << index << ") = {" << index
<< "};" << std::endl;
  index = index + 1;
  ( * outfile) << "Line Loop(" << index << ") = {" << lower_index
+ 5 + 14 * N + 3 << ", " << lateral_index + 4 + 8 * N + 2 << ",
-" << upper_index + 5 + 14 * N + 3 << ", -" << lateral_index +
3 + 8 * N + 1 << "};" << std::endl;
  ( * outfile) << "Plane Surface(" << index << ") = {" << index
<< "};" << std::endl;
```

```
  index = index + 1;
}
void  create_base_volumes(int  base_surface,  int  loop_index,
std::ofstream * outfile) {
  int index = loop_index;
  // first section
  int start = base_surface - 1;
  ( * outfile) << "Surface Loop(" << index << ") = {" << start
+ 1 << ", " << start + 1001 << ", " << start + 4001 << ", " <<
start + 4002 << ", " << start + 4003 << ", " << start + 4004 <<
"};" << std::endl;
  ( * outfile) << "Volume(" << index << ") = {" << index << "};"
<< std::endl;
  index = index + 1;
  // create triangle of first section
  ( * outfile) << "Surface Loop(" << index << ") = {" << start
+ 2 << ", " << start + 1002 << ", " << start + 4004 << ", " <<
start + 4009 << ", " << start + 4010 << "};" << std::endl;
  ( * outfile) << "Volume(" << index << ") = {" << index << "};"
<< std::endl;
  index = index + 1;
  // wire section
  for (int i = 0; i < N; i++) {
    ( * outfile) << "Surface Loop(" << index << ") = {" << start
+ 3 + 5 * i << ", " << start + 1003 + 5 * i << ", " << start +
4005 + 14 * i << ", " << start + 4006 + 14 * i << ", " << start
+ 4007 + 14 * i << ", " << start + 4008 + 14 * i << "};" <<
std::endl;
    ( * outfile) << "Volume(" << index << ") = {" << index <<
"};" << std::endl;
    index = index + 1;
    // create volumes around each wire
    ( * outfile) << "Surface Loop(" << index << ") = {" << start
+ 4 + 5 * i << ", " << start + 1004 + 5 * i << ", " << start +
4005 + 14 * i << ", " << start + 4009 + 14 * i << ", " << start
+ 4011 + 14 * i << ", " << start + 4012 + 14 * i << "};" <<
std::endl;
    ( * outfile) << "Volume(" << index << ") = {" << index <<
"};" << std::endl;
    index = index + 1;
    ( * outfile) << "Surface Loop(" << index << ") = {" << start
+ 5 + 5 * i << ", " << start + 1005 + 5 * i << ", " << start +
4006 + 14 * i << ", " << start + 4012 + 14 * i << ", " << start
+ 4013 + 14 * i << ", " << start + 4014 + 14 * i << "};" <<
std::endl;
    ( * outfile) << "Volume(" << index << ") = {" << index <<
"};" << std::endl;
    index = index + 1;
    ( * outfile) << "Surface Loop(" << index << ") = {" << start
+ 6 + 5 * i << ", " << start + 1006 + 5 * i << ", " << start +
4008 + 14 * i << ", " << start + 4015 + 14 * i << ", " << start
+ 4016 + 14 * i << ", " << start + 4017 + 14 * i << "};" <<
std::endl;
    ( * outfile) << "Volume(" << index << ") = {" << index <<
"};" << std::endl;
```

```
    index = index + 1;
    ( * outfile) << "Surface Loop(" << index << ") = {" << start
+ 7 + 5 * i << ", " << start + 1007 + 5 * i << ", " << start +
4007 + 14 * i << ", " << start + 4010 + 14 * i << ", " << start
+ 4016 + 14 * i << ", " << start + 4018 + 14 * i << "};" <<
std::endl;
    ( * outfile) << "Volume(" << index << ") = {" << index <<
"};" << std::endl;
    index = index + 1;
  }
  // create lozenge in middle of wires
  if (N > 1) {
    int base_surface_lozenge = 8 + 5 * (N - 1);
    for (int i = 0; i < N - 1; i++) {
      ( * outfile) << "Surface Loop(" << index << ") = {" <<
start + base_surface_lozenge + i << ", " << start + 1000 +
base_surface_lozenge + i << ", " << start + 4000 + 14 * (i + 1)
<< ", " << start + 4001 + 14 * (i + 1) << ", " << start + 4009
+ 14 * (i + 1) << ", " << start + 4010 + 14 * (i + 1) << "};"
<< std::endl;
      ( * outfile) << "Volume(" << index << ") = {" << index <<
"};" << std::endl;
      index = index + 1;
    }
  }
  // final section
  int base = (base_surface - 1) + 6 * N + 2;
  ( * outfile) << "Surface Loop(" << index << ") = {" << base
<< ", " << base + 1000 << ", " << base + 4011 + 8 * (N - 1) <<
", " << base + 4012 + 8 * (N - 1) << ", " << base + 4013 + 8 *
(N - 1) << ", " << base + 4014 + 8 * (N - 1) << "};" << std::endl;
  ( * outfile) << "Volume(" << index << ") = {" << index << "};"
<< std::endl;
  index = index + 1;
  // create triangle of final section
  base = base + 1;
  ( * outfile) << "Surface Loop(" << index << ") = {" << base
<< ", " << base + 1000 << ", " << base + 4005 + 8 * (N - 1) <<
", " << base + 4006 + 8 * (N - 1) << ", " << base + 4010 + 8 *
(N - 1) << "};" << std::endl;
  ( * outfile) << "Volume(" << index << ") = {" << index << "};"
<< std::endl;
  index = index + 1;
}
void create_base_volumes_middle_section(int base_surface, int
loop_index, std::ofstream * outfile) {
  int index = loop_index;
  int start = base_surface - 1;
  //first section
  ( * outfile) << "Surface Loop(" << index << ") = {" << start
+ 1 << ", " << start + 1001 << ", " << start + 4001 << ", " <<
start + 4002 << ", " << start + 4003 << ", " << start + 4004 <<
"};" << std::endl;
  ( * outfile) << "Volume(" << index << ") = {" << index << "};"
<< std::endl;
```

```
  index = index + 1;
  // wire section
  for (int i = 0; i < N; i++) {
    ( * outfile) << "Surface Loop(" << index << ") = {" << start
+ 3 + 5 * i << ", " << start + 1003 + 5 * i << ", " << start +
4005 + 14 * i << ", " << start + 4006 + 14 * i << ", " << start
+ 4007 + 14 * i << ", " << start + 4008 + 14 * i << "};" <<
std::endl;
    ( * outfile) << "Volume(" << index << ") = {" << index <<
"};" << std::endl;
    index = index + 1;
  }
  // final section
  int base = (base_surface - 1) + 6 * N + 2;
  ( * outfile) << "Surface Loop(" << index << ") = {" << base
<< ", " << base + 1000 << ", " << base + 4011 + 8 * (N - 1) <<
", " << base + 4012 + 8 * (N - 1) << ", " << base + 4013 + 8 *
(N - 1) << ", " << base + 4014 + 8 * (N - 1) << "};" << std::endl;
  ( * outfile) << "Volume(" << index << ") = {" << index << "};"
<< std::endl;
  index = index + 1;
}
//struct 2
void    create_base_points_y_shift(std::string    z_pos,    int
base_number, std::ofstream * outfile) {
  int point = base_number;
  // create first section
  ( * outfile) << "Point(" << point << ") = { 0, w+h, " << z_pos
<< ", el};" << std::endl;
  point = point + 1;
  ( * outfile) << "Point(" << point << ") = { 0, (2*w+h), " <<
z_pos << ", el};" << std::endl;
  point = point + 1;
  // create wires
  for (int i = 1; i < N + 1; i++) {
    //(*outfile) << i << std::endl;
    ( * outfile) << "Point(" << point << ") = { delta + " << i
- 1 << "*d, w+h, " << z_pos << ", el};" << std::endl;
    point = point + 1;
    ( * outfile) << "Point(" << point << ") = { delta + " << i
- 1 << "*d, (2*w+h), " << z_pos << ", el};" << std::endl;
    point = point + 1;
    ( * outfile) << "Point(" << point << ") = { delta + " << i
- 1 << "*d + d/2, w+h, " << z_pos << ", el};" << std::endl;
    point = point + 1;
    ( * outfile) << "Point(" << point << ") = { delta + " << i
- 1 << "*d + d/2, (2*w+h), " << z_pos << ", el};" << std::endl;
    point = point + 1;
    ( * outfile) << "Point(" << point << ") = { delta + " << i
- 1 << "*d + d/2,  w+h+(w/2), " << z_pos << ", el};" << std::endl;
    point = point + 1;
    ( * outfile) << "Point(" << point << ") = { delta + " << i
- 1 << "*d + d/2 - a, w+h+(w/2), " << z_pos << ", el};" <<
std::endl;
    point = point + 1;
```

```
    ( * outfile) << "Point(" << point << ") = { delta + " << i
- 1 << "*d + d/2 + a, w+h+(w/2), " << z_pos << ", el};" <<
std::endl;
    point = point + 1;
    ( * outfile) << "Point(" << point << ") = { delta + " << i
- 1 << "*d + d/2, w+h+(w/2) + a, " << z_pos << ", el};" <<
std::endl;
    point = point + 1;
    ( * outfile) << "Point(" << point << ") = { delta + " << i
- 1 << "*d + d/2, w+h+(w/2) - a, " << z_pos << ", el};" <<
std::endl;
    point = point + 1;
  }
  // create final section
  ( * outfile) << "Point(" << point << ") = { delta + " << N <<
" *d, w+h, " << z_pos << ", el};" << std::endl;
  point = point + 1;
  ( * outfile) << "Point(" << point << ") = { delta + " << N <<
" *d, ((w+h)+w), " << z_pos << ", el};" << std::endl;
  point = point + 1;
  ( * outfile) << "Point(" << point << ") = { 2*delta + " << N
<< " *d, w+h, " << z_pos << ", el};" << std::endl;
  point = point + 1;
  ( * outfile) << "Point(" << point << ") = { 2*delta + " << N
<< " *d, ((w+h)+w), " << z_pos << ", el};" << std::endl;
  point = point + 1;
}
void    create_base_lines_y_shift(std::string    z_pos,    int
base_number_points,  int  base_number_lines,  std::ofstream  *
outfile) {
  int line = base_number_lines;
  int start = base_number_points - 1;
  // create first section
  ( * outfile) << "Line(" << line << ") = {" << start + 1 << ",
" << start + 2 << "};" << std::endl;
  line = line + 1;
  ( * outfile) << "Line(" << line << ") = {" << start + 1 << ",
" << start + 3 << "};" << std::endl;
  line = line + 1;
  ( * outfile) << "Line(" << line << ") = {" << start + 3 << ",
" << start + 4 << "};" << std::endl;
  line = line + 1;
  ( * outfile) << "Line(" << line << ") = {" << start + 2 << ",
" << start + 4 << "};" << std::endl;
  line = line + 1;
  for (int i = 0; i < N; i++) {
    ( * outfile) << "Line(" << line << ") = {" << start + 3 + i
* 9 << ", " << start + 8 + i * 9 << "};" << std::endl;
    line = line + 1;
    ( * outfile) << "Line(" << line << ") = {" << start + 4 + i
* 9 << ", " << start + 8 + i * 9 << "};" << std::endl;
    line = line + 1;
    ( * outfile) << "Line(" << line << ") = {" << start + 6 + i
* 9 << ", " << start + 10 + i * 9 << "};" << std::endl;
    line = line + 1;
```

