NOVA SCHOOL OF SCIENCE & TECHNOLOGY

Departamento de Engenharia Eletrotécnica e de Computadores

Pedro Alexandre Pereira da Silva

Licenciado em Ciências da Engenharia Eletrotécnica e de Computadores

Thermal Mapping and Air Quality Sensor Modules for Wildfire Suppression

Mestrado em Engenharia Eletrotécnica e de Computadores

NOVA University Lisbon

Novembro, 2021

Departamento de Engenharia Eletrotécnica
e de Computadores

Thermal Mapping and Air Quality Sensor Modules for Wildfire Suppression

Pedro Alexandre Pereira da Silva

Licenciado em Ciências da Engenharia Eletrotécnica e de Computadores

Orientador: Luís Oliveira
Professor Associado com Agregação, NOVA University Lisbon

Mestrado em Engenharia Eletrotécnica e de Computadores

NOVA University Lisbon
Novembro, 2021

[Thermal Mapping and Air Quality Sensor Modules for Wildfire Suppression]

Copyright © [Pedro Alexandre Pereira da Silva], Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa.

A Faculdade de Ciências e Tecnologia e a Universidade Nova de Lisboa têm o direito, perpétuo e sem limites geográficos, de arquivar e publicar esta dissertação através de exemplares impressos reproduzidos em papel ou de forma digital, ou por qualquer outro meio conhecido ou que venha a ser inventado, e de a divulgar através de repositórios científicos e de admitir a sua cópia e distribuição com objetivos educacionais ou de investigação, não comerciais, desde que seja dado crédito ao autor e editor.

To my dear grandmother, whom I miss every day.

Acknowledgements

I'd like to thank Professor Luís Oliveira for his continued support and guidance throughout this whole process, which proved instrumental in overcoming any roadblocks encountered.

My best wishes for a full recovery, as well as my gratitude go out to Miguel Lourenço, whose skill and knowledge of the project helped me understand its scope and create a far better result than would otherwise be possible.

I would also like to extend my gratitude to everyone working alongside me as part of the FoRester project PCIF/SSI/0102/2017. Thank you for allowing me to contribute to your endeavors and creating something that truly makes people's lives better.

A monumental thank you to my better half Renata Surrécio, whose photography skills and will to stand in the sun for hours with me made the extensive testing possible, and whose unshakable faith in me kept me going through many sleepless nights and exhausting days.

Lastly, a warm thank you to my family for being with me every step of the way. Everything I accomplished during my time at this institution is because of the opportunity I was given by them.

Abstract

With the increased effects of climate change, extreme weather is becoming increasingly frequent. Seasons of low humidity and elevated temperatures, in conjunction with growing encroachment of urban populations upon rural and forest areas, have increased both the frequency and severity of wildfires. When a wildfire occurs, massive amounts of equipment and personnel are deployed to combat it on the front lines, putting firefighters in dangerous, potentially fatal, situations where they deal with flames and reduced air quality, due to low oxygen and toxic gases. Even after the fires are mostly put out there is still the risk of reignition due to leftover embers, which if left unattended or unnoticed can cripple the firefighting efforts due to new fronts being created.

In order to assist with these issues, two compact sensor modules for vehicle application were created. The first module is the Thermal Mapping consists of a fire detection system using a multipixel infrared sensor linked to a small screen to provide real-time thermal imaging and enabling a user to better detect hotspots and hidden flames even at large distances. The second module is the Alarm Module, which is installed in a vehicle and monitors the oxygen levels as well as its position via GPS and transmits them both to the user through Bluetooth; this ensures that when conditions grow too dangerous the responders are alerted immediately in order to retreat and that rescuers can find them easily if needed.

Keywords: Wildfire Prevention, Infrared Thermal Sensor, CO Sensor, Oxygen Sensor, GNSS Receiver, Arduino, ESP32, Bluetooth Low Energy,

Resumo

Com o aumento dos efeitos das mudanças climáticas, as condições meteorológicas extremas são cada vez mais frequentes. Períodos de baixa humidade e temperaturas elevadas em conjunto com a crescente expansão de populações urbanas para áreas rurais e florestais aumentaram a frequência e a intensidade dos incêndios florestais. Quando ocorre um incêndio florestal, há uma grande mobilização de equipamento e pessoal para o combater, colocando os bombeiros em situações perigosas e potencialmente fatais devido a chamas e qualidade do ar reduzida por escasso oxigênio e elevada quantidade de gases tóxicos. Mesmo que os incêndios sejam controlados ainda há o risco de reaparecimento de chamas devido a brasas e cinza, que se deixadas por apagar por completo ou não forem detetadas podem inviabilizar os esforços de combate a incêndios devido à criação de novas frentes.

A fim de ajudar nestas questões, foram criados dois módulos de sensores compactos para aplicação em veículos. O primeiro módulo é de Mapeamento Térmico, que consiste num sistema de deteção de incêndio utilizando um sensor infravermelho multipixel ligado a um pequeno ecrã para permitir a visualização da imagem térmica em tempo real e permitir ao utilizador detetar melhor os pontos quentes e as chamas escondidas, mesmo a grandes distâncias. O segundo módulo é o Módulo de Alarme, que é instalado num veículo e monitoriza os níveis de oxigénio bem como a sua posição via GPS e transmite ambos ao utilizador através de Bluetooth; isto assegura que quando as condições se tornam demasiado perigosas os respondedores são imediatamente alertados para recuar e que os socorristas podem encontrá-los facilmente se necessário.

Palavras-chave: Prevenção de Incêndios Florestais, Sensor Térmico Infravermelho, Sensor de Monóxido de Carbono, Sensor de Oxigénio, Recetor GNSS, Arduino, ESP32, Bluetooth Low Energy.

Table of Contents

1. INTRODUCTION 1

1.1 - CONTEXT AND MOTIVATION 1

1.2 - TARGET GOALS AND PROBLEM APPROACH 2

1.3 - MAIN CONTRIBUTIONS 2

1.4 - OUTLINE 3

2. TECHNOLOGIES USED 5

2.1 - THERMAL RADIATION 5

2.2 - TYPES OF SENSORS 6

2.3 - IR SENSORS 7

2.4 - O_2 SENSOR 8

2.4.1 - Electrochemical Oxygen Sensors 8

2.4.2 - Zirconia Oxygen Sensors 10

2.4.3 - Choosing the sensor 11

2.5 - CO DETECTOR 11

2.5.1 - Metal Oxide detectors 11

2.5.2 - Electrochemical detectors 12

2.5.3 - Infrared detectors 12

2.5.4 - Choosing a CO detector 13

2.6 - MICROCONTROLLER UNIT (MCU) 14

2.7-GNSS RECEIVER: 14

2.7.1- Satellite Constellation 14

3. ALARM MODULE AND THERMAL MAPPING MODULE 19

3.1 - THE ALARM MODULE 19

3.1.1 - Initial state 19

3.1.2 - Areas for improvement 20

3.1.3 - New Alarm module design 20

3.2 - THERMAL MAPPING MODULE 24

3.2.1 - Initial state 24

3.2.2 - Areas for improvement 25

3.2.3 - New Thermal mapping module design 26

3.3 - COMMUNICATION 27

3.3.1 - Bluetooth Low Energy 28

3.3.2 - Usage in the project 28

4. PROJECT DEVELOPMENT 29

4.1 - THERMAL MAPPING MODULE 29
4.1.1-Short range testing *31*
4.1.2- New Module implementation *32*
4.1.3-Field tests *39*
4.1.4 - BLE communication *47*
4.1.5- Final notes and module assessment *49*
4.2- ALARM MODULE 50
4.2.1- Oxygen Sensor *50*
4.2.2- GNSS Receiver *54*
4.2.3- BLE communication *58*
4.3-BILL OF MATERIALS: 61
5. CONCLUSION AND FUTURE WORK 62
5.1- CONCLUSION 62
5.2- FUTURE WORK 63
I. ANNEX 1- CODING 69