```
    ( * outfile) << "Line(" << line << ") = {" << start + 5 + i
* 9 << ", " << start + 11 + i * 9 << "};" << std::endl;
    line = line + 1;
    ( * outfile) << "Line(" << line << ") = {" << start + 9 + i
* 9 << ", " << start + 12 + i * 9 << "};" << std::endl;
    line = line + 1;
    ( * outfile) << "Line(" << line << ") = {" << start + 9 + i
* 9 << ", " << start + 13 + i * 9 << "};" << std::endl;
    line = line + 1;
    ( * outfile) << "Line(" << line << ") = {" << start + 3 + i
* 9 << ", " << start + 5 + i * 9 << "};" << std::endl;
    line = line + 1;
    ( * outfile) << "Line(" << line << ") = {" << start + 5 + i
* 9 << ", " << start + 12 + i * 9 << "};" << std::endl;
    line = line + 1;
    ( * outfile) << "Line(" << line << ") = {" << start + 4 + i
* 9 << ", " << start + 6 + i * 9 << "};" << std::endl;
    line = line + 1;
    ( * outfile) << "Line(" << line << ") = {" << start + 6 + i
* 9 << ", " << start + 13 + i * 9 << "};" << std::endl;
    line = line + 1;
    ( * outfile) << "Circle(" << line << ") = {" << start + 8 +
i * 9 << ", " << start + 7 + i * 9 << ", " << start + 10 + i *
9 << "};" << std::endl;
    line = line + 1;
    ( * outfile) << "Circle(" << line << ") = {" << start + 10
+ i * 9 << ", " << start + 7 + i * 9 << ", " << start + 9 + i *
9 << "};" << std::endl;
    line = line + 1;
    ( * outfile) << "Circle(" << line << ") = {" << start + 9 +
i * 9 << ", " << start + 7 + i * 9 << ", " << start + 11 + i *
9 << "};" << std::endl;
    line = line + 1;
    ( * outfile) << "Circle(" << line << ") = {" << start + 11
+ i * 9 << ", " << start + 7 + i * 9 << ", " << start + 8 + i *
9 << "};" << std::endl;
    line = line + 1;
  }
  // create final section
  int point = 3 + 9 * N;
  ( * outfile) << "Line(" << line << ") = {" << start + point
<< ", " << start + point + 1 << "};" << std::endl;
  line = line + 1;
  ( * outfile) << "Line(" << line << ") = {" << start + point
<< ", " << start + point + 2 << "};" << std::endl;
  line = line + 1;
  ( * outfile) << "Line(" << line << ") = {" << start + point +
2 << ", " << start + point + 3 << "};" << std::endl;
  line = line + 1;
  ( * outfile) << "Line(" << line << ") = {" << start + point +
1 << ", " << start + point + 3 << "};" << std::endl;
  line = line + 1;
}
void connect_y_shift(int lower_point, int upper_point, int
base_line, std::ofstream * outfile) {
```

```
  int line = base_line;
  // first section
  for (int i = 0; i < 4; i++) {
    ( * outfile) << "Line(" << line << ") = {" << lower_point +
i << ", " << upper_point + i << "};" << std::endl;
    line = line + 1;
  }
  // wires section
  for (int i = 0; i < N; i++) {
    ( * outfile) << "Line(" << line << ") = {" << (lower_point
- 1) + 8 + i * 9 << ", " << (upper_point - 1) + 8 + i * 9 <<
"};" << std::endl;
    line = line + 1;
    ( * outfile) << "Line(" << line << ") = {" << (lower_point
- 1) + 10 + i * 9 << ", " << (upper_point - 1) + 10 + i * 9 <<
"};" << std::endl;
    line = line + 1;
    ( * outfile) << "Line(" << line << ") = {" << (lower_point
- 1) + 11 + i * 9 << ", " << (upper_point - 1) + 11 + i * 9 <<
"};" << std::endl;
    line = line + 1;
    ( * outfile) << "Line(" << line << ") = {" << (lower_point
- 1) + 9 + i * 9 << ", " << (upper_point - 1) + 9 + i * 9 <<
"};" << std::endl;
    line = line + 1;
    ( * outfile) << "Line(" << line << ") = {" << (lower_point
- 1) + 5 + i * 9 << ", " << (upper_point - 1) + 5 + i * 9 <<
"};" << std::endl;
    line = line + 1;
    ( * outfile) << "Line(" << line << ") = {" << (lower_point
- 1) + 6 + i * 9 << ", " << (upper_point - 1) + 6 + i * 9 <<
"};" << std::endl;
    line = line + 1;
    ( * outfile) << "Line(" << line << ") = {" << (lower_point
- 1) + 12 + i * 9 << ", " << (upper_point - 1) + 12 + i * 9 <<
"};" << std::endl;
    line = line + 1;
    ( * outfile) << "Line(" << line << ") = {" << (lower_point
- 1) + 13 + i * 9 << ", " << (upper_point - 1) + 13 + i * 9 <<
"};" << std::endl;
    line = line + 1;
  }
  // final section
  ( * outfile) << "Line(" << line << ") = {" << (lower_point -
1) + 5 + N * 9 << ", " << (upper_point - 1) + 5 + N * 9 << "};"
<< std::endl;
  line = line + 1;
  ( * outfile) << "Line(" << line << ") = {" << (lower_point -
1) + 6 + N * 9 << ", " << (upper_point - 1) + 6 + N * 9 << "};"
<< std::endl;
  line = line + 1;
}
void create_base_surface_y_shift(int base_line, int loop_index,
std::ofstream * outfile) {
  int index = loop_index;
```

```
  // first section
  int start = base_line - 1;
  ( * outfile) << "Line Loop(" << index << ") = {" << start + 1
<< ", " << start + 4 << ", -" << start + 3 << ", -" << start +
2 << "};" << std::endl;
  ( * outfile) << "Plane Surface(" << index << ") = {" << index
<< "};" << std::endl;
  index = index + 1;
  ( * outfile) << "Line Loop(" << index << ") = {" << start + 3
<< ", " << start + 6 << ", -" << start + 5 << "};" << std::endl;
  ( * outfile) << "Plane Surface(" << index << ") = {" << index
<< "};" << std::endl;
  index = index + 1;
  // wire section
  for (int i = 0; i < N; i++) {
    ( * outfile) << "Line Loop(" << index << ") = {" << start +
15 + 14 * i << ", " << start + 16 + 14 * i << ", " << start +
17 + 14 * i << ", " << start + 18 + 14 * i << "};" << std::endl;
    ( * outfile) << "Plane Surface(" << index << ") = {" <<
index << "};" << std::endl;
    index = index + 1;
    ( * outfile) << "Line Loop(" << index << ") = {" << start +
5 + 14 * i << ", -" << start + 18 + 14 * i << ", -" << start +
8 + 14 * i << ", -" << start + 11 + 14 * i << "};" << std::endl;
    ( * outfile) << "Surface(" << index << ") = {" << index <<
"};" << std::endl;
    index = index + 1;
    ( * outfile) << "Line Loop(" << index << ") = {" << start +
8 + 14 * i << ", -" << start + 17 + 14 * i << ", " << start + 9
+ 14 * i << ", -" << start + 12 + 14 * i << "};" << std::endl;
    ( * outfile) << "Surface(" << index << ") = {" << index <<
"};" << std::endl;
    index = index + 1;
    ( * outfile) << "Line Loop(" << index << ") = {" << start +
16 + 14 * i << ", " << start + 10 + 14 * i << ", -" << start +
14 + 14 * i << ", " << start + 7 + 14 * i << "};" << std::endl;
    ( * outfile) << "Surface(" << index << ") = {" << index <<
"};" << std::endl;
    index = index + 1;
    ( * outfile) << "Line Loop(" << index << ") = {" << start +
15 + 14 * i << ", -" << start + 7 + 14 * i << ", -" << start +
13 + 14 * i << ", " << start + 6 + 14 * i << "};" << std::endl;
    ( * outfile) << "Surface(" << index << ") = {" << index <<
"};" << std::endl;
    index = index + 1;
  }
  for (int i = 0; i < N - 1; i++) {
    ( * outfile) << "Line Loop(" << index << ") = {" << start +
10 + 14 * i << ", " << start + 20 + 14 * i << ", -" << start +
19 + 14 * i << ", -" << start + 9 + 14 * i << "};" << std::endl;
    ( * outfile) << "Surface(" << index << ") = {" << index <<
"};" << std::endl;
    index = index + 1;
  }
```

```
  // final section
  int base = (base_line - 1) + 4 + 14 * N + 1;
  ( * outfile) << "Line Loop(" << index << ") = {" << base + 2
<< ", -" << base + 3 << ", -" << base << ", " << base + 1 <<
"};" << std::endl;
  ( * outfile) << "Plane Surface(" << index << ") = {" << index
<< "};" << std::endl;
  index = index + 1;
  ( * outfile) << "Line Loop(" << index << ") = {" << base <<
", -" << base - 9 << ", " << base - 10 << "};" << std::endl;
  ( * outfile) << "Plane Surface(" << index << ") = {" << index
<< "};" << std::endl;
  index = index + 1;
}
void   create_lateral_surface_y_shift(int   lower_index,   int
upper_index, int lateral_index, int line_index, std::ofstream *
outfile) {
  // first section
  int index = line_index;
  lower_index--;
  upper_index--;
  lateral_index--;
  ( * outfile) << "Line Loop(" << index << ") = {" << lower_index
+ 1 << ", " << lateral_index + 2 << ", -" << upper_index + 1 <<
", -" << lateral_index + 1 << "};" << std::endl;
  ( * outfile) << "Plane Surface(" << index << ") = {" << index
<< "};" << std::endl;
  index = index + 1;
  ( * outfile) << "Line Loop(" << index << ") = {" << lower_index
+ 2 << ", " << lateral_index + 3 << ", -" << upper_index + 2 <<
", -" << lateral_index + 1 << "};" << std::endl;
  ( * outfile) << "Plane Surface(" << index << ") = {" << index
<< "};" << std::endl;
  index = index + 1;
  ( * outfile) << "Line Loop(" << index << ") = {" << lower_index
+ 4 << ", " << lateral_index + 4 << ", -" << upper_index + 4 <<
", -" << lateral_index + 2 << "};" << std::endl;
  ( * outfile) << "Plane Surface(" << index << ") = {" << index
<< "};" << std::endl;
  index = index + 1;
  ( * outfile) << "Line Loop(" << index << ") = {" << lower_index
+ 3 << ", " << lateral_index + 4 << ", -" << upper_index + 3 <<
", -" << lateral_index + 3 << "};" << std::endl;
  ( * outfile) << "Plane Surface(" << index << ") = {" << index
<< "};" << std::endl;
  index = index + 1;
  // wire section
  for (int i = 0; i < N; i++) {
    ( * outfile) << "Line Loop(" << index << ") = {" <<
lower_index + 18 + 14 * i << ", " << lateral_index + 5 + 8 * i
<< ", -" << upper_index + 18 + 14 * i << ", -" << lateral_index
+ 7 + 8 * i << "};" << std::endl;
    ( * outfile) << "Surface(" << index << ") = {" << index <<
"};" << std::endl;
    index = index + 1;
```