List of Figures

FIGURE 1- IR THERMAL SENSOR COMPONENTS (SOURCE: [2]) 7
FIGURE 2 - ELECTROCHEMICAL SENSOR (SOURCE : [13]) 9
FIGURE 3-SENSING CELL (SOURCE: [13]) 10
FIGURE 4 - COMPARISON OF SENSOR TECHNOLOGIES FOR DOMESTIC APPLICATIONS (SOURCE: [20]) 13
FIGURE 5 - THREE SATELLITES TRILATERATION WITHOUT DELAY (A) AND WITH AN UNEXPECTED DELAY (B) 16
FIGURE 6- GNSS RECEIVER BLOCK DIAGRAM (SOURCE: [29]) 16
FIGURE 7- FRONT END STRUCTURE OF A GNSS RECEIVER (SOURCE: [30]) 17
FIGURE 8-GENERIC DIAGRAM OF A SINGLE CHANNEL IN THE BASEBAND PROCESSING BLOCK (SOURCE: [31]) 18
FIGURE 9 - INITIAL VERSION OF THE ALARM MODULE (SOURCE: [40]) 20
FIGURE 10 - ALARM MODULE BLOCK DIAGRAM 21
FIGURE 11-SENSOR CIRCUITRY (SOURCE: [..... 21
FIGURE 12- CO CONCENTRATION CHART (SOURCE: [46]) 22
FIGURE 13-CO SENSOR CONTROL CIRCUIT 23
FIGURE 14 - ALARM MODULE SCHEMATIC 24
FIGURE 15 - INITIAL SETUP OF THE THERMAL MAPPING MODULE (SOURCE: [40]) 24
FIGURE 16 - THERMAL MAPPING MODULE BLOCK DIAGRAM 25
FIGURE 17 - ILI9341 BLOCK DIAGRAM (SOURCE: [47]) 26
FIGURE 18 - THERMAL MAPPING MODULE SCHEMATIC 27
FIGURE 19 - INITIAL THERMAL MAPPING CODE 30
FIGURE 20 - NATURAL LIGHT READING WITH NO FLAME TARGET 30
FIGURE 21 - TOP LEFT: 3 METERS; TOP RIGHT: 5 METERS; BOTTOM: 7 METERS 31
FIGURE 22-FIRST PART OF THE SETUP 32
FIGURE 23-MLX90640_I2CWRITE DECLARATION 32
FIGURE 24-DISPLAY OUTLINE CODE 33
FIGURE 25-COLOR SCALE CODE 34
FIGURE 26-BLE COMMUNICATION SETUP 34
FIGURE 27-BLE SERVICE STARTUP 34
FIGURE 28 - FRAME READING FUNCTION 35
FIGURE 29 - SENSOR READING FUNCTION 36
FIGURE 30 - IMAGE DRAWING FUNCTION 36
FIGURE 31 - MLX90640 REFRESH RATE 36
FIGURE 32 - COLOR TIERS AND CALCULATION 37
FIGURE 33 - BLE INCLUDES AND UUID'S 37
FIGURE 34 - MODULE CONNECTION FUNCTION 38
FIGURE 35 - BLE COMMUNICATION IN THE MAIN LOOP 38
FIGURE 36 - FIRE BURNING 40
FIGURE 37 - MEASUREMENT OF THE EMBERS (295.7ºC) 40
FIGURE 38 - 20M TEST SETUP (FLAME IS ON THE TOP RIGHT, BEHIND THE WOOD RAILING) 41
FIGURE 39 - READINGS AT 20M. TOP LEFT- EMBERS; OTHERS- FLAMES BURNING 41
FIGURE 40 - UNLIT SOURCE AND READING 42
FIGURE 41 - FLAME TEMPERATURE READINGS 43

FIGURE 42 - TESTING ENVIRONMENT 44
FIGURE 43 - READINGS WITH (TOP)AND WITHOUT (BOTTOM) FLAMES ABOVE THE RIM OF THE PIT 45
FIGURE 44 - FLAME DETECTED IN THE CENTER-LEFT OF THE SCREEN 46
FIGURE 45 - ENVIRONMENT AND READINGS AT 26 METERS 47
FIGURE 46 - BLE SCANNER LOGO 47
FIGURE 47 - (A) AVAILABLE DEVICES 48
FIGURE 48 - VALUES BEING READ 49
FIGURE 49 - OXYGEN TEST SETUP 51
FIGURE 50 - INITIAL FLAME READINGS 52
FIGURE 51 - BREATH READINGS AFTER FLAME 52
FIGURE 52 - REFERENCE VOLTAGE AND DIGITAL PIN 53
FIGURE 53 - READ VOLTAGE FUNCTION 54
FIGURE 54 - OXYGEN PERCENTAGE CALCULATION 54
FIGURE 55 - DIFFERENCE OF ALMOST 30 METERS IN LITTLE OVER 1 MINUTE 55
FIGURE 56 - LIBRARIES INCLUDED 56
FIGURE 57 - GPS VARIABLE DECLARATIONS 56
FIGURE 58 - MAIN LOOP OF THE ALARM MODULE 57
FIGURE 59 - DISPLAY GPS INFO 57
FIGURE 60-DISPLAY DATE AND TIME 58
FIGURE 61-GET OXYGEN CONCENTRATION 58
FIGURE 62 - BLE COMMUNICATION FOR THE ALARM MODULE 59
FIGURE 63-ALARM MODULE AVAILABLE AND TRANSMITTED DATA 60

List of Tables

TABLE 1 - THERMAL MAPPING MODULE BOM 61
TABLE 2 - ALARM MODULE BOM 61

1

1. Introduction

1.1 - Context and Motivation

The threat of wildfires has been steadily increasing throughout each year. As temperatures continue to rise and droughts become more frequent, the number and severity of wildfires grow as well, posing an existential threat to our way of life. As such, new and bolder measures are being taken to combat wildfires.

In 2018, after the wildfire of Monchique, it was reported by responders that management manpower and equipment was disastrous [3]. Poor management of resources can have dire results; the speed of response to a call for help by responders who find themselves in a dangerous situation can be the difference between life and death.

Additionally, the prevention of reignitions makes the wildfire suppression efforts as efficient as possible. Crews needing to be dispatched to an area multiple times slows down the control process and prevents them from focusing their attention on critical areas.

This work will serve as a continuation of an ongoing project developed recently by students of FCT's Electrical Engineering department. The original project sought to create an alarm module to assist in fighting wildfires. The module would detect areas where there is danger due to flames, measure the air quality inside the vehicle, and transmit the information to both the firefighters on the field and an online centralized system in order to better coordinate the fire suppression efforts.

1.2 - Target Goals and Problem Approach

The main objective of this thesis is to design and implement a practical version of the previous work. This means developing sensor modules that provide information on the infrared emissions of the surrounding environment, as well as the quality of the air inside the vehicle, but that allow the user to communicate with the central network through an internet connection. This means implementing communication through Bluetooth, rather than using a softAPI, in order to maintain a smartphone's mobile data connection.

In order to make the system easier to integrate into a vehicle, the original module's functions were split into two separate systems. The first module, which uses IR thermal sensors, will no longer be used to continuously monitor for danger, instead focusing on the detection of reignition spots. It will show the information in a streamlined, easy-to-grasp format, rather than using a value array, each one of which needs to be individually analyzed.

The second module will monitor air quality and display GPS coordinates, serving as an alarm module. It will inform the user of the oxygen concentration in the air, as well as the presence of dangerous levels of carbon monoxide.

1.3 - Main Contributions

This work will see a significant improvement in the development of the wildfire suppression assistance system. It will ensure the safety of firefighters as well as a swifter way of scouting the terrain and discovering areas of danger through IR scanning.

The implementation of Bluetooth communication will allow for smoother communication between responders and the personnel coordinating the overall firefighting effort, drastically shortening the delay between a distress call and the arrival of support, as it enables responders to maintain simultaneous and reliable communication with the sensor modules and the dispatch center.

Making the scanning and thermal mapping simpler means less attention needs to be given to each specific area or reading, thus expediting the follow-up patrols and ensuring no hotspots or reignition sources go undetected, even if they are at a distance as long as 50 meters of the scouter.

1.4 - Outline

This thesis is divided into 5 sections, including this first introductory one. The next section will describe the technologies used in the project, from the types of sensor to the microcontroller. The third section describes the initial state of the module at the start of the project and how it was reworked to better fulfill the project goals. The fourth section describes in detail how the new modules were implemented and tested. This includes all software used as well as an explanation of the calibration and choices made in implementation.

2

2. Technologies Used

This chapter consists of a review of the current state of the technology utilized in this project. In researching these topics, seeing as this project builds upon an ongoing work, the references cited regarding the components that are already implemented are the same.

This chapter aims to provide the basic concepts regarding energy transmission through thermal radiation, how the IR thermal sensors, O_2 sensors, and CO sensors obtain their readings, which type of sensors are available and which are more appropriate for usage in this project. Additionally, this chapter presents a summarized explanation the GNSS receiver, Synthetic Aperture Radar, and Microcontroller Unit; this last one has several options presented despite the selection of the board being a moot point in the context of this project since firmware for It has already been developed.

2.1 - Thermal Radiation

Thermal Radiation Thermal radiation is electromagnetic radiation that is emitted by a body as a result of its temperature. All objects with a temperature above absolute zero emit thermal radiation in a spectrum of wavelengths [1]. Most of this radiation is in the infrared (IR) region of the electromagnetic spectrum, but some also spread into the visible light range [2].

When an object is exposed to radiation, the energy can be divided into three categories: reflected, transmitted, and absorbed. For each of these components, any given object has a coefficient which determines how much of each is produced from the incident radiation [2].

These coefficients are determined by the objects' composition. Of the three components, absorbed radiation is the one that causes an object's temperature to increase. [2]

The basis for the IR thermal sensor is all bodies emit radiation energy, which conforms to the following equation [3]:

$$P = e\sigma A(T^4_{obj} - T^4_{amb}) \quad (2.1)$$

where:

P = energy radiated;

e = emissivity;

σ = Stefan-Boltzmann constant = 5.6703×10^{-8} W/m^{2}K^4;

T_{obj} = Object temperature;

T_{amb} = Environment temperature;

The value of "e" has a minimum of 0 and a maximum of 1 (known as a black body). The higher the value of e, the better emitter the object is.

This equation describes how easy it will be to detect hotspots. A warmer environment will result in increased T_{amb} and therefore a lower level of emitted infrared energy.

2.2 - Types of Sensors

We will first differentiate between the two types of sensors. An active sensor is used to measure signals transmitted by the sensor which are reflected back to it by the target, whereas a passive sensor simply responds to stimuli from the environment [5]. As such, active sensors require their own power source to operate.