```
    ( * outfile) << "Line Loop(" << index << ") = {" <<
lower_index + 17 + 14 * i << ", " << lateral_index + 7 + 8 * i
<< ", -" << upper_index + 17 + 14 * i << ", -" << lateral_index
+ 8 + 8 * i << "};" << std::endl;
    ( * outfile) << "Surface(" << index << ") = {" << index <<
"};" << std::endl;
    index = index + 1;
    ( * outfile) << "Line Loop(" << index << ") = {" <<
lower_index + 15 + 14 * i << ", " << lateral_index + 6 + 8 * i
<< ", -" << upper_index + 15 + 14 * i << ", -" << lateral_index
+ 5 + 8 * i << "};" << std::endl;
    ( * outfile) << "Surface(" << index << ") = {" << index <<
"};" << std::endl;
    index = index + 1;
    ( * outfile) << "Line Loop(" << index << ") = {" <<
lower_index + 16 + 14 * i << ", " << lateral_index + 8 + 8 * i
<< ", -" << upper_index + 16 + 14 * i << ", -" << lateral_index
+ 6 + 8 * i << "};" << std::endl;
    ( * outfile) << "Surface(" << index << ") = {" << index <<
"};" << std::endl;
    index = index + 1;
     ( * outfile) << "Line Loop(" << index << ") = {" <<
lower_index + 5 + 14 * i << ", " << lateral_index + 5 + 8 * i
<< ", -" << upper_index + 5 + 14 * i << ", -" << lateral_index
+ 3 + 8 * i << "};" << std::endl;
    ( * outfile) << "Plane Surface(" << index << ") = {" <<
index << "};" << std::endl;
    index = index + 1;
    ( * outfile) << "Line Loop(" << index << ") = {" <<
lower_index + 6 + 14 * i << ", " << lateral_index + 5 + 8 * i
<< ", -" << upper_index + 6 + 14 * i << ", -" << lateral_index
+ 4 + 8 * i << "};" << std::endl;
    ( * outfile) << "Plane Surface(" << index << ") = {" <<
index << "};" << std::endl;
    index = index + 1;
    ( * outfile) << "Line Loop(" << index << ") = {" <<
lower_index + 11 + 14 * i << ", " << lateral_index + 9 + 8 * i
<< ", -" << upper_index + 11 + 14 * i << ", -" << lateral_index
+ 3 + 8 * i << "};" << std::endl;
    ( * outfile) << "Plane Surface(" << index << ") = {" <<
index << "};" << std::endl;
    index = index + 1;
    ( * outfile) << "Line Loop(" << index << ") = {" <<
lower_index + 8 + 14 * i << ", " << lateral_index + 7 + 8 * i
<< ", -" << upper_index + 8 + 14 * i << ", -" << lateral_index
+ 9 + 8 * i << "};" << std::endl;
    ( * outfile) << "Plane Surface(" << index << ") = {" <<
index << "};" << std::endl;
    index = index + 1;
    ( * outfile) << "Line Loop(" << index << ") = {" <<
lower_index + 12 + 14 * i << ", " << lateral_index + 11 + 8 * i
<< ", -" << upper_index + 12 + 14 * i << ", -" << lateral_index
+ 9 + 8 * i << "};" << std::endl;
    ( * outfile) << "Plane Surface(" << index << ") = {" <<
index << "};" << std::endl;
```

```
    index = index + 1;
    ( * outfile) << "Line Loop(" << index << ") = {" <<
lower_index + 9 + 14 * i << ", " << lateral_index + 11 + 8 * i
<< ", -" << upper_index + 9 + 14 * i << ", -" << lateral_index
+ 8 + 8 * i << "};" << std::endl;
    ( * outfile) << "Plane Surface(" << index << ") = {" <<
index << "};" << std::endl;
    index = index + 1;
    ( * outfile) << "Line Loop(" << index << ") = {" <<
lower_index + 10 + 14 * i << ", " << lateral_index + 12 + 8 * i
<< ", -" << upper_index + 10 + 14 * i << ", -" << lateral_index
+ 8 + 8 * i << "};" << std::endl;
    ( * outfile) << "Plane Surface(" << index << ") = {" <<
index << "};" << std::endl;
    index = index + 1;
    ( * outfile) << "Line Loop(" << index << ") = {" <<
lower_index + 7 + 14 * i << ", " << lateral_index + 6 + 8 * i
<< ", -" << upper_index + 7 + 14 * i << ", -" << lateral_index
+ 10 + 8 * i << "};" << std::endl;
    ( * outfile) << "Plane Surface(" << index << ") = {" <<
index << "};" << std::endl;
    index = index + 1;
    ( * outfile) << "Line Loop(" << index << ") = {" <<
lower_index + 14 + 14 * i << ", " << lateral_index + 12 + 8 * i
<< ", -" << upper_index + 14 + 14 * i << ", -" << lateral_index
+ 10 + 8 * i << "};" << std::endl;
    ( * outfile) << "Plane Surface(" << index << ") = {" <<
index << "};" << std::endl;
    index = index + 1;
    ( * outfile) << "Line Loop(" << index << ") = {" <<
lower_index + 13 + 14 * i << ", " << lateral_index + 10 + 8 * i
<< ", -" << upper_index + 13 + 14 * i << ", -" << lateral_index
+ 4 + 8 * i << "};" << std::endl;
    ( * outfile) << "Plane Surface(" << index << ") = {" <<
index << "};" << std::endl;
    index = index + 1;
  }
  // last section
  ( * outfile) << "Line Loop(" << index << ") = {" << lower_index
+ 5 + 14 * N << ", " << lateral_index + 4 + 8 * N << ", -" <<
upper_index + 5 + 14 * N << ", -" << lateral_index + 3 + 8 * N
<< "};" << std::endl;
  ( * outfile) << "Plane Surface(" << index << ") = {" << index
<< "};" << std::endl;
  index = index + 1;
  ( * outfile) << "Line Loop(" << index << ") = {" << lower_index
+ 5 + 14 * N + 2 << ", " << lateral_index + 4 + 8 * N + 2 << ",
-" << upper_index + 5 + 14 * N + 2 << ", -" << lateral_index +
3 + 8 * N + 2 << "};" << std::endl;
  ( * outfile) << "Plane Surface(" << index << ") = {" << index
<< "};" << std::endl;
  index = index + 1;
  ( * outfile) << "Line Loop(" << index << ") = {" << lower_index
+ 5 + 14 * N + 1 << ", " << lateral_index + 4 + 8 * N + 1 << ",
-" << upper_index + 5 + 14 * N + 1 << ", -" << lateral_index +
```

```
3 + 8 * N << "};" << std::endl;
  ( * outfile) << "Plane Surface(" << index << ") = {" << index
<< "};" << std::endl;
  index = index + 1;
  ( * outfile) << "Line Loop(" << index << ") = {" << lower_index
+ 5 + 14 * N + 3 << ", " << lateral_index + 4 + 8 * N + 2 << ",
-" << upper_index + 5 + 14 * N + 3 << ", -" << lateral_index +
3 + 8 * N + 1 << "};" << std::endl;
  ( * outfile) << "Plane Surface(" << index << ") = {" << index
<< "};" << std::endl;
  index = index + 1;
}
void    create_base_volumes_y_shift(int    base_surface,    int
loop_index, std::ofstream * outfile) {
  int index = loop_index;
  // first section
  int start = base_surface - 1;
  ( * outfile) << "Surface Loop(" << index << ") = {" << start
<< ", " << start + 1001 << ", " << start + 4001 << ", " << start
+ 4002 << ", " << start + 4003 << ", " << start + 4004 << "};"
<< std::endl;
  ( * outfile) << "Volume(" << index << ") = {" << index << "};"
<< std::endl;
  index = index + 1;
  // create triangle of first section
  ( * outfile) << "Surface Loop(" << index << ") = {" << start
+ 1 << ", " << start + 1002 << ", " << start + 4004 << ", " <<
start + 4009 << ", " << start + 4010 << "};" << std::endl;
  ( * outfile) << "Volume(" << index << ") = {" << index << "};"
<< std::endl;
  index = index + 1;
  // wire section
  for (int i = 0; i < N; i++) {
    ( * outfile) << "Surface Loop(" << index << ") = {" << start
+ 2 + 5 * i << ", " << start + 1003 + 5 * i << ", " << start +
4005 + 14 * i << ", " << start + 4006 + 14 * i << ", " << start
+ 4007 + 14 * i << ", " << start + 4008 + 14 * i << "};" <<
std::endl;
    ( * outfile) << "Volume(" << index << ") = {" << index <<
"};" << std::endl;
    index = index + 1;
    // create volumes around each wire
    ( * outfile) << "Surface Loop(" << index << ") = {" << start
+ 3 + 5 * i << ", " << start + 1004 + 5 * i << ", " << start +
4005 + 14 * i << ", " << start + 4009 + 14 * i << ", " << start
+ 4011 + 14 * i << ", " << start + 4012 + 14 * i << "};" <<
std::endl;
    ( * outfile) << "Volume(" << index << ") = {" << index <<
"};" << std::endl;
    index = index + 1;
    ( * outfile) << "Surface Loop(" << index << ") = {" << start
+ 4 + 5 * i << ", " << start + 1005 + 5 * i << ", " << start +
4006 + 14 * i << ", " << start + 4012 + 14 * i << ", " << start
+ 4013 + 14 * i << ", " << start + 4014 + 14 * i << "};" <<
std::endl;
```

```
    ( * outfile) << "Volume(" << index << ") = {" << index <<
"};" << std::endl;
    index = index + 1;
    ( * outfile) << "Surface Loop(" << index << ") = {" << start
+ 5 + 5 * i << ", " << start + 1006 + 5 * i << ", " << start +
4008 + 14 * i << ", " << start + 4015 + 14 * i << ", " << start
+ 4016 + 14 * i << ", " << start + 4017 + 14 * i << "};" <<
std::endl;
    ( * outfile) << "Volume(" << index << ") = {" << index <<
"};" << std::endl;
    index = index + 1;
    ( * outfile) << "Surface Loop(" << index << ") = {" << start
+ 6 + 5 * i << ", " << start + 1007 + 5 * i << ", " << start +
4007 + 14 * i << ", " << start + 4010 + 14 * i << ", " << start
+ 4016 + 14 * i << ", " << start + 4018 + 14 * i << "};" <<
std::endl;
    ( * outfile) << "Volume(" << index << ") = {" << index <<
"};" << std::endl;
    index = index + 1;
  }
  // create lozenge in middle of wires
  if (N > 1) {
    int base_surface_lozenge = 7 + 5 * (N - 1);
    for (int i = 0; i < N - 1; i++) {
      ( * outfile) << "Surface Loop(" << index << ") = {" <<
start + base_surface_lozenge + i << ", " << start + 1000 +
base_surface_lozenge + 1 + i << ", " << start + 4000 + 14 * (i
+ 1) << ", " << start + 4001 + 14 * (i + 1) << ", " << start +
4009 + 14 * (i + 1) << ", " << start + 4010 + 14 * (i + 1) <<
"};" << std::endl;
      ( * outfile) << "Volume(" << index << ") = {" << index <<
"};" << std::endl;
      index = index + 1;
    }
  }
  // final section
  int base = (base_surface - 1) + 6 * N + 1;
  ( * outfile) << "Surface Loop(" << index << ") = {" << base
<< ", " << base + 1000 + 1 << ", " << base + 4011 + 1 + 8 * (N
- 1) << ", " << base + 4012 + 1 + 8 * (N - 1) << ", " << base +
4013 + 1 + 8 * (N - 1) << ", " << base + 4014 + 1 + 8 * (N - 1)
<< "};" << std::endl;
  ( * outfile) << "Volume(" << index << ") = {" << index << "};"
<< std::endl;
  index = index + 1;
  // create triangle of final section
  base = base + 1;
  ( * outfile) << "Surface Loop(" << index << ") = {" << base
<< ", " << base + 1000 + 1 << ", " << base + 4005 + 1 + 8 * (N
- 1) << ", " << base + 4006 + 1 + 8 * (N - 1) << ", " << base +
4010 + 1 + 8 * (N - 1) << "};" << std::endl;
  ( * outfile) << "Volume(" << index << ") = {" << index << "};"
<< std::endl;
  index = index + 1;
}
```

```
void            create_base_volumes_middle_section_y_shift(int
base_surface, int loop_index, std::ofstream * outfile) {
  int index = loop_index;
  int start = base_surface - 1;
  //first section
  ( * outfile) << "Surface Loop(" << index << ") = {" << start
+ 1 << ", " << start + 1001 << ", " << start + 4001 << ", " <<
start + 4002 << ", " << start + 4003 << ", " << start + 4004 <<
"};" << std::endl;
  ( * outfile) << "Volume(" << index << ") = {" << index << "};"
<< std::endl;
  index = index + 1;
  // wire section
  for (int i = 0; i < N; i++) {
    ( * outfile) << "Surface Loop(" << index << ") = {" << start
+ 3 + 5 * i << ", " << start + 1003 + 5 * i << ", " << start +
4005 + 14 * i << ", " << start + 4006 + 14 * i << ", " << start
+ 4007 + 14 * i << ", " << start + 4008 + 14 * i << "};" <<
std::endl;
    ( * outfile) << "Volume(" << index << ") = {" << index <<
"};" << std::endl;
    index = index + 1;
  }
  // final section
  int base = (base_surface - 1) + 6 * N + 2;
  ( * outfile) << "Surface Loop(" << index << ") = {" << base
<< ", " << base + 1000 << ", " << base + 4011 + 8 * (N - 1) <<
", " << base + 4012 + 8 * (N - 1) << ", " << base + 4013 + 8 *
(N - 1) << ", " << base + 4014 + 8 * (N - 1) << "};" << std::endl;
  ( * outfile) << "Volume(" << index << ") = {" << index << "};"
<< std::endl;
  index = index + 1;
}
// discover of base mid point
int calculate_middle_point_of_struct(int N) {
  int base_mid_point = 5;
  for (int i = 0; i < N - 1; i++) {
    if (i % 2 == 0) {
      base_mid_point += 7;
    } else {
      base_mid_point += 2;
    }
  }
  return base_mid_point;
}
// creation of connection between structures
void  create_connection_between_structs(int  base_number_lines,
int loop_index, int surface_loop_index, std::ofstream * outfile)
{
  int mid_point = calculate_middle_point_of_struct(N);
  int before_mid_point = 0;
  int after_mid_point = 0;
  if (N % 2 == 0) {
    before_mid_point = mid_point - 7;
    after_mid_point = mid_point + 2;
```

```
  } else {
    before_mid_point = mid_point - 2;
    after_mid_point = mid_point + 7;
  }
  int line = base_number_lines;
  // discover starting line index
  int base_line_before_mid_point = 14;
  int base_line_left_before_mid_point = 4010;
  int base_line_right_after_mid_point = 4004;
  int base_line_after_mid_point = 13;
  if (N % 2 == 0) {
    for (int i = 0; i < N; i = i + 2) {
      base_line_before_mid_point += 14;
      base_line_left_before_mid_point += 8;
      base_line_right_after_mid_point += 8;
      base_line_after_mid_point += 14;
    }
    if (N > 2) {
      base_line_before_mid_point = base_line_before_mid_point -
1;
      base_line_left_before_mid_point                          =
base_line_left_before_mid_point;
      base_line_right_after_mid_point                          =
base_line_right_after_mid_point - 2;
      base_line_after_mid_point  =  base_line_after_mid_point  -
13;
    } else {
      base_line_before_mid_point = base_line_before_mid_point -
1;
      base_line_left_before_mid_point                          =
base_line_left_before_mid_point;
      base_line_right_after_mid_point                          =
base_line_right_after_mid_point - 2;
      base_line_after_mid_point  =  base_line_after_mid_point  -
13;
    }
  } else {
    for (int i = 1; i < N; i = i + 2) {
      base_line_before_mid_point += 14;
      base_line_left_before_mid_point += 8;
      base_line_right_after_mid_point += 8;
      base_line_after_mid_point += 14;
    }
    if (N > 1) {
      base_line_before_mid_point = base_line_before_mid_point;
      base_line_left_before_mid_point                          =
base_line_left_before_mid_point + 2;
      base_line_right_after_mid_point                          =
base_line_right_after_mid_point;
      base_line_after_mid_point = base_line_after_mid_point;
    } else {
      base_line_before_mid_point = base_line_before_mid_point;
      base_line_left_before_mid_point                          =
base_line_left_before_mid_point + 2;
      base_line_right_after_mid_point                          =
```

```
base_line_right_after_mid_point;
      base_line_after_mid_point = base_line_after_mid_point;
    }
  }
  // write connection between structs
  int index = loop_index;
  int surface_index = surface_loop_index;
  //square of  bottom base of struct n°1
  ( * outfile) << "Line(" << line << ") = {" <<
(before_mid_point) << ", " << (before_mid_point + 1000) << "};"
<< std::endl;
  line = line + 1;
  ( * outfile) << "Line(" << line << ") = {" << (after_mid_point)
<< ", " << (after_mid_point + 1000) << "};" << std::endl;
  line = line + 1;
  // surface base bottom 1st struct
  ( * outfile) << "Line Loop(" << index << ") = {" <<
base_line_left_before_mid_point << ", -" <<
base_line_before_mid_point + 1000 << ", -" <<
base_line_after_mid_point + 1000 << ", -" <<
base_line_right_after_mid_point << ", " <<
base_line_after_mid_point << ", " << base_line_before_mid_point
<< "};" << std::endl;
  ( * outfile) << "Plane Surface(" << index << ") = {" << index
<< "};" << std::endl;
  index = index + 1;
  //square of  top base of struct n°1
  ( * outfile) << "Line(" << line << ") = {" << (before_mid_point
+ 2000) << ", " << (before_mid_point + 3000) << "};" <<
std::endl;
  line = line + 1;
  ( * outfile) << "Line(" << line << ") = {" << (after_mid_point
+ 2000) << ", " << (after_mid_point + 3000) << "};" << std::endl;
  line = line + 1;
  // surface base  top 1st struct
  ( * outfile) << "Line Loop(" << index << ") = {" <<
base_line_left_before_mid_point + 2000 << ", -" <<
base_line_before_mid_point + 3000 << ", -" <<
base_line_after_mid_point + 3000 << ", -" <<
base_line_right_after_mid_point + 2000 << ", " <<
base_line_after_mid_point + 2000 << ", " <<
base_line_before_mid_point + 2000 << "};" << std::endl;
  ( * outfile) << "Plane Surface(" << index << ") = {" << index
<< "};" << std::endl;
  index = index + 1;
  //square of  bottom base of struct n°2
  ( * outfile) << "Line(" << line << ") = {" << (before_mid_point
+ 10000 - 1) << ", " << (before_mid_point + 11000) << "};" <<
std::endl;
  line = line + 1;
  ( * outfile) << "Line(" << line << ") = {" << (after_mid_point
+ 10000 - 1) << ", " << (after_mid_point + 11000) << "};" <<
std::endl;
  line = line + 1;
  // surface base bottom 2nd struct
```

```
  ( * outfile) << "Line Loop(" << index << ") = {" <<
base_line_left_before_mid_point - 1 + 10000 << ", -" <<
base_line_before_mid_point - 2 + 11000 << ", -" <<
base_line_after_mid_point - 2 + 11000 << ", -" <<
base_line_right_after_mid_point - 1 + 10000 << ", " <<
base_line_after_mid_point - 3 + 10000 << ", " <<
base_line_before_mid_point - 3 + 10000 << "};" << std::endl;
  ( * outfile) << "Plane Surface(" << index << ") = {" << index
<< "};" << std::endl;
  index = index + 1;
  //square of top base of struct n°2
  ( * outfile) << "Line(" << line << ") = {" << (before_mid_point
+ 12000) << ", " << (before_mid_point + 13000) << "};" <<
std::endl;
  line = line + 1;
  ( * outfile) << "Line(" << line << ") = {" << (after_mid_point
+ 12000) << ", " << (after_mid_point + 13000) << "};" <<
std::endl;
  line = line + 1;
  // surface base top 2nd struct
  ( * outfile) << "Line Loop(" << index << ") = {" <<
base_line_left_before_mid_point - 1 + 12000 << ", -" <<
base_line_before_mid_point - 2 + 13000 << ", -" <<
base_line_after_mid_point - 2 + 13000 << ", -" <<
base_line_right_after_mid_point - 1 + 12000 << ", " <<
base_line_after_mid_point - 2 + 12000 << ", " <<
base_line_before_mid_point - 2 + 12000 << "};" << std::endl;
  ( * outfile) << "Plane Surface(" << index << ") = {" << index
<< "};" << std::endl;
  index = index + 1;
  // create transversal link
  ( * outfile) << "Line(" << line << ") = {" << (after_mid_point
+ 1) << ", " << (after_mid_point - 1 + 10000) << "};" <<
std::endl;
  line = line + 1;
  ( * outfile) << "Line(" << line << ") = {" << (after_mid_point
+ 1 + 1000) << ", " << (after_mid_point + 11000) << "};" <<
std::endl;
  line = line + 1;
  ( * outfile) << "Line(" << line << ") = {" << (after_mid_point
+ 1 + 2000) << ", " << (after_mid_point + 12000) << "};" <<
std::endl;
  line = line + 1;
  ( * outfile) << "Line(" << line << ") = {" << (after_mid_point
+ 1 + 3000) << ", " << (after_mid_point + 13000) << "};" <<
std::endl;
  line = line + 1;
  ( * outfile) << "Line(" << line << ") = {" << (before_mid_point
+ 1) << ", " << (before_mid_point - 1 + 10000) << "};" <<
std::endl;
  line = line + 1;
  ( * outfile) << "Line(" << line << ") = {" << (before_mid_point
+ 1 + 1000) << ", " << (before_mid_point + 11000) << "};" <<
std::endl;
  line = line + 1;
```

```
  ( * outfile) << "Line(" << line << ") = {" << (before_mid_point
+ 1 + 2000) << ", " << (before_mid_point + 12000) << "};" <<
std::endl;
  line = line + 1;
  ( * outfile) << "Line(" << line << ") = {" << (before_mid_point
+ 1 + 3000) << ", " << (before_mid_point + 13000) << "};" <<
std::endl;
  line = line + 1;
  // surface base bottom 1st transversal link
  ( * outfile) << "Line Loop(" << index << ") = {" <<
base_number_lines + 8 << ", -" << base_line_before_mid_point -
3 + 10000 << ", -" << base_line_after_mid_point - 3 + 10000 <<
",   -"   <<   base_number_lines   +   12   <<   ",   "   <<
base_line_after_mid_point << ", " << base_line_before_mid_point
<< "};" << std::endl;
  ( * outfile) << "Plane Surface(" << index << ") = {" << index
<< "};" << std::endl;
  index = index + 1;
  // surface base top 1st transversal link
  ( * outfile) << "Line Loop(" << index << ") = {" <<
base_number_lines + 9 << ", -" << base_line_before_mid_point -
2 + 11000 << ", -" << base_line_after_mid_point - 2 + 11000 <<
",   -"   <<   base_number_lines   +   13   <<   ",   "   <<
base_line_after_mid_point    +    1000    <<    ",    "    <<
base_line_before_mid_point + 1000 << "};" << std::endl;
  ( * outfile) << "Plane Surface(" << index << ") = {" << index
<< "};" << std::endl;
  index = index + 1;
  // lateral left surface of 1st transversal link
  ( * outfile) << "Line Loop(" << index << ") = {" <<
base_number_lines + 8 << ", " << base_line_left_before_mid_point
- 1 + 10000 << ", -" << base_number_lines + 9 << ", -" <<
base_line_left_before_mid_point << "};" << std::endl;
  ( * outfile) << "Plane Surface(" << index << ") = {" << index
<< "};" << std::endl;
  index = index + 1;
  // lateral right surface of 1st transversal link
  ( * outfile) << "Line Loop(" << index << ") = {" <<
base_number_lines     +     12     <<     ",     "     <<
base_line_right_after_mid_point  -  1  +  10000  <<  ",  -"  <<
base_number_lines     +     13     <<     ",     -"     <<
base_line_right_after_mid_point << "};" << std::endl;
  ( * outfile) << "Plane Surface(" << index << ") = {" << index
<< "};" << std::endl;
  index = index + 1;
  // surface base bottom 2nd transversal link
  ( * outfile) << "Line Loop(" << index << ") = {" <<
base_number_lines + 10 << ", -" << base_line_before_mid_point -
2 + 12000 << ", -" << base_line_after_mid_point - 2 + 12000 <<
",   -"   <<   base_number_lines   +   14   <<   ",   "   <<
base_line_after_mid_point    +    2000    <<    ",    "    <<
base_line_before_mid_point + 2000 << "};" << std::endl;
  ( * outfile) << "Plane Surface(" << index << ") = {" << index
<< "};" << std::endl;
  index = index + 1;
```