Active sensors operate using frequencies below 35GHz and thus are less affected by weather factors, which facilitates long distance detection [6].

Passive sensors operate above the 35GHz range and are more suited for short and medium range.

2.3 - IR Sensors

An IR thermal sensor measures the temperature of an object by receiving its radiated energy [7]. As a body is exposed to radiant energy, this energy is partially absorbed, and the rest either passes through the body or is reflected [8]. Reflected energy from the body returns mostly as infrared (IR) waves. These waves are received as input by the sensor, which converts the IR wave signal into an electric signal [2].

The sensor receives the energy through a lens to focus and filter the IR waves in order to obtain the desired range. IR thermometers use wavelengths between 14μm and 7 μm, due to it being the bandwidth with strongest radiation and lowest atmospheric attenuation [2].

Afterwards, an IR detector converts the now-filtered energy it into the electric signal. This signal has an amplitude of microvolts, therefore a high-gain amplifier is required to increase its power, as well as a signal-conditioning module to stabilize the values and linearize the signal. Finally, an Analog-to-Digital Converter (ADC) is placed to convert the electric signal into digital readable data. In Figure 1 we can see how the modules are connected.

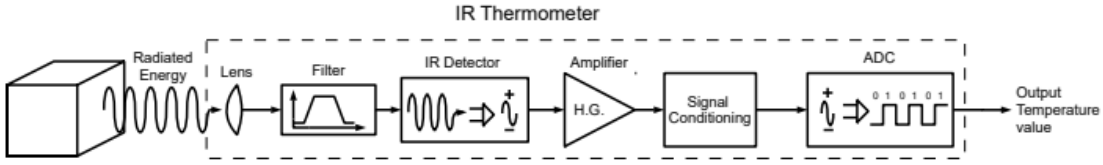


Figure 1- IR thermal sensor components (source: [2])

The IR detector works with either single or double wavelength range. The former measures the energy in a certain moment, whereas double wavelength makes use of two different readings with different ranges, thus providing increased accuracy. In this case, the value read would be the average between the two readings [2], [4].

Sensors can also differ in terms of range. A sensor with narrow range is used in measuring substantial variations in emissivity, which can occur when conditions such as smoke are obscuring the target. Sensors of wider range have reduced accuracy but boast greater range of measured temperatures [2].

There are two types of infrared detectors: photodiodes and thermal detectors.

A photodiode is a diode which generates current by releasing electrons when it is hit by photons from a light source. If the light it receives is more intense, the electron flow will be greater [2],[9].

A thermal detector's properties depend on the incident radiation [2]. Three types of thermal detector exist:

- **Thermopile:** Consisting of two thermocouples which are either in series or in parallel, it generates electrical energy from thermal energy. It is used when one thermocouple's output is too small [10];
- **Bolometer:** Measures radiated energy with a varying resistor [11];
- **Pyroelectric**: Generates electric potential when its temperature varies [12].

2.4 - O_2 Sensor

An oxygen (O_2) sensor is a device which allows a user to determine the ratio of oxygen in a certain environment. A few examples of applications are automotive industry, industrial safety, medical facilities and modified atmospheric packaging. Each application or environment uses the best suited among different types of sensor technology [11]. Most oxygen sensors quantify oxygen levels in gas or liquid with the use of one of three technologies: electrochemical, zirconia, or optical [12].

Each type of oxygen sensor has a different method for obtaining its measurements. In this section we will be looking at the two of technologies previously mentioned: Electrochemical and Zirconia. This is due to Optical oxygen sensors not being fit to use in this context.

2.4.1 - Electrochemical Oxygen Sensors

Electrochemical oxygen sensors are mainly used to measure oxygen levels in ambient air. They quantify a chemical reaction within the sensor, which generates an electrical output in proportion to the oxygen level. Because certain electrochemical sensors generate their analog current, they can be self-powered [12]. The ambient temperature electrochemical sensor (usually called galvanic sensor) is a partially sealed cylindrical device which contains two electrodes (dissimilar) immersed in an electrolyte, commonly a water solution of potassium hydroxide [13].

The ambient temperature electrochemical sensor (usually called galvanic sensor) is a partially sealed cylindrical device which contains two electrodes (dissimilar) immersed in an electrolyte, commonly a water solution of potassium hydroxide, as seen in Figure 2 [13].

Figure 2 - Electrochemical Sensor (source : [13])

Due to the usage of potassium, they are referred to as PO_2 sensors in some fields. These sensors operate like a metal/air battery; O_2 gets dissociated and reduced at the cathode to hydroxyl ions. Those pass through the electrolyte and oxidize the metal anode. A current, which is proportional to the rate of O_2 consumption, is generated when the cathode and anode are electrically loaded with a resistor (typically between 50 and 300Ω). This process can be described as the following reactions [14]:

Cathode reaction:

$$O_2 + 2H_2O + 4e^- \leftrightarrow 4OH^- \quad (2.2)$$

Anode reaction:

$$2Pb + 4OH^- \leftrightarrow 2PbO + 2H_2O + 4e^- \quad (2.3)$$

Overall cell reaction:

$$2Pb + O_2 \leftrightarrow 2PbO \quad (2.4)$$

2.4.2 - Zirconia Oxygen Sensors

This sensor (also known as the “high temperature” electrochemical sensor) is based on application of the Nernst principle [15].

$$\Delta V = \frac{k_B T}{e_0} \times \ln\left(\frac{c_1}{c_2}\right) \tag{2.5}$$

k_B = Boltzmann constant ($k_B = 1.38\times10^{-23}$J/K)

T = Temperature, in K

e_0 = Elementary charge ($e_0 = 1.602\times10\text{-}^{19}$C)

c_i = Ion concentration in mol/kg

These sensors are based on a solid-state electrolyte made of zirconium oxide stabilized with yttrium oxide. This probe is plated on opposing sides with platinum - these become the sensor electrodes. At the outer surfaces, there are two further platinum rings which along with the center platinum ring, provide the electrical connections to the actual cell. Two outer alumina (Al_2O_3) discs filter and prevent any ambient particulate matter from entering the sensor and also remove any "un-burned" gas concentrations. This prevents contamination of the cell which may lead to unstable measurement readings. Overall, the Zirconia Oxygen Sensor System's cell assembly is surrounded by a heater coil which produces the necessary 700°C required for operation, as seen in Figure 3 [16].

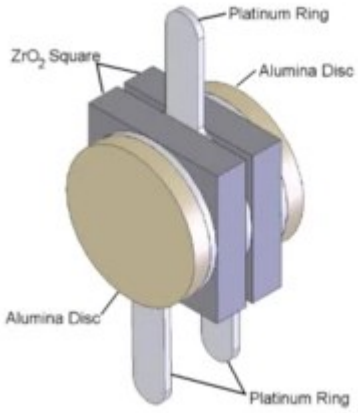


Figure 3-Sensing cell (source: [13])

This operating temperature makes the zirconium lattice becomes porous at a molecular level. This change makes It possible for oxygen Ions to move from a higher to a lower

concentration, based on the partial pressure of oxygen. The gap in partial pressure is achieved by exposing one electrode to air (21 % oxygen) and the other one to the sample gas. A voltage between the two electrodes (whose magnitude is a function of the oxygen partial pressure differential created by the reference gas and sample gas) is thus created by the movement of oxygen Ions across the zirconium oxide [13].

2.4.3 - Choosing the sensor

Considering the previous two technologies, deciding which type to use in this case would be difficult if not for availability and energy factors. These types of sensors have similar characteristics in many aspects, a zirconia sensor boasts a better response time, the technology currently available that both uses this method and is compatible with an Arduino board requires the usage of a component known as a lambda shield as well as an external power source between 10 and 24V [17]. As such, it becomes impossible to utilize the zirconia sensors with the rest of the modules, and so the electrochemical sensor will be chosen.

2.5 - CO Detector

Carbon monoxide (CO) is a flammable gas without color, odor or taste and with a slightly lower density than air. It is toxic in concentrations above about 35 ppm (parts per million). A CO detector is a device that detects CO presence, thus pre-venting poisoning from that gas. A gas detector/sensor or a vapor detector, com-pliant with the above requirements, consists of an assembly of electrical components associated with a sensing means inside a chamber, or by separate components to detect toxic and/or combustible gases or vapors. The detector includes provision for the connection to a power source and signaling circuits [18].

Much like the Oxygen sensor, there are many different technologies available for the measuring of CO. We will be taking a look at three of the most commonly used methods for measuring carbon monoxide: Metal Oxide detectors, Electrochemical detectors, and Infrared detectors.

2.5.1 - Metal Oxide detectors

Metal Oxide detectors, also known as semiconductor detectors, function on the principle that exposition to CO affects the electrical resistance of metal oxides, which changes according to CO concentration. So, a measure of the resistance change of the material provides a measure of CO level. This is called a semiconductor sensor since conductivity is between a conductor and an insulator [19].