```
  // surface base top 2nd transversal link
  ( * outfile) << "Line Loop(" << index << ") = {" <<
base_number_lines + 11 << ", -" << base_line_before_mid_point -
2 + 13000 << ", -" << base_line_after_mid_point - 2 + 13000 <<
",  -"  <<  base_number_lines  +  15  <<  ",  "  <<
base_line_after_mid_point  +  3000  <<  ",  "  <<
base_line_before_mid_point + 3000 << "};" << std::endl;
  ( * outfile) << "Plane Surface(" << index << ") = {" << index
<< "};" << std::endl;
  index = index + 1;
  // lateral left surface of 2nd transversal link
  ( * outfile) << "Line Loop(" << index << ") = {" <<
base_number_lines  +  10  <<  ",  "  <<
base_line_left_before_mid_point - 1 + 12000 << ", -" <<
base_number_lines  +  11  <<  ",  -"  <<
base_line_left_before_mid_point + 2000 << "};" << std::endl;
  ( * outfile) << "Plane Surface(" << index << ") = {" << index
<< "};" << std::endl;
  index = index + 1;
  // lateral right surface of 2nd transversal link
  ( * outfile) << "Line Loop(" << index << ") = {" <<
base_number_lines  +  14  <<  ",  "  <<
base_line_right_after_mid_point - 1 + 12000 << ", -" <<
base_number_lines  +  15  <<  ",  -"  <<
base_line_right_after_mid_point + 2000 << "};" << std::endl;
  ( * outfile) << "Plane Surface(" << index << ") = {" << index
<< "};" << std::endl;
  index = index + 1;
  // create volumes for each link
  ( * outfile) << "Surface Loop(" << surface_index << ") = {"
<< loop_index << ", " << loop_index + 2 << ", " << loop_index +
4 << ", " << loop_index + 5 << ", " << loop_index + 6 << ", "
<< loop_index + 7 << "};" << std::endl;
  ( * outfile) << "Volume(" << surface_index << ") = {" <<
surface_index << "};" << std::endl;
  surface_index = surface_index + 1;
  ( * outfile) << "Surface Loop(" << surface_index << ") = {"
<< loop_index + 1 << ", " << loop_index + 3 << ", " << loop_index
+ 8 << ", " << loop_index + 9 << ", " << loop_index + 10 << ",
" << loop_index + 11 << "};" << std::endl;
  ( * outfile) << "Volume(" << surface_index << ") = {" <<
surface_index << "};" << std::endl;
  surface_index = surface_index + 1;
}
//define physical volumes
define_physical_group_wires(int  base_volume,  std::ofstream  *
outfile) {
  int start = base_volume - 1;
  ( * outfile) << "Physical Volume(\"Wires\") = {";
  for (int i = 0; i < N; i++) {
    int elm1 = start + 3 + 5 * i;
    ( * outfile) << elm1;
    ( * outfile) << ", ";
      int elm2 = start + 1002 + i;
    ( * outfile) << elm2;
```

```
    ( * outfile) << ", ";
    int elm3 = start + 2003 + 5 * i;
    ( * outfile) << elm3;
    ( * outfile) << ", ";
    int elm4 = start + 10003 + 5 * i;
    ( * outfile) << elm4;
    ( * outfile) << ", ";

    int elm5 = start + 11002 + i;
    ( * outfile) << elm5;
    ( * outfile) << ", ";
    int elm6 = start + 12003 + 5 * i;
    ( * outfile) << elm6;
    if (i == N - 1) {
      ( * outfile) << "";
    } else {
      ( * outfile) << ", ";
    }
  }
  ( * outfile) << "};" << std::endl;
}
define_physical_group_air(int    base_volume,    std::ofstream
*outfile){
int start = base_volume -1;
 (*outfile) << "Physical Volume(\"Air\") = {";
 int init = start + 2 + 1;
 (*outfile) << init;
 (*outfile) << ", " ;
for (int i = 0; i < N; i++) {
int elm1 = start + 4 + 5*i+1;
(*outfile) << elm1;
(*outfile) << ", " ;
int elm2 = start + 5 + 5*i+1;
(*outfile) << elm2;
(*outfile) << ", " ;
int elm3 = start + 6 + 5*i+1;
(*outfile) << elm3;
(*outfile) << ", " ;
int elm4 = start + 7 + 5*i+1;
(*outfile) << elm4;
(*outfile) << ", " ;
}
// create lozenge in middle of wires
int base = (base_volume-1)+ 6*N + 2 ;
for (int i = 0; i < N-1; i++) {
int elm5 = base - i;
(*outfile) << elm5;
(*outfile) << ", " ;
}
int end = (base_volume-1) + 6*N + 4 ;
(*outfile) << end;
(*outfile) << "};"<< std::endl;
}
void       define_physical_group_plastic(int       base_volume,
std::ofstream * outfile) {
```

```
  int start = base_volume - 1;
  ( * outfile) << "Physical Volume(\"Plastic\") = {";
  ( * outfile) << start + 1;
  ( * outfile) << ", ";
  ( * outfile) << start + 2;
  ( * outfile) << ", ";
  ( * outfile) << start + 1 + 1000;
  ( * outfile) << ", ";
  ( * outfile) << start + 1 + 2000;
  ( * outfile) << ", ";
  ( * outfile) << start + 2 + 2000;
  ( * outfile) << ", ";
  ( * outfile) << start + 1 + 10000;
  ( * outfile) << ", ";
  ( * outfile) << start + 2 + 10000;
  ( * outfile) << ", ";
  ( * outfile) << start + 1 + 11000;
  ( * outfile) << ", ";
  ( * outfile) << start + 1 + 12000;
  ( * outfile) << ", ";
  ( * outfile) << start + 2 + 12000;
  ( * outfile) << ", ";
  // wire section
  for (int i = 0; i < N; i++) {
    ( * outfile) << start + 4 + 5 * i;
    ( * outfile) << ", ";
    ( * outfile) << start + 4 + 5 * i + 2000;
    ( * outfile) << ", ";
    ( * outfile) << start + 4 + 5 * i + 10000;
    ( * outfile) << ", ";
    ( * outfile) << start + 4 + 5 * i + 12000;
    ( * outfile) << ", ";
    ( * outfile) << start + 5 + 5 * i;
    ( * outfile) << ", ";
    ( * outfile) << start + 5 + 5 * i + 2000;
    ( * outfile) << ", ";
    ( * outfile) << start + 5 + 5 * i + 10000;
    ( * outfile) << ", ";
    ( * outfile) << start + 5 + 5 * i + 12000;
    ( * outfile) << ", ";
    ( * outfile) << start + 6 + 5 * i;
    ( * outfile) << ", ";
    ( * outfile) << start + 6 + 5 * i + 2000;
    ( * outfile) << ", ";
    ( * outfile) << start + 6 + 5 * i + 10000;
    ( * outfile) << ", ";
    ( * outfile) << start + 6 + 5 * i + 12000;
    ( * outfile) << ", ";
    ( * outfile) << start + 7 + 5 * i;
    ( * outfile) << ", ";
    ( * outfile) << start + 7 + 5 * i + 2000;
    ( * outfile) << ", ";
    ( * outfile) << start + 7 + 5 * i + 10000;
    ( * outfile) << ", ";
    ( * outfile) << start + 7 + 5 * i + 12000;
```

```
    ( * outfile) << ", ";
  }
  int base_volume_lozenge = 8 + 5 * (N - 1);
  for (int i = 0; i < N - 1; i++) {
    ( * outfile) << start + base_volume_lozenge + i;
    ( * outfile) << ", ";
    ( * outfile) << start + base_volume_lozenge + i + 2000;
    ( * outfile) << ", ";
    ( * outfile) << start + base_volume_lozenge + i + 10000;
    ( * outfile) << ", ";
    ( * outfile) << start + base_volume_lozenge + i + 12000;
    ( * outfile) << ", ";
  }
  // final section
  int base = (base_volume - 1) + 6 * N + 2;
  ( * outfile) << base;
  ( * outfile) << ", ";
  ( * outfile) << base_volume + 1000 + N + 1;
  ( * outfile) << ", ";
  ( * outfile) << base + 2000;
  ( * outfile) << ", ";
  ( * outfile) << base + 10000;
  ( * outfile) << ", ";
  ( * outfile) << base_volume + 11000 + N + 1;
  ( * outfile) << ", ";
  ( * outfile) << base + 12000;
  ( * outfile) << ", ";
  // create triangle of final section
  int tri = base + 1;
  ( * outfile) << tri;
  ( * outfile) << ",";
  base = base + 1000;
  ( * outfile) << tri;
  ( * outfile) << ",";
  tri = tri + 2000;
  ( * outfile) << tri;
  ( * outfile) << ",";
  ( * outfile) << tri + 8000;
  ( * outfile) << ",";
  ( * outfile) << tri + 4000;
  ( * outfile) << ",";
  tri = tri + 10000;
  ( * outfile) << tri;
  ( * outfile) << ",";
  ( * outfile) << 19001;
  ( * outfile) << ",";
  ( * outfile) << 19002;
  ( * outfile) << "";
  ( * outfile) << "};" << std::endl;
}
// define physical surfaces
define_physical_surface_left(int base_surface, std::ofstream *
outfile) {
  ( * outfile) << "Physical Surface(\"Left\") = {";
  int base = (base_surface - 1) + 6 * N + 2;
```

```
  ( * outfile) << base + 4000 + 8 * N + 4;
  ( * outfile) << ", ";
  ( * outfile) << base + 6000 + 8 * N + 4;
  ( * outfile) << ", ";
  ( * outfile) << base + 14000 + 8 * N + 4;
  ( * outfile) << ", ";
  ( * outfile) << base + 16000 + 8 * N + 4;
  ( * outfile) << "";
  ( * outfile) << "};" << std::endl;
}
define_physical_surface_gap_left(int              base_surface,
std::ofstream * outfile) {
  ( * outfile) << "Physical Surface(\"Gap Left\") = {";
  int base = (base_surface - 1) + 6 * N + 2;
  ( * outfile) << base + 5000 + 8 * N + 4;
  ( * outfile) << ", ";
  ( * outfile) << base + 15000 + 8 * N + 4;
  ( * outfile) << "";
  ( * outfile) << "};" << std::endl;
}
define_physical_surface_right(int base_surface, std::ofstream *
outfile) {
  ( * outfile) << "Physical Surface(\"Right\") = {";
  int start = base_surface - 1;
  ( * outfile) << start + 4001;
  ( * outfile) << ", ";
  //(*outfile) << start + 5001 ;
  //(*outfile)<< ", ";
  ( * outfile) << start + 6001;
  ( * outfile) << ", ";
  ( * outfile) << start + 14001;
  ( * outfile) << ", ";
  //(*outfile) << start + 15001 ;
  //(*outfile)<< ", ";
  ( * outfile) << start + 16001;
  ( * outfile) << "";
  ( * outfile) << "};" << std::endl;
}
define_physical_surface_gap_right(int             base_surface,
std::ofstream * outfile) {
  ( * outfile) << "Physical Surface(\"Gap Right\") = {";
  int start = base_surface - 1;
  ( * outfile) << start + 5001;
  ( * outfile) << ", ";
  ( * outfile) << start + 15001;
  ( * outfile) << "";
  ( * outfile) << "};" << std::endl;
}
define_physical_surface_bottom(int base_surface, std::ofstream
* outfile) {
  // first section
  int start = base_surface - 1;
  ( * outfile) << "Physical Surface(\"Bottom\") = {";
  ( * outfile) << start + 1;
  ( * outfile) << ", ";
```