In metal oxide sensors, CO detection results from the reaction between CO and adsorbed oxygen. Adsorption of atmospheric oxygen occurs on the oxide surface, resulting in the formation of oxygen ions at the surface level by trapping electrons from the metal oxide. Those ions repel electrons in the bulk, which results in a field of reduced electrons, causing an increase in resistance. Adsorption of CO on the surface will then lead to a chemical reaction with the oxygen ions, forming CO2 and reducing sensor resistance by removal of the ions. This resistance reduction is measured electronically, providing a measure of the CO level [20]. One notable characteristic of the semiconductor sensor is the need for a considerable energy input for stabilization and a pulsing heating system for continuous function. This is due to high heat (300 – 400ºC) being required to purge the moisture and contaminants, and because this type of sensor is most sensitive between 80 and 100ºC respectively [20].

2.5.2 - Electrochemical detectors

Electrochemical detectors work similarly to batteries. They have terminals named electrodes, made from metallic platinum, immersed into a chemical solution called an electrolyte. The electrolyte conducts electricity more readily in the presence of carbon monoxide, causing a current flow in a detector circuit, which is then used to trigger an alarm. [21]

CO enters the cell through a diffusion hole at the top. Once it reaches the top electrode, the catalytic effect of the platinum makes It react with water, producing carbon dioxide, ions and electrons. These are removed by the external circuit, causing the protons (positively charged) to flow through the electrolyte to the electrode (called counter electrode) and react with oxygen and electrons to form water. Reactions are as follows [20]:

At the sensing electrode:

$$CO + H_2O \rightarrow CO_2 + 2H^+ + 2e^- \quad (2.6)$$

At the counter electrode:

$$^1/_2O_2 + 2H+ + 2e^- \rightarrow H_2O \quad (2.7)$$

Overall:

$$CO + ^1/_2\, O_2 \rightarrow CO_2 \quad (2.8)$$

2.5.3 - Infrared detectors

Infrared detectors use NDIR, also known as nondispersive technology, to measure the gas contents in the air, detecting the presence of CO through a spectroscopic sensor. These sensors consist of four main components: an infrared source, a sample chamber, a light filter,

and an infrared detector. A fraction of the radiation from the source, travels to the reference detector without attenuation for com-parison, thus providing a reference [22].

The IR light is directed through the chamber towards the detector, causing ab-sorption of specific wavelengths; the attenuation of these particular wavelengths is measured by the detector, allowing gas concentration determination. This works well for CO measuring due to the gas' characteristic absorption of infrared light, at a wavelength of about 4.6μm [20].

2.5.4 - Choosing a CO detector

Figure 4 compares the three technologies previously surveyed, showing that the infra-red sensor is superior on all aspects except for cost [20].

Comparison of Sensor Technologies for Domestic Applications

Sensor Property	Sensor Technology			
	Colorimetric	Metal Semiconductor	Electrochemical	Infrared
Short-term stability	Difficult to assess	Fair	Good	Excellent
Lifetime	>5 years Data being collected	5–10 years	>5 years Data being collected	>5 years
Resolution	Fair	Fair	Good	Excellent
Immunity to false alarm	Fair	Good	Good	Excellent
Immunity to false negative	Good	Good	Good	Excellent
Immunity to poisoning	Good	Good	Good	Excellent
Humidity dependency	Fair	Fair	Good-to-excellent	Excellent
Temperature dependency	Fair	Fair	Fair	Excellent
Sensitivity drift	Unknown	Moderate	Moderate	Low
Response time	Fair	Fair	Good	Excellent
Selectivity	Good	Good	Good	Excellent
Power consumption	Low	High	Low	Medium
Cost of sensor	Low	Low	Low	High

Figure 4 - Comparison of Sensor Technologies for domestic applications (source: [20])

Cost is, however, one of the most important factors when considering components, so as to facilitate development and production. It is also necessary to assess the compatibility of the sensor with the module as a whole, so the options are limited to what can be integrated into the controller.

2.6 - Microcontroller unit (MCU)

An MCU is chip which functions as a computer with a specific purpose. An MCU contains [33, 34]:

- Central Processing Unit (CPU): Executes saved programs;
- Non-Volatile Memory (NVM): Holds data for the programs to be executed. Can be ROM, EEPROM or Flash [35];
- Internal Clock: Maintains synchronism;
- Random Access Memory (RAM): Unlike Non-Volatile Memory, cannot hold memory without a power supply. Is used to hold variables;
- Input/Output (I/O) pins: Serve as the interface between the MCU and the outside world in order to obtain commands or sensor data.

For this project, the ESP32 board was chosen, as it is compatible with the previous hardware selected, boasts a 2-core processor capable of handling multiple tasks simultaneously, is low in acquisition cost, and can be programmed using Arduino IDE as well as RTOS.

2.7-GNSS Receiver:

A GNSS receiver is a navigation device which calculates its own location. By receiving data from satellites in a constellation, the device is able to calculate its position on Earth by a process known as trilateration [23]. These concepts will be further explored in the following chapters.

2.7.1- Satellite Constellation

A satellite constellation consists of a group of satellites working in tandem to provide receivers with their positioning [24]. The most used global navigation constellations are:

- Global Positioning System (GPS):Created by the USA, it has 31 working satellites, with no less than 24 available 95% of the time [25]. Its readings are accurate up to 5 meters [26].
- Galileo: Created by the EU; when fully operational, it will have 24 satellites plus spares [27]. Its accuracy is around 1 meter [26].

2.7.1.1- Trilateration

To obtain a correct and trustworthy location, the receiver requires 4 satellites. Trilateration consists of utilizing data from all connected satellites to obtain the receiver's position.

When considering a two-dimensional location, 3 circular areas are sufficient to determine receiver's location. In this application, however, the altitude of the receiver must also be known. To determine it, an additional satellite is needed.

The sensor determines its distance to a satellite by using the data sent by each satellite. The data is transmitted through radio signals and is called Navigation Message (NAV) [26].

2.7.1.2- Navigation Message

All satellites transmit a constant stream of data sets: the NAV. The NAV holds different types of information, such as satellite position, atmospheric signal corrections, and an identifying signal specific to each satellite known as Pseudo-Random Noise code (PRN code) [26].

This last code allows the receiver to calculate its distance to the satellite, since it has a short enough period to be able to repeat itself many times per second. Both the receiver and the transmitter start the code simultaneously; when the receiver receives the code sent by the satellite, it will be with a certain delay [23, 26]. Since the signal's travel speed is known (Lightspeed), the receiver can calculate the distance to the specific satellite by multiplying this speed by the time delay. Knowing both the distance to the satellite and where it is located, it is possible to perform the Trilateration.

2.7.1.3- Differential GPS (DGPS)

Despite satellites sending atmospheric corrections, effects such as the deflection of signals in buildings, or the weather itself affecting signal velocity can result in further errors. For these types of errors, we make use of Differential GPS (DGPS) [23, 28]. In two-dimensional trilateration with 3 satellites, assuming no delays, the positioning circles relative to each satellite will cross at a single point, as shown in Figure 5(a). Should a satellite experience unexpected transmission delays, the circles will cross in various points, as per Figure 5(b).

The basis for correcting positional errors is obtaining the location of a known location: either a Static Base Location (SBL) or a Satellite Based Augmentation System (SBAS).

SBL stations are located on Earth and know their exact locations, and thus are able to correct errors by themselves by continuously requesting their location to satellites. They proceed to calculate the discrepancy between its own correct location and the one obtained by satellite.

SBAS functions the same way as SBL. It consists of a group of geostationary satellites which receive the average correction in their area sent by base stations.

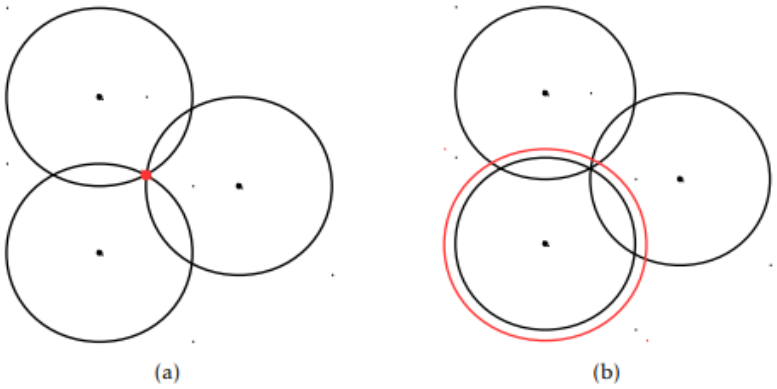


Figure 5 - Three satellites trilateration without delay (a) and with an unexpected delay (b)

2.7.1.4- GNSS Receiver Architecture

GNSS receivers comprise:

1. Antenna;
2. Radio Frequency (RF) Front End;
3. Baseband Processing module;
4. Application Processing module [29].

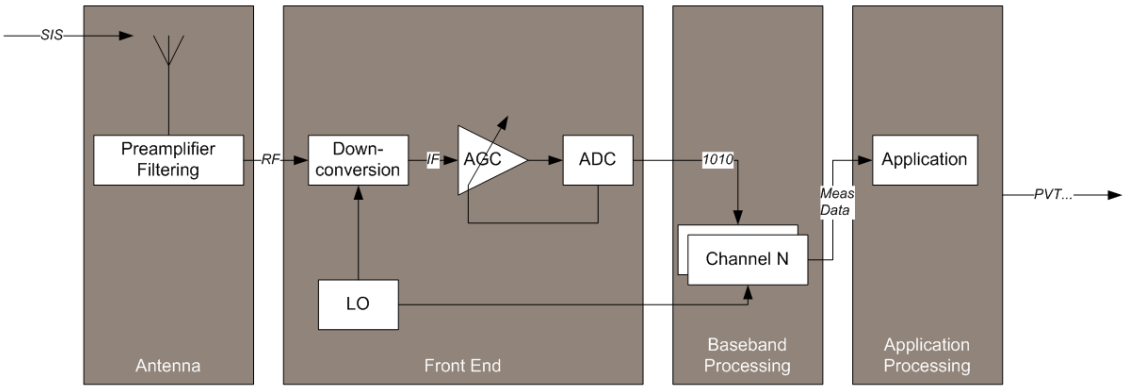


Figure 6- GNSS receiver block diagram (source: [29])