```
  ( * outfile) << start + 2;
  ( * outfile) << ", ";
  ( * outfile) << base_surface - 1 + 10000;
  ( * outfile) << ", ";
  ( * outfile) << base_surface + 10000;
  ( * outfile) << ", ";
  // wire section
  for (int i = 0; i < N; i++) {
    ( * outfile) << start + 3 + 5 * i;
    ( * outfile) << ", ";
    ( * outfile) << base_surface + 1 + 5 * i + 10000;
    ( * outfile) << ", ";
    ( * outfile) << start + 4 + 5 * i;
    ( * outfile) << ", ";
    ( * outfile) << base_surface + 2 + 5 * i + 10000;
    ( * outfile) << ", ";
    ( * outfile) << start + 5 + 5 * i;
    ( * outfile) << ", ";
    ( * outfile) << base_surface + 3 + 5 * i + 10000;
    ( * outfile) << ", ";
    ( * outfile) << start + 6 + 5 * i;
    ( * outfile) << ", ";
    ( * outfile) << base_surface + 4 + 5 * i + 10000;
    ( * outfile) << ", ";
    ( * outfile) << start + 7 + 5 * i;
    ( * outfile) << ", ";
    ( * outfile) << base_surface + 5 + 5 * i + 10000;
    ( * outfile) << ", ";
  }
  if (N > 1) {
    int base_surface_lozenge = 8 + 5 * (N - 1);
    for (int i = 0; i < N - 1; i++) {
      ( * outfile) << start + base_surface_lozenge + i;
      ( * outfile) << ", ";
      ( * outfile) << base_surface - 1 + base_surface_lozenge +
i + 10000 - 1;
      ( * outfile) << ", ";
    }
  }
  // final section
  int base = (base_surface - 1) + 6 * N + 2;
  ( * outfile) << base;
  ( * outfile) << ", ";
  ( * outfile) << base - 1 + 10000;
  ( * outfile) << ", ";
  // create triangle of final section
  base = base + 1;
  ( * outfile) << base;
  ( * outfile) << ",";
  base = base + 1;
  ( * outfile) << base - 2 + 10000;
  ( * outfile) << ",";
  //link between structs
  ( * outfile) << 18005;
  ( * outfile) << "";
```

```
  ( * outfile) << "};" << std::endl;
}
define_physical_surface_top(int base_surface, std::ofstream *
outfile) {
  // first section
  int start = base_surface - 1;
  ( * outfile) << "Physical Surface(\"Top\") = {";
  ( * outfile) << start + 1 + 3000;
  ( * outfile) << ", ";
  ( * outfile) << start + 2 + 3000;
  ( * outfile) << ", ";
  ( * outfile) << start + 1 + 13000;
  ( * outfile) << ", ";
  ( * outfile) << start + 2 + 13000;
  ( * outfile) << ", ";
  // wire section
  for (int i = 0; i < N; i++) {
    ( * outfile) << start + 3 + 5 * i + 3000;
    ( * outfile) << ", ";
    ( * outfile) << start + 3 + 5 * i + 13000;
    ( * outfile) << ", ";
    ( * outfile) << start + 4 + 5 * i + 3000;
    ( * outfile) << ", ";
    ( * outfile) << start + 4 + 5 * i + 13000;
    ( * outfile) << ", ";
    ( * outfile) << start + 5 + 5 * i + 3000;
    ( * outfile) << ", ";
    ( * outfile) << start + 5 + 5 * i + 13000;
    ( * outfile) << ", ";
    ( * outfile) << start + 6 + 5 * i + 3000;
    ( * outfile) << ", ";
    ( * outfile) << start + 6 + 5 * i + 13000;
    ( * outfile) << ", ";
    ( * outfile) << start + 7 + 5 * i + 3000;
    ( * outfile) << ", ";
    ( * outfile) << start + 7 + 5 * i + 13000;
    ( * outfile) << ", ";
  }
  if (N > 1) {
    int base_surface_lozenge = 8 + 5 * (N - 1);
    for (int i = 0; i < N - 1; i++) {
      ( * outfile) << start + base_surface_lozenge + i + 3000;
      ( * outfile) << ", ";
      ( * outfile) << start + base_surface_lozenge + i + 13000;
      ( * outfile) << ", ";
    }
  }
  // final section
  int base = (base_surface - 1) + 6 * N + 2 + 3000;
  ( * outfile) << base;
  ( * outfile) << ", ";
  ( * outfile) << base + 10000;
  ( * outfile) << ", ";
  // create triangle of final section
  base = base + 1;
```

```
  ( * outfile) << base;
  ( * outfile) << ",";
  ( * outfile) << base + 10000;
  ( * outfile) << ",";
  //link between structs
  ( * outfile) << 18010;
  ( * outfile) << "";
  ( * outfile) << "};" << std::endl;
}
define_physical_surface_front(int base_surface, std::ofstream *
outfile) {
  //######front base
  int start = base_surface - 1;
  ( * outfile) << "Physical Surface(\"Front\") = {";
  ( * outfile) << start + 14003;
  ( * outfile) << ", ";
  ( * outfile) << start + 15003;
  ( * outfile) << ", ";
  ( * outfile) << start + 16003;
  ( * outfile) << ", ";
  ( * outfile) << start + 15004;
  ( * outfile) << ", ";
  for (int i = 0; i < N; i++) {
    int base_surface_lozenge = 4 + 14 * (N - i) - 1;
    ( * outfile) << start + base_surface_lozenge + 1 + 14000;
    ( * outfile) << ", ";
    ( * outfile) << start + base_surface_lozenge + 1 + 15000;
    ( * outfile) << ", ";
    ( * outfile) << start + base_surface_lozenge + 1 + 16000;
    ( * outfile) << ", ";
    ( * outfile) << start + base_surface_lozenge + 14000;
    ( * outfile) << ", ";
    ( * outfile) << start + base_surface_lozenge + 15000;
    ( * outfile) << ", ";
    ( * outfile) << start + base_surface_lozenge + 16000;
    ( * outfile) << ", ";
  }
  // final section
  int base = (base_surface - 1) + 7 + 14 * N + 1;
  ( * outfile) << base + 14000;
  ( * outfile) << ", ";
  ( * outfile) << base + 15000;
  ( * outfile) << ", ";
  ( * outfile) << (base - 3) + 15000;
  ( * outfile) << ", ";
  ( * outfile) << base + 16000;
  ( * outfile) << ", ";
  //####front inside
  ( * outfile) << base_surface - 1 + 10000;
  ( * outfile) << ", ";
  ( * outfile) << base_surface + 10000;
  ( * outfile) << ", ";
  ( * outfile) << base_surface - 1 + 11000;
  ( * outfile) << ", ";
  ( * outfile) << base_surface + 11000;
```

```
  ( * outfile) << ", ";
  ( * outfile) << base_surface - 1 + 12000;
  ( * outfile) << ", ";
  ( * outfile) << base_surface + 12000;
  ( * outfile) << ", ";
  ( * outfile) << base_surface - 1 + 13000;
  ( * outfile) << ", ";
  ( * outfile) << base_surface + 13000;
  ( * outfile) << ", ";
  // wire section
  int lateral_init = 0;
  int lateral_1 = 0;
  int lateral_2 = 0;
  int lateral_3 = 0;
  int lateral_4 = 0;
  if (N == 1) {
    lateral_init = 5;
  } else {
    lateral_init = 5;
    for (int i = 1; i < N; i++) {
      lateral_init += 14;
    }
  }
  for (int i = 0; i < N; i++) {
    //last base surfaces
    ( * outfile) << base_surface + 1 + 5 * i + 10000;
    ( * outfile) << ", ";
    ( * outfile) << base_surface + 1 + 5 * i + 11000;
    ( * outfile) << ", ";
    ( * outfile) << base_surface + 1 + 5 * i + 12000;
    ( * outfile) << ", ";
    ( * outfile) << base_surface + 1 + 5 * i + 13000;
    ( * outfile) << ", ";
    //lateral surfaces
    ( * outfile) << lateral_init + 14000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_init + 15000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_init + 16000;
    ( * outfile) << ", ";
    lateral_1 = lateral_init;
    ( * outfile) << lateral_1 + 14000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_1 + 15000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_1 + 16000;
    ( * outfile) << ", ";
    lateral_2 = lateral_1 + 1;
    ( * outfile) << lateral_2 + 14000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_2 + 15000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_2 + 16000;
    ( * outfile) << ", ";
    lateral_3 = lateral_2 + 1;
```

```
    ( * outfile) << lateral_3 + 14000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_3 + 15000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_3 + 16000;
    ( * outfile) << ", ";
    lateral_4 = lateral_3 + 1;
    ( * outfile) << lateral_4 + 14000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_4 + 15000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_4 + 16000;
    ( * outfile) << ", ";
    ( * outfile) << base_surface + 2 + 5 * i + 10000;
    ( * outfile) << ", ";
    ( * outfile) << base_surface + 2 + 5 * i + 11000;
    ( * outfile) << ", ";
    ( * outfile) << base_surface + 2 + 5 * i + 12000;
    ( * outfile) << ", ";
    ( * outfile) << base_surface + 2 + 5 * i + 13000;
    ( * outfile) << ", ";
    //lateral surfaces
    lateral_init = lateral_init - 14;
    ( * outfile) << lateral_init + 14000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_init + 15000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_init + 16000;
    ( * outfile) << ", ";
    lateral_1 = lateral_init;
    ( * outfile) << lateral_1 + 14000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_1 + 15000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_1 + 16000;
    ( * outfile) << ", ";
    lateral_2 = lateral_1 + 1;
    ( * outfile) << lateral_2 + 14000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_2 + 15000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_2 + 16000;
    ( * outfile) << ", ";
    lateral_3 = lateral_2 + 1;
    ( * outfile) << lateral_3 + 14000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_3 + 15000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_3 + 16000;
    ( * outfile) << ", ";
    lateral_4 = lateral_3 + 1;
    ( * outfile) << lateral_4 + 14000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_4 + 15000;
    ( * outfile) << ", ";
```

```
    ( * outfile) << lateral_4 + 16000;
    ( * outfile) << ", ";
    ( * outfile) << base_surface + 3 + 5 * i + 10000;
    ( * outfile) << ", ";
    ( * outfile) << base_surface + 3 + 5 * i + 11000;
    ( * outfile) << ", ";
    ( * outfile) << base_surface + 3 + 5 * i + 12000;
    ( * outfile) << ", ";
    ( * outfile) << base_surface + 3 + 5 * i + 13000;
    ( * outfile) << ", ";
    //lateral surfaces
    lateral_init = lateral_init - 14;
    ( * outfile) << lateral_init + 14000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_init + 15000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_init + 16000;
    ( * outfile) << ", ";
    lateral_1 = lateral_init;
    ( * outfile) << lateral_1 + 14000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_1 + 15000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_1 + 16000;
    ( * outfile) << ", ";
    lateral_2 = lateral_1 + 1;
    ( * outfile) << lateral_2 + 14000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_2 + 15000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_2 + 16000;
    ( * outfile) << ", ";
    lateral_3 = lateral_2 + 1;
    ( * outfile) << lateral_3 + 14000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_3 + 15000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_3 + 16000;
    ( * outfile) << ", ";
    lateral_4 = lateral_3 + 1;
    ( * outfile) << lateral_4 + 14000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_4 + 15000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_4 + 16000;
    ( * outfile) << ", ";
    ( * outfile) << base_surface + 4 + 5 * i + 10000;
    ( * outfile) << ", ";
    ( * outfile) << base_surface + 4 + 5 * i + 11000;
    ( * outfile) << ", ";
    ( * outfile) << base_surface + 4 + 5 * i + 12000;
    ( * outfile) << ", ";
    ( * outfile) << base_surface + 4 + 5 * i + 13000;
    ( * outfile) << ", ";
    //lateral surfaces
```

```
    lateral_init = lateral_init - 14;
    ( * outfile) << lateral_init + 14000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_init + 15000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_init + 16000;
    ( * outfile) << ", ";
    lateral_1 = lateral_init;
    ( * outfile) << lateral_1 + 14000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_1 + 15000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_1 + 16000;
    ( * outfile) << ", ";
    lateral_2 = lateral_1 + 1;
    ( * outfile) << lateral_2 + 14000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_2 + 15000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_2 + 16000;
    ( * outfile) << ", ";
    lateral_3 = lateral_2 + 1;
    ( * outfile) << lateral_3 + 14000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_3 + 15000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_3 + 16000;
    ( * outfile) << ", ";
    lateral_4 = lateral_3 + 1;
    ( * outfile) << lateral_4 + 14000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_4 + 15000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_4 + 16000;
    ( * outfile) << ", ";
    ( * outfile) << base_surface + 5 + 5 * i + 10000;
    ( * outfile) << ", ";
    ( * outfile) << base_surface + 5 + 5 * i + 11000;
    ( * outfile) << ", ";
    ( * outfile) << base_surface + 5 + 5 * i + 12000;
    ( * outfile) << ", ";
    ( * outfile) << base_surface + 5 + 5 * i + 13000;
    ( * outfile) << ", ";
    //lateral surfaces
    lateral_init = lateral_init - 14;
    ( * outfile) << lateral_init + 14000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_init + 15000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_init + 16000;
    ( * outfile) << ", ";
    lateral_1 = lateral_init;
    ( * outfile) << lateral_1 + 14000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_1 + 15000;
```

```
    ( * outfile) << ", ";
    ( * outfile) << lateral_1 + 16000;
    ( * outfile) << ", ";
    lateral_2 = lateral_1 + 1;
    ( * outfile) << lateral_2 + 14000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_2 + 15000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_2 + 16000;
    ( * outfile) << ", ";
    lateral_3 = lateral_2 + 1;
    ( * outfile) << lateral_3 + 14000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_3 + 15000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_3 + 16000;
    ( * outfile) << ", ";
    lateral_4 = lateral_3 + 1;
    ( * outfile) << lateral_4 + 14000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_4 + 15000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_4 + 16000;
    ( * outfile) << ", ";
    //      first
    ( * outfile) << base_surface + 6 + 5 * i + 10000;
    ( * outfile) << ", ";
    ( * outfile) << base_surface + 6 + 5 * i + 11000;
    ( * outfile) << ", ";
    ( * outfile) << base_surface + 6 + 5 * i + 12000;
    ( * outfile) << ", ";
    ( * outfile) << base_surface + 6 + 5 * i + 13000;
    ( * outfile) << ", ";
    //lateral surfaces
    lateral_init = lateral_init - 14;
    ( * outfile) << lateral_init + 14000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_init + 15000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_init + 16000;
    ( * outfile) << ", ";
    lateral_1 = lateral_init;
    ( * outfile) << lateral_1 + 14000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_1 + 15000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_1 + 16000;
    ( * outfile) << ", ";
    lateral_2 = lateral_1 + 1;
    ( * outfile) << lateral_2 + 14000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_2 + 15000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_2 + 16000;
    ( * outfile) << ", ";
```

```
    lateral_3 = lateral_2 + 1;
    ( * outfile) << lateral_3 + 14000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_3 + 15000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_3 + 16000;
    ( * outfile) << ", ";
    lateral_4 = lateral_3 + 1;
    ( * outfile) << lateral_4 + 14000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_4 + 15000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_4 + 16000;
    ( * outfile) << ", ";
  }
  if (N > 1) {
    int base_surface_lozenge = 8 + 5 * (N - 1);
    for (int i = 0; i < N - 1; i++) {
      ( * outfile) << base_surface + base_surface_lozenge + i +
10000 - 1;
      ( * outfile) << ", ";
      ( * outfile) << base_surface + base_surface_lozenge + i +
11000 - 1;
      ( * outfile) << ", ";
      ( * outfile) << base_surface + base_surface_lozenge + i +
12000 - 1;
      ( * outfile) << ", ";
      ( * outfile) << base_surface + base_surface_lozenge + i +
13000 - 1;
      ( * outfile) << ", ";
    }
  }
  // final section
  int base_ = (base_surface - 1) + 6 * N + 2;
  ( * outfile) << base_ + 10000;
  ( * outfile) << ", ";
  ( * outfile) << base_ + 11000;
  ( * outfile) << ", ";
  ( * outfile) << base_ + 12000;
  ( * outfile) << ", ";
  ( * outfile) << base_ + 13000;
  ( * outfile) << ", ";
  // create triangle of final section
  base_ = base_ + 1;
  ( * outfile) << base_ + 10000;
  ( * outfile) << ",";
  ( * outfile) << base_ + 11000;
  ( * outfile) << ",";
  ( * outfile) << base_ + 12000;
  ( * outfile) << ", ";
  ( * outfile) << base_ + 13000;
  ( * outfile) << ", ";
  // back inside
  ( * outfile) << start + 14002;
  ( * outfile) << ", ";
```

```
  ( * outfile) << start + 15002;
  ( * outfile) << ", ";
  ( * outfile) << start + 16002;
  ( * outfile) << ", ";
  for (int i = 0; i < N; i++) {
    int base_surface_lozenge = 3 + 14 * (N - i) - 1;
    ( * outfile) << start + base_surface_lozenge - 3 + 14000;
    ( * outfile) << ", ";
    ( * outfile) << start + base_surface_lozenge - 3 + 15000;
    ( * outfile) << ", ";
    ( * outfile) << start + base_surface_lozenge - 3 + 16000;
    ( * outfile) << ", ";
    ( * outfile) << start + base_surface_lozenge - 5 + 14000;
    ( * outfile) << ", ";
    ( * outfile) << start + base_surface_lozenge - 5 + 15000;
    ( * outfile) << ", ";
    ( * outfile) << start + base_surface_lozenge - 5 + 16000;
    ( * outfile) << ", ";
  }
  // final section
  int base_b = (base_surface - 1) + 7 + 14 * N + 1;
  ( * outfile) << base_b - 1 + 14000;
  ( * outfile) << ", ";
  ( * outfile) << base_b - 1 + 15000;
  ( * outfile) << ", ";
  ( * outfile) << base_b - 1 + 16000;
  ( * outfile) << "";
  ( * outfile) << "};" << std::endl;
}
define_physical_surface_back(int base_surface, std::ofstream *
outfile) {
  int start = base_surface - 1;
  ( * outfile) << "Physical Surface(\"Back\") = {";
  ( * outfile) << start + 4002;
  ( * outfile) << ", ";
  ( * outfile) << start + 5002;
  ( * outfile) << ", ";
  ( * outfile) << start + 6002;
  ( * outfile) << ", ";
  for (int i = 0; i < N; i++) {
    int base_surface_lozenge = 14 * (N - i) - 3;
    ( * outfile) << start + base_surface_lozenge + 2 + 4000;
    ( * outfile) << ", ";
    ( * outfile) << start + base_surface_lozenge + 2 + 5000;
    ( * outfile) << ", ";
    ( * outfile) << start + base_surface_lozenge + 2 + 6000;
    ( * outfile) << ", ";
    ( * outfile) << start + base_surface_lozenge + 4000;
    ( * outfile) << ", ";
    ( * outfile) << start + base_surface_lozenge + 5000;
    ( * outfile) << ", ";
    ( * outfile) << start + base_surface_lozenge + 6000;
    ( * outfile) << ", ";
  }
  // final section
```

```
int base = (base_surface - 1) + 7 + 14 * N;
( * outfile) << base + 4000;
( * outfile) << ", ";
( * outfile) << base + 5000;
( * outfile) << ", ";
( * outfile) << base + 6000;
( * outfile) << ", ";
// inside surface
( * outfile) << (base - 2) + 5000;
( * outfile) << ", ";
//#### back inside
( * outfile) << start + 1;
( * outfile) << ", ";
( * outfile) << start + 2;
( * outfile) << ", ";
( * outfile) << start + 1 + 1000;
( * outfile) << ", ";
( * outfile) << start + 2 + 1000;
( * outfile) << ", ";
( * outfile) << start + 1 + 2000;
( * outfile) << ", ";
( * outfile) << start + 2 + 2000;
( * outfile) << ", ";
( * outfile) << start + 1 + 3000;
( * outfile) << ", ";
( * outfile) << start + 2 + 3000;
( * outfile) << ", ";
// wire section
int lateral_init = 0;
int lateral_1 = 0;
int lateral_2 = 0;
int lateral_3 = 0;
int lateral_4 = 0;
if (N == 1) {
  lateral_init = 5;
} else {
  lateral_init = 5;
  for (int i = 1; i < N; i++) {
    lateral_init += 14;
  }
  ( * outfile) << start + 4 + 5000;
  ( * outfile) << ", ";
 }
for (int i = 0; i < N; i++) {
  ( * outfile) << start + 3 + 5 * i;
  ( * outfile) << ", ";
  ( * outfile) << start + 3 + 5 * i + 1000;
  ( * outfile) << ", ";
  ( * outfile) << start + 3 + 5 * i + 2000;
  ( * outfile) << ", ";
  ( * outfile) << start + 3 + 5 * i + 3000;
  ( * outfile) << ", ";
  //lateral surfaces
  ( * outfile) << lateral_init + 4000;
  ( * outfile) << ", ";
```

```
    ( * outfile) << lateral_init + 5000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_init + 6000;
    ( * outfile) << ", ";
    lateral_1 = lateral_init;
    ( * outfile) << lateral_1 + 4000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_1 + 5000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_1 + 6000;
    ( * outfile) << ", ";
    lateral_2 = lateral_1 + 1;
    ( * outfile) << lateral_2 + 4000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_2 + 5000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_2 + 6000;
    ( * outfile) << ", ";
    lateral_3 = lateral_2 + 1;
    ( * outfile) << lateral_3 + 4000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_3 + 5000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_3 + 6000;
    ( * outfile) << ", ";
    lateral_4 = lateral_3 + 1;
    ( * outfile) << lateral_4 + 4000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_4 + 5000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_4 + 6000;
    ( * outfile) << ", ";
    ( * outfile) << start + 4 + 5 * i;
    ( * outfile) << ", ";
    ( * outfile) << start + 4 + 5 * i + 1000;
    ( * outfile) << ", ";
    ( * outfile) << start + 4 + 5 * i + 2000;
    ( * outfile) << ", ";
    ( * outfile) << start + 4 + 5 * i + 3000;
    ( * outfile) << ", ";
    lateral_init = lateral_init - 14;
    //lateral surfaces
    ( * outfile) << lateral_init + 4000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_init + 5000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_init + 6000;
    ( * outfile) << ", ";
    lateral_1 = lateral_init;
    ( * outfile) << lateral_1 + 4000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_1 + 5000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_1 + 6000;
    ( * outfile) << ", ";
```