Antenna: Besides the antenna, this section is equipped with a Low Noise Amplifier (LNA). It receives a signal in RF, so the signal's electrical power Is reduced; hence the need for the amplifier in order to improve its reading [29];

RF Front End: The front end section receives the signal from the antenna and prepares it for subsequent processing; this goal can be achieved through several implementations. Time and frequency reference for all this section's components is provided by a frequency

synthesizer, shown in Figure 7. The referred components perform necessary steps to process and convert the RF signal to a base-band digital signal:

1. Filtering and amplification: performs noise reduction and out-of-band rejection in the received signals, as well as amplification stages that minimize the effect of transmission losses.
2. Down-conversion: This provides conversion of the signal to baseband, either through direct conversion techniques or through heterodyning approaches (multi-stage approaches which shift the spectrum to intermediate frequencies (IF) with the appropriate band filtering, resulting ultimately in the desired conversion to baseband.
3. Quantization: signal conversion by analog to digital converters (ADC), so that quantization errors and dynamic ranges are guaranteed to accommodate the signal's characteristics.
4. Automatic Gain Control: In this stage, the front end section gain is adjusted in order to use the full dynamic range. It has a close relation with the two previous steps [30].

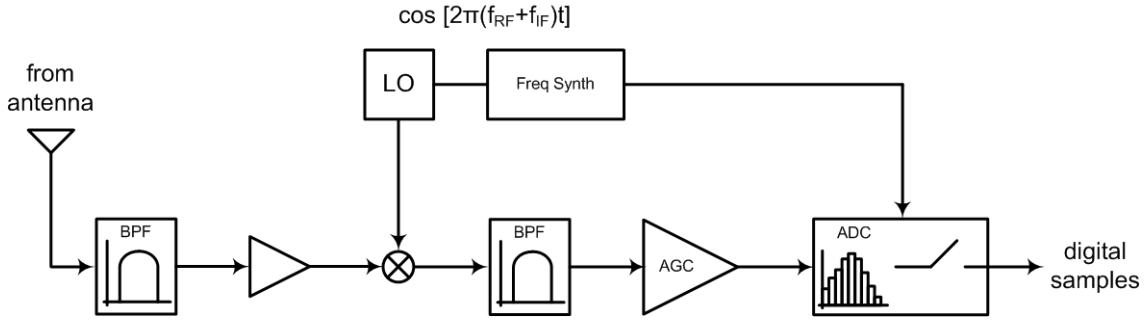


Figure 7- Front End Structure of a GNSS receiver (source: [30])

Baseband Processing: Receives each signal and tracks its code delay and carrier phase measurements [29]. The signal is initially fed to a Doppler frequency stripping module to account for the satellite's orbital movement, followed by a block to compare It with the code calculated by the receiver in order to confirm the correctness of both PRN code delay and Doppler frequency [31]. Afterwards, the signal enters a Delay Lock Loop (DLL), a Phase Lock Loop (PLL) and a Frequency Lock Loop (FLL) to verify and adjust the PRN delay, the carrier signal misalignment, and Doppler frequency misalignment produced by the signal generator [32]. This estimation is then returned to both the Doppler removal and the local PRN code generator blocks [31]. The incoming signal is passed through a demodulating and processing

block, partitioning the full message into smaller messages where information on the location of the satellite and atmospheric delays (among others) is included. [31];

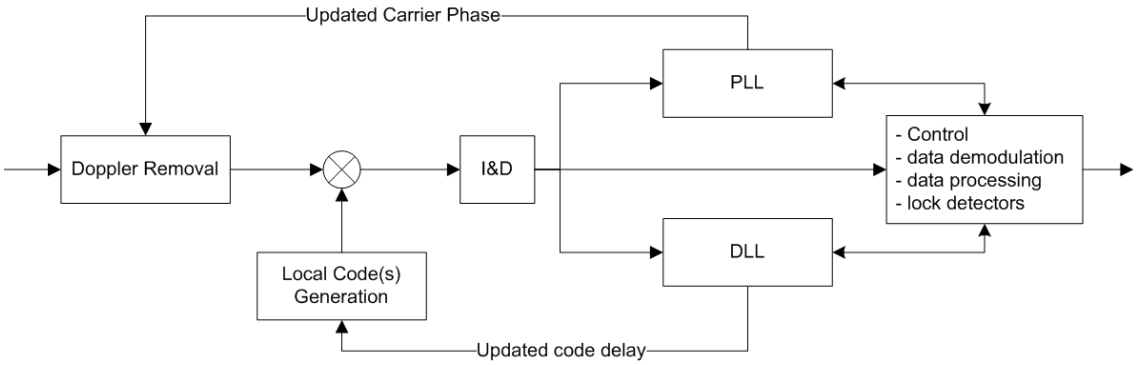


Figure 8-Generic diagram of a single channel in the Baseband Processing block (source: [31])

Application Processing: Extracts the message from the Baseband module to calculate information such as Position, Velocity and Time (PVT).

3

3. Alarm Module and Thermal Mapping Module

At the beginning of the project, it was first necessary to assess the state of the development and decide the needed improvements to models' design. The working version of the module was a single unit which performed the air quality measurements as well as the thermal mapping.

The first development stage consisted of splitting the initial module into two separate modules: one to serve as an alarm module and another to serve as a thermal mapping module. These would be used in tandem, with the former serving to detect dangers outside the vehicle as well as monitoring the conditions inside it, and the latter allowing for better detection of reignition spots that firefighters might miss when surveying the landscape.

3.1 - The Alarm Module

3.1.1 - Initial state

The first alarm module (Figure 9) featured four single-pixel IR thermal sensors (MLX90614), one ME2-O_2 oxygen sensor, one NEO-7m GPS chip, and a 0.96-inch OLED display. All of these components were connected to and controlled by an ESP32 development board. The four MLX90614 sensors serve to detect fires on all sides of the vehicle, whereas the ME2-O_2 sensor measures the air quality of the inside. Upon detecting dangerous conditions for the firefighters, the module will send an alarm as well as the vehicle's coordinates to a central server through a SoftAPI which could be accessed by a smartphone.

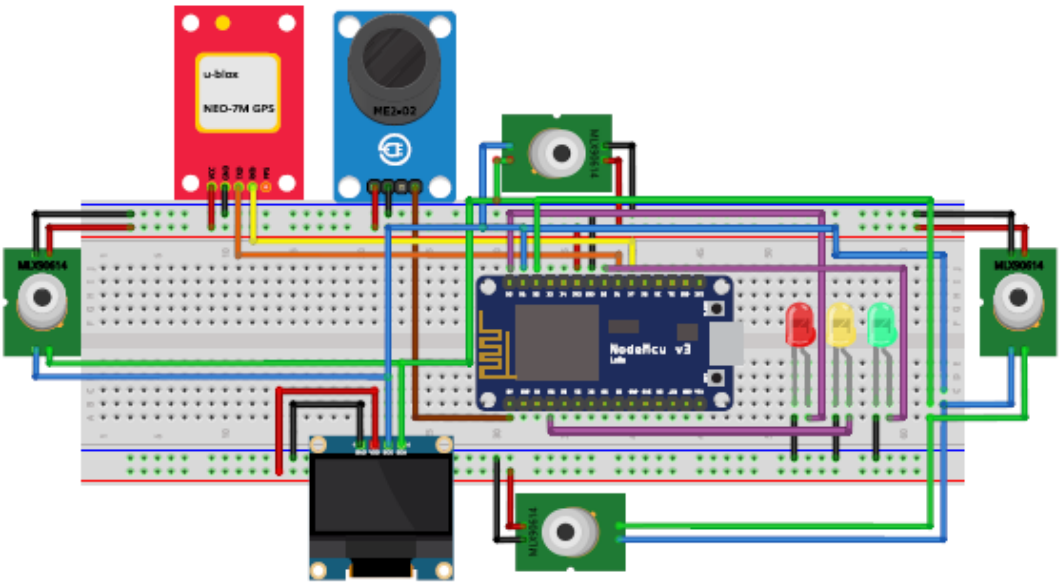


Figure 9 - Initial version of the alarm module (source: [40])

3.1.2 - Areas for improvement

This version of the module fulfills two important functions at once, which in theory is the best option, as a holistic approach means a more cost-effective solution to ensuring safety. However, this design creates a problem in its physical and practical implementation: simultaneously monitoring the inside and outside of the vehicle means the module needs to access both at once.