```
    lateral_2 = lateral_1 + 1;
    ( * outfile) << lateral_2 + 4000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_2 + 5000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_2 + 6000;
    ( * outfile) << ", ";
    lateral_3 = lateral_2 + 1;
    ( * outfile) << lateral_3 + 4000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_3 + 5000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_3 + 6000;
    ( * outfile) << ", ";
    lateral_4 = lateral_3 + 1;
    ( * outfile) << lateral_4 + 4000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_4 + 5000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_4 + 6000;
    ( * outfile) << ", ";
    ( * outfile) << start + 5 + 5 * i;
    ( * outfile) << ", ";
    ( * outfile) << start + 5 + 5 * i + 1000;
    ( * outfile) << ", ";
    ( * outfile) << start + 5 + 5 * i + 2000;
    ( * outfile) << ", ";
    ( * outfile) << start + 5 + 5 * i + 3000;
    ( * outfile) << ", ";
    lateral_init = lateral_init - 14;
    //lateral surfaces
    ( * outfile) << lateral_init + 4000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_init + 5000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_init + 6000;
    ( * outfile) << ", ";
    lateral_1 = lateral_init;
    ( * outfile) << lateral_1 + 4000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_1 + 5000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_1 + 6000;
    ( * outfile) << ", ";
    lateral_2 = lateral_1 + 1;
    ( * outfile) << lateral_2 + 4000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_2 + 5000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_2 + 6000;
    ( * outfile) << ", ";
    lateral_3 = lateral_2 + 1;
    ( * outfile) << lateral_3 + 4000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_3 + 5000;
```

```
    ( * outfile) << ", ";
    ( * outfile) << lateral_3 + 6000;
    ( * outfile) << ", ";
    lateral_4 = lateral_3 + 1;
    ( * outfile) << lateral_4 + 4000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_4 + 5000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_4 + 6000;
    ( * outfile) << ", ";
    ( * outfile) << start + 6 + 5 * i;
    ( * outfile) << ", ";
    ( * outfile) << start + 6 + 5 * i + 1000;
    ( * outfile) << ", ";
    ( * outfile) << start + 6 + 5 * i + 2000;
    ( * outfile) << ", ";
    ( * outfile) << start + 6 + 5 * i + 3000;
    ( * outfile) << ", ";
    lateral_init = lateral_init - 14;
    //lateral surfaces
    ( * outfile) << lateral_init + 4000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_init + 5000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_init + 6000;
    ( * outfile) << ", ";
    lateral_1 = lateral_init;
    ( * outfile) << lateral_1 + 4000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_1 + 5000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_1 + 6000;
    ( * outfile) << ", ";
    lateral_2 = lateral_1 + 1;
    ( * outfile) << lateral_2 + 4000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_2 + 5000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_2 + 6000;
    ( * outfile) << ", ";
    lateral_3 = lateral_2 + 1;
    ( * outfile) << lateral_3 + 4000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_3 + 5000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_3 + 6000;
    ( * outfile) << ", ";
    lateral_4 = lateral_3 + 1;
    ( * outfile) << lateral_4 + 4000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_4 + 5000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_4 + 6000;
    ( * outfile) << ", ";
    ( * outfile) << start + 7 + 5 * i;
```

```
    ( * outfile) << ", ";
    ( * outfile) << start + 7 + 5 * i + 1000;
    ( * outfile) << ", ";
    ( * outfile) << start + 7 + 5 * i + 2000;
    ( * outfile) << ", ";
    ( * outfile) << start + 7 + 5 * i + 3000;
    ( * outfile) << ", ";
    lateral_init = lateral_init - 14;
    //lateral surfaces
    ( * outfile) << lateral_init + 4000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_init + 5000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_init + 6000;
    ( * outfile) << ", ";
    lateral_1 = lateral_init;
    ( * outfile) << lateral_1 + 4000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_1 + 5000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_1 + 6000;
    ( * outfile) << ", ";
    lateral_2 = lateral_1 + 1;
    ( * outfile) << lateral_2 + 4000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_2 + 5000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_2 + 6000;
    ( * outfile) << ", ";
    lateral_3 = lateral_2 + 1;
    ( * outfile) << lateral_3 + 4000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_3 + 5000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_3 + 6000;
    ( * outfile) << ", ";
    lateral_4 = lateral_3 + 1;
    ( * outfile) << lateral_4 + 4000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_4 + 5000;
    ( * outfile) << ", ";
    ( * outfile) << lateral_4 + 6000;
    ( * outfile) << ", ";
  }
  if (N > 1) {
    int base_surface_lozenge = 8 + 5 * (N - 1);
    for (int i = 0; i < N - 1; i++) {
      ( * outfile) << start + base_surface_lozenge + i + 1000;
      ( * outfile) << ", ";
      ( * outfile) << start + base_surface_lozenge + i + 2000;
      ( * outfile) << ", ";
    }
  }
  // final section
  int base_ = (base_surface - 1) + 6 * N + 2;
```

```
  ( * outfile) << base_;
  ( * outfile) << ", ";
  ( * outfile) << base_ + 1000;
  ( * outfile) << ", ";
  ( * outfile) << base_ + 2000;
  ( * outfile) << ", ";
  ( * outfile) << base_ + 3000;
  ( * outfile) << ", ";
  // create triangle of final section
  base = base + 1;
  ( * outfile) << base_ + 1;
  ( * outfile) <<",";
  ( * outfile) << base_ + 1 + 1000;
  ( * outfile) <<",";
  ( * outfile) << base_ + 1 + 2000;
  ( * outfile) <<",";
  ( * outfile) << base_ + 1 + 3000;
  ( * outfile) << ", ";
  // back inside
  ( * outfile) << start + 4003;
  ( * outfile) << ", ";
  ( * outfile) << start + 5003;
  ( * outfile) << ", ";
  ( * outfile) << start + 6003;
  ( * outfile) << ", ";
  for (int i = 0; i < N; i++) {
    int base_surface_lozenge = 4 + 14 * (N - i) - 1;
    ( * outfile) << start + base_surface_lozenge + 1 + 4000;
    ( * outfile) << ", ";
    ( * outfile) << start + base_surface_lozenge + 1 + 5000;
    ( * outfile) << ", ";
    ( * outfile) << start + base_surface_lozenge + 1 + 6000;
    ( * outfile) << ", ";
    ( * outfile) << start + base_surface_lozenge + 4000;
    ( * outfile) << ", ";
    ( * outfile) << start + base_surface_lozenge + 5000;
    ( * outfile) << ", ";
    ( * outfile) << start + base_surface_lozenge + 6000;
    ( * outfile) << ", ";
  }
  // final section
  int base_b = (base_surface - 1) + 7 + 14 * N + 1;
  ( * outfile) << base_b + 4000;
  ( * outfile) << ", ";
  ( * outfile) << base_b + 5000;
  ( * outfile) << ", ";
  ( * outfile) << base_b + 6000;
  ( * outfile) << "";
  ( * outfile) << "};" << std::endl;
}
define_physical_surface_junction(std::ofstream * outfile) {
  ( * outfile) << "Physical Surface(\"Junction\") = {";
  ( * outfile) << 18006;
  ( * outfile) << ", ";
  ( * outfile) << 18008;
```

```
  ( * outfile) << ", ";
  ( * outfile) << 18012;
  ( * outfile) << ", ";
  ( * outfile) << 18007;
  ( * outfile) << ", ";
  ( * outfile) << 18009;
  ( * outfile) << ", ";
  ( * outfile) << 18011;
  ( * outfile) << "";
  ( * outfile) << "};" << std::endl;
}
int main() {
  std::ofstream outfile;
  outfile.open("model.geo");
  // define parameters
  float a = 0.0002;
  float d = 0.02;
  create_parameter("w", 4.0 * a, & outfile);
  create_parameter("delta", 0.01, & outfile);
  create_parameter("d", d, & outfile);
  create_parameter("a", a, & outfile;
  create_parmeter("el", 0.1, & outfile);
  create_parameter("h", d / 2.0, & outfile);
  create_parameter("l", 1.42, & outfile);
  // creat 1st struct
  create_base_points("0", 1, & outfile);
  create_base_points("h", 1001, & outfile);
  create_base_points("l-h", 2001, & outfile);
  create_base_points("l", 3001, & outfile);
  create_base_lines("0", 1, 1, & outfile);
  create_base_lines("0", 1001, 1001, & outfile);
  create_base_lines("0", 2001, 2001, & outfile);
  create_base_lines("0", 3001, 3001, & outfile);
  connect(1, 1001, 4001, & outfile);
  connect(1001, 2001, 5001, & outfile);
  connect(2001, 3001, 6001, & outfile);
  create_base_surface(1, 1, & outfile);
  create_base_surface(1001, 1001, & outfile);
  create_base_surface(2001, 2001, & outfile);
  create_base_surface(3001, 3001, & outfile);
  create_lateral_surface(1, 1001, 4001, 4001, & outfile);
  create_lateral_surface(1001, 2001, 5001, 5001, & outfile);
  create_lateral_surface(2001, 3001, 6001, 6001, & outfile);
  create_base_volumes(1, 1, & outfile);
  create_base_volumes_middle_section(1001, 1001, & outfile);
  create_base_volumes(2001, 2001, & outfile);
  // create 2nd struct
  create_base_points_y_shift("0", 10000, & outfile);
  create_base_points_y_shift("h", 11001, & outfile);
  create_base_points_y_shift("l-h", 12001, & outfile);
  create_base_points_y_shift("l", 13001, & outfile);
  create_base_lines_y_shift("0", 10000, 10000, & outfile);
  create_base_lines_y_shift("0", 11001, 11001, & outfile);
  create_base_lines_y_shift("0", 12001, 12001, & outfile);
  create_base_lines_y_shift("0", 13001, 13001, & outfile);
```

```
  connect_y_shift(10000, 11001, 14001, & outfile);
  connect_y_shift(11001, 12001, 15001, & outfile);
  connect_y_shift(12001, 13001, 16001, & outfile);
  create_base_surface_y_shift(10000, 10000, & outfile);
  create_base_surface_y_shift(11001, 11001, & outfile);
  create_base_surface_y_shift(12001, 12001, & outfile);
  create_base_surface_y_shift(13001, 13001, & outfile);
  create_lateral_surface_y_shift(10000, 11001, 14001, 14001, &
outfile);
  create_lateral_surface_y_shift(11001, 12001, 15001, 15001, &
outfile);
  create_lateral_surface_y_shift(12001, 13001, 16001, 16001, &
outfile);
  create_base_volumes_y_shift(10001, 10001, & outfile);
  create_base_volumes_middle_section_y_shift(11001,  11001,  &
outfile);
  create_base_volumes(12001, 12001, & outfile);
  // connection between structs
  create_connection_between_structs(17001,  18001,  19001,  &
outfile);
  // define physical group volumes
  define_physical_group_wires(1, & outfile);
  define_physical_group_plastic(1, & outfile);
  // define physical surfaces
  define_physical_surface_left(1, & outfile);
  define_physical_surface_right(1, & outfile);
  define_physical_surface_gap_left(1, & outfile);
  define_physical_surface_gap_right(1, & outfile);
  define_physical_surface_bottom(1, & outfile);
  define_physical_surface_top(1, & outfile);
  define_physical_surface_front(1, & outfile);
  define_physical_surface_back(1, & outfile);
  define_physical_surface_junction(1, & outfile);
  outfile.close();
  return 0;
}
```