To accomplish this, it would be necessary to either adapt the vehicles to accommodate the module or make the module large enough to reach both sides. The first solution is costly and would require more extensive study of the vehicle's structure, while the second one means losing the sensor module's compact design and increase the risk of damage or deterioration due to the inclusion of flexible connecting cables. As neither solution is viable while keeping with the objectives of the project, a new design needed to be implemented.

3.1.3 - New Alarm module design

The first step of the new design process was redefining the function of the alarm module. It was decided that the monitoring of outside conditions would be scrapped in favor of exclusively monitoring the conditions inside the vehicle. This was due to the single-pixel sensors providing very little information compared to what the occupants of the vehicle could perceive on their own.

The new module would be installed on the inside of the vehicle to monitor the air quality inside of it. To provide additional information, it was decided that a carbon monoxide (CO) sensor would also be included in the alarm module, as carbon monoxide poses as great a threat to the responders' wellbeing as oxygen deprivation. The module provides the information to the user not through an OLED screen, but through a smartphone connected to the module. The communication methods are detailed in the end of the chapter, as this change was made to both modules.

With the module's features decided, a block diagram was created:

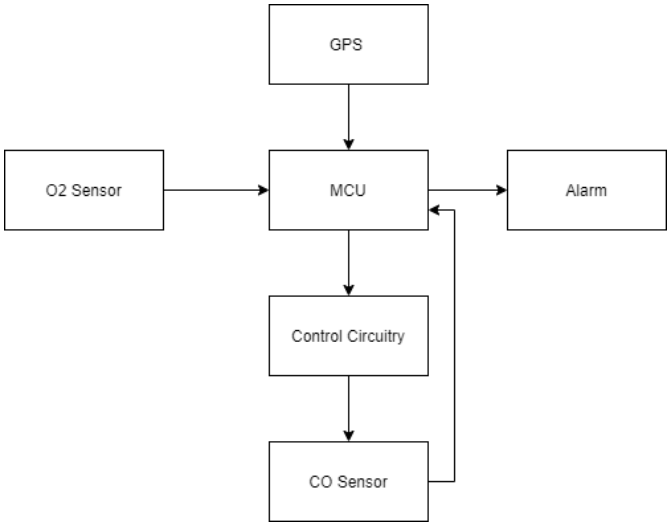


Figure 10 - Alarm module block diagram

Research conducted during the development of this module [41] showed the carbon monoxide sensor works in two stages: a low temperature stage which detects carbon monoxide by absorption, and a high temperature stage which clears out the gases absorbed during the previous stage. Normally, this variation is done by changing a potentiometer (R_L) connected to the sensor in order to change the current provided to it by the microcontroller.

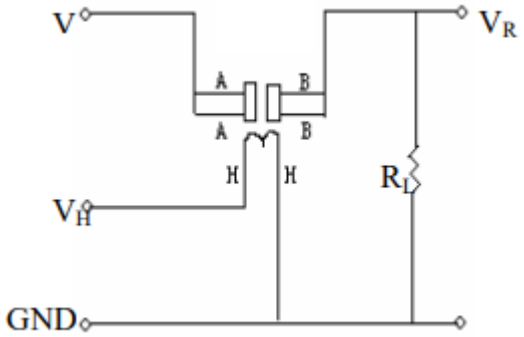


Figure 11-Sensor circuitry (source: [48]

The concentration read depends on the resistance ratio of the sensor (R_s/R_O), where $Rs=(Vc/V_{RL}-1)\times R_L$.

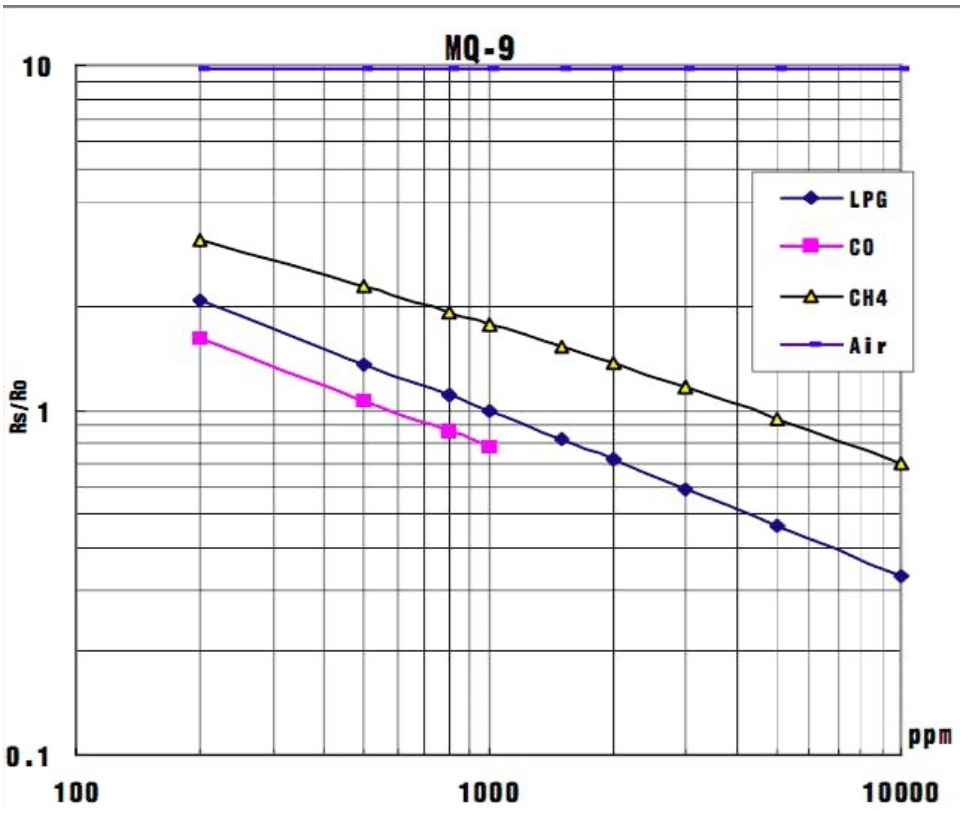


Figure 12- CO concentration chart (source: [46])

However, this needs to be done manually, and is therefore impossible if practicality of use is being considered. As such, is was decided that a control circuit (seen in Figure 13) would be added to adjust the voltage flowing into the sensor by controlling the output of one of the ESP-32 digital pins through pulse width modulation (PWM) with feedback.

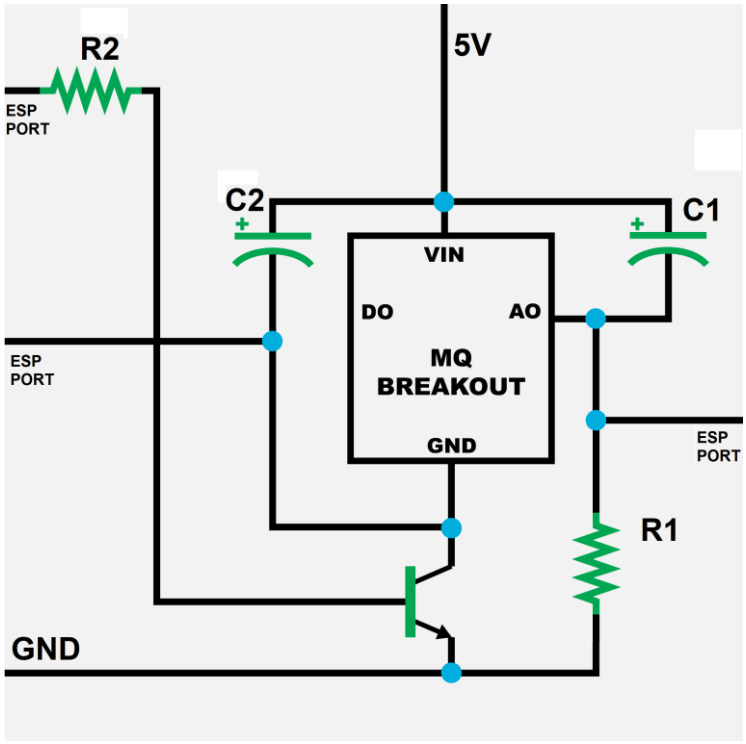


Figure 13-CO sensor control circuit

When the NPN transistor is constantly turned on, the voltage is 5V across the sensor, allowing for the heating phase. When we apply the PMW to the input the current pulses and is smoothed out by the capacitor, this keeping a stable lower voltage and allowing the reading cycle to take place.

As for the remaining components, both the GNSS receiver and the oxygen sensor (Grove ME2 O_2 sensor and NEO-7m GNSS receiver) require no modifications or additional circuitry, thus they can simply be connected to the microcontroller. As such, a new final schematic was designed:

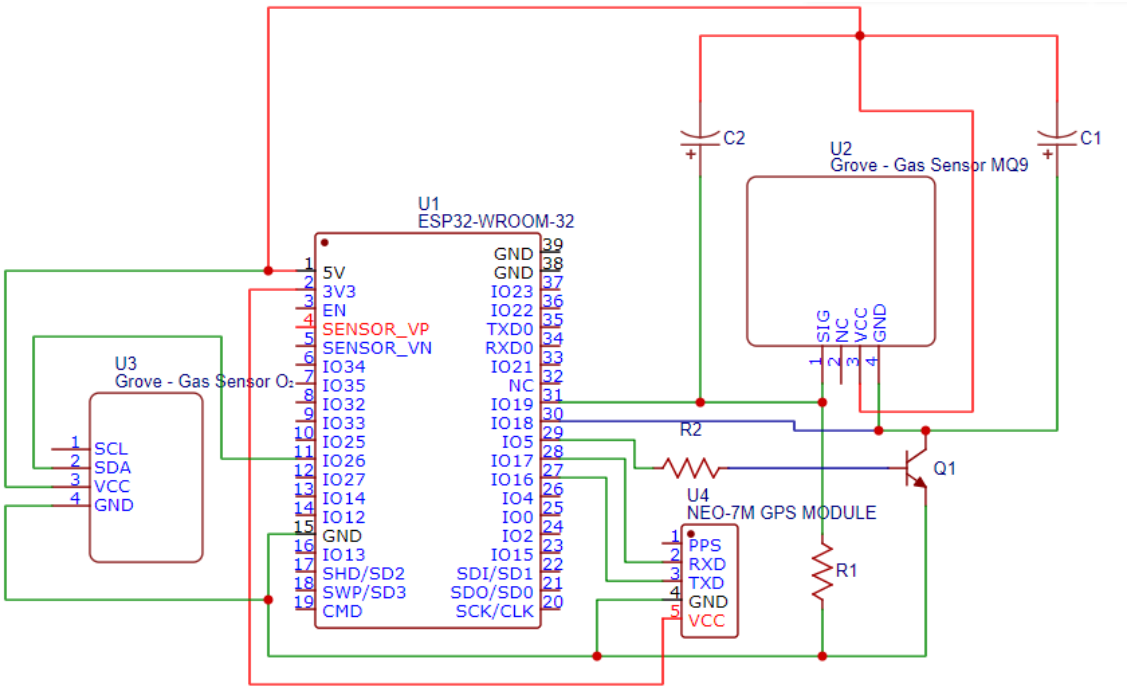


Figure 14 - Alarm module schematic

3.2 - Thermal mapping module

3.2.1 - Initial state

At the beginning of the project, the thermal mapping module captured a thermal image using the MLX90640 multi-pixel IR sensor, retrieved the current location using a GNSS receiver, and sent this information to the user in the form of a Json file through SoftAPI accessed via smartphone. Additionally, the OLED screen displayed certain messages when conditions were met.

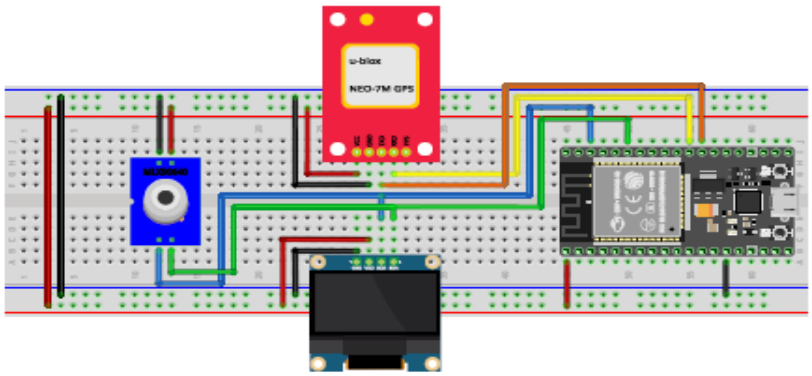


Figure 15 - Initial setup of the thermal mapping module (source: [40])

3.2.2 - Areas for improvement

The intended usage of this module is to assist fire suppression efforts by serving as a thermal camera which can detect hotspots that would go otherwise unnoticed by the naked eye. This function requires an active user to monitor the readings at all times, and the readings should be as simple to interpret as possible.

The most significant modification was changing the way the data is presented to the user. Sending the readings as a Json with each pixel's value written down makes it so each individual value needs to be read and interpreted by the user, which is time-consuming and inefficient. To expedite this process, it was decided that this module would no longer send the readings to the user through SoftAPI, instead making use of a screen to display a color-coded image corresponding to the values obtained from the sensor.

Additionally, the GNSS receiver was removed due to the information provided not being relevant to the module's usage. The thermal mapping module is to be used in low danger situations, therefore knowing the location of the user at all times is not a priority.

Finally, some values read by the IR sensor are sent to the user's smartphone, but the communication method has been changed, similarly to the Alarm module, and will be detailed at the end of the chapter.

With the changes decided, a block diagram was created:

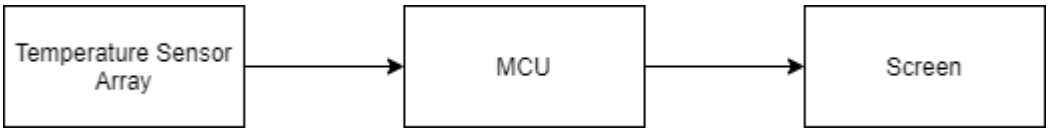


Figure 16 - Thermal mapping module block diagram

3.2.3 - New Thermal mapping module design

Having created the block diagram, new components needed to be chosen, specifically the screen to display the thermal image. The screen needed to have a high enough resolution and big enough size to display the image clearly while also being compatible with the ESP32 development board and of low cost.

The chosen display for implementation was the ILI9341 Thin Film transistor LCD. It has a resolution of 240x360 pixels, full RGB support, and can function with an 8-bit data bus MCU interface, making it compatible with the ESP32 development board.

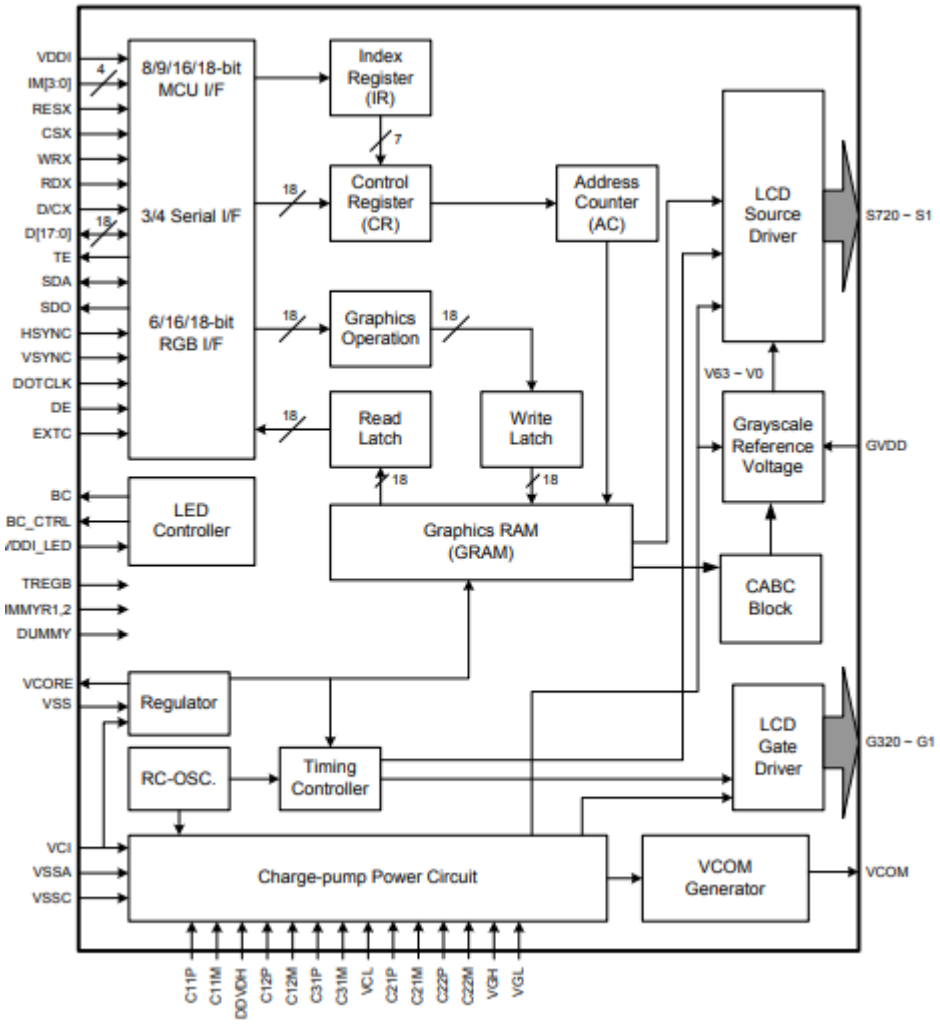


Figure 17 - ILI9341 block diagram (source: [47])

Seeing as there is no change to be made to the IR sensor, after selecting the display the thermal mapping schematic was created:

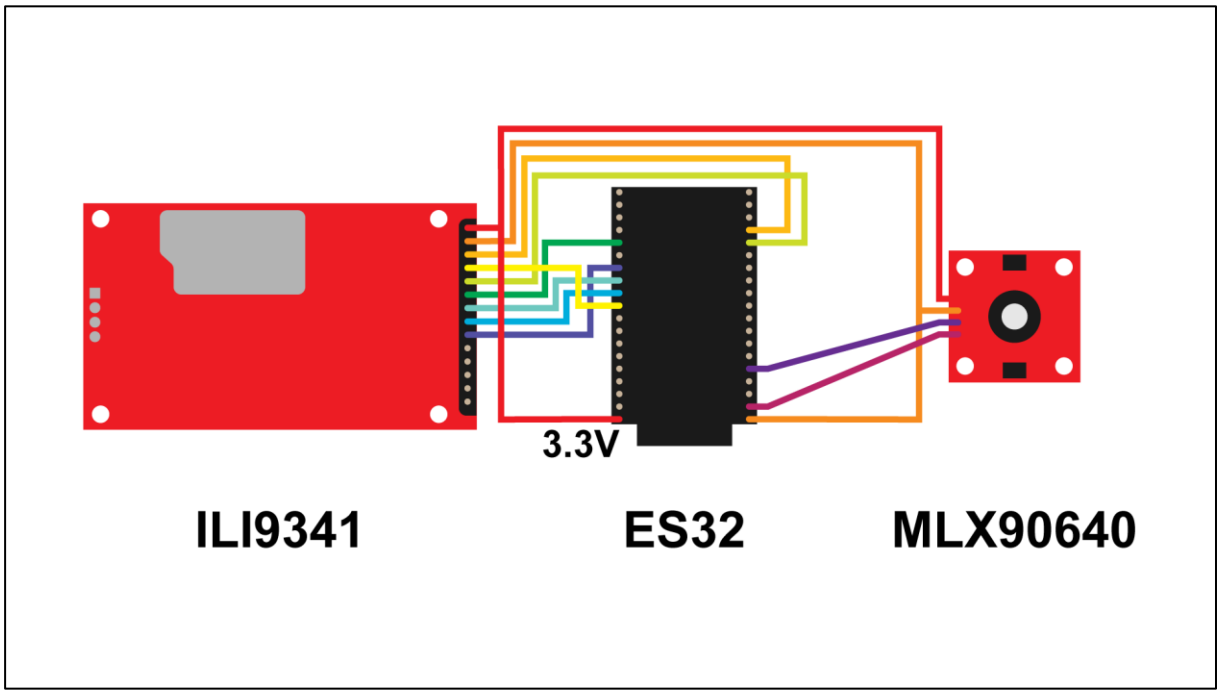


Figure 18 - Thermal mapping module schematic

3.3 - Communication

While the previous versions of each module featured a SoftAPI approach to transmitting data to the user, this method required the smartphone user to switch between connecting to the ESP32 to collect the values from the sensors and an actual internet connection to forward them to the central platform at ArcGIS. Not only does this make for tedious work, it also requires a dedicated user constantly focused on performing this operation which, particularly with the alarm module, means its autonomous capabilities are squandered.

It was therefore crucial to find another method of transmitting data. The first studied solution was having the ESP32 board itself connect to the internet and post information to a dedicated website address. This solution was deemed too costly, as it would require installing and maintaining a mobile wi-fi hotspot on each vehicle. The chosen approach would still be using the smartphone as a proxy to transmit information, but instead of using a SoftAPI connection, the smartphone would stay connected to the internet and connect to the ESP32 board through Bluetooth Low Energy (BLE).

3.3.1 - Bluetooth Low Energy

Bluetooth Low Energy (BLE) is a Bluetooth-based communication protocol and is better suited for low power IoT systems, as its packet capacity is limited. Devices using BLE can have two different functions: Central Device and Peripheral Device. Smartphones, tablets or computers mostly work as central devices and collect data from peripheral devices, which consist of sensors or other low power systems that connect to the central one.

When using BLE, data is sorted by Generic Attribute Profile (GATT). GATT is composed of one or more services, which in turn are made up of characteristics, which in this case are the sensor values themselves. Services and characteristics are identified by a 16-bit Unique Universal Identifier (UUID), which can be customized, though some types of characteristics have pre-assigned identifiers. [42]

3.3.2 - Usage in the project

To transmit the sensor values to the smartphone, various characteristics will be used simultaneously. Each characteristic will correspond to a value or condition; in the thermal mapping module the smartphone currently receives information regarding the highest temperature read as well as the temperature of the central pixel, and in the alarm module the smartphone receives the GPS coordinates for the current latitude and longitude, as well as presence of dangerous levels of CO or lack of Oxygen. Each of these characteristics will be of type "NOTIFY" which will therefore continuously update its value. There are other types of characteristics ("WRITE" and "READ" that in future developments can be used to control certain features of the modules or the smartphone, but in the current version only "NOTIFY" type characteristics are used).

At present, the communication through BLE between the modules is done through an app called BLE Scanner, as this was the most straightforward and easy to use method available. This app allows the smartphone to receive data corresponding to multiple characteristics at the same time, thus streamlining the monitoring process.

4

4. Project development

The first focus of the development was on the thermal mapping module, as this one required the most study, testing, and changes made to the hardware. After this module was completed, the work on the Alarm module began.

4.1 - Thermal Mapping Module

Development of this module started with studying the previously developed software, due to some reported problems with communication between the ESP32 and the computer, as well as testing the module itself to determine the accuracy of the readings. Unlike the rest of the software, which was developed in Arduino IDE, the first part of the analyzed software was developed in Visual Studio code. It collected the values read from the MLX90640 IR sensor and sent them through the serial port in the form of a matrix corresponding to each pixel, as well as values from 4 single-pixel IR sensors and a GNSS receiver. This version did not transmit these values as a Json, which for the initial analysis was not relevant, as the target was assessing the reading capabilities of the sensor and module.

The first error detected was in the first readings. Due to a subpage reading error, only the first subpage of the MLX90640 sensor was being read, resulting in every other pixel being marked as "0.00ºC". After the second reading, the missing sub-page would be read, giving the appearance of a complete image.

```
printf("Reading MLX90640");
int status = MLX90640_GetSubFrameData (slaveAddress, mlx90640Frame);
int page1 = MLX90640_GetSubPageNumber(mlx90640Frame);
status = MLX90640_GetSubFrameData (slaveAddress, mlx90640Frame);
int page2 = MLX90640_GetSubPageNumber(mlx90640Frame);
tr = MLX90640_GetTa(mlx90640Frame, &mlx90640) - TA_SHIFT;
MLX90640_CalculateTo(mlx90640Frame, &mlx90640, emissivity, tr, mlx90640To);
int x = 0;
printf("Thermal Image:\n\r");
//recolher a imagem térmica
for(int p=0;p< 24; p++){

        for(int q=0; q<32; q++){
            if(p>0 && p<23 && q>0 && q<31){
                image[x]=(mlx90640To[x]+mlx90640To[x+1]+mlx90640To[x-1]+mlx90640To[x+32]+mlx90640To[x-32])/5;
                printf("%.2f ", image[x]);
                x++;
            }else{
                image[x]=mlx90640To[x];
                x++;
            }

        }
        printf("\n\r");

}
printf("\n\r");
```

Figure 19 - Initial Thermal Mapping code

This image was, however, misleading; the readings for the two subpages would be out of sync with each other, resulting in a sort of chess effect where one set of pixels would be of the current read, while the other was of the previous one:

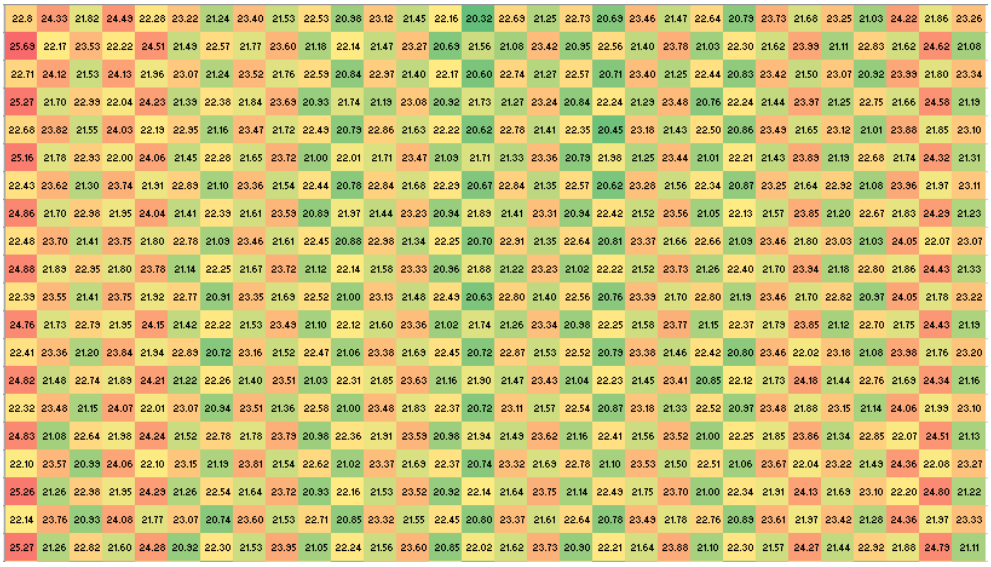


Figure 20 - Natural light reading with no flame target

This problem was corrected at a later date, as for the short range flame detection tests it was not a cause for concern.

4.1.1-Short range testing

The IR sensor was tested using a small alcohol burner, which achieved temperatures of about 180ºC, at a distance of 3, 5 and 7 meters. These tests would later prove to have little practical relevance due to the low temperature and small size of the flame compared to the field tests later performed but were nonetheless useful in determining problems with the firmware as well as some limits of the sensor.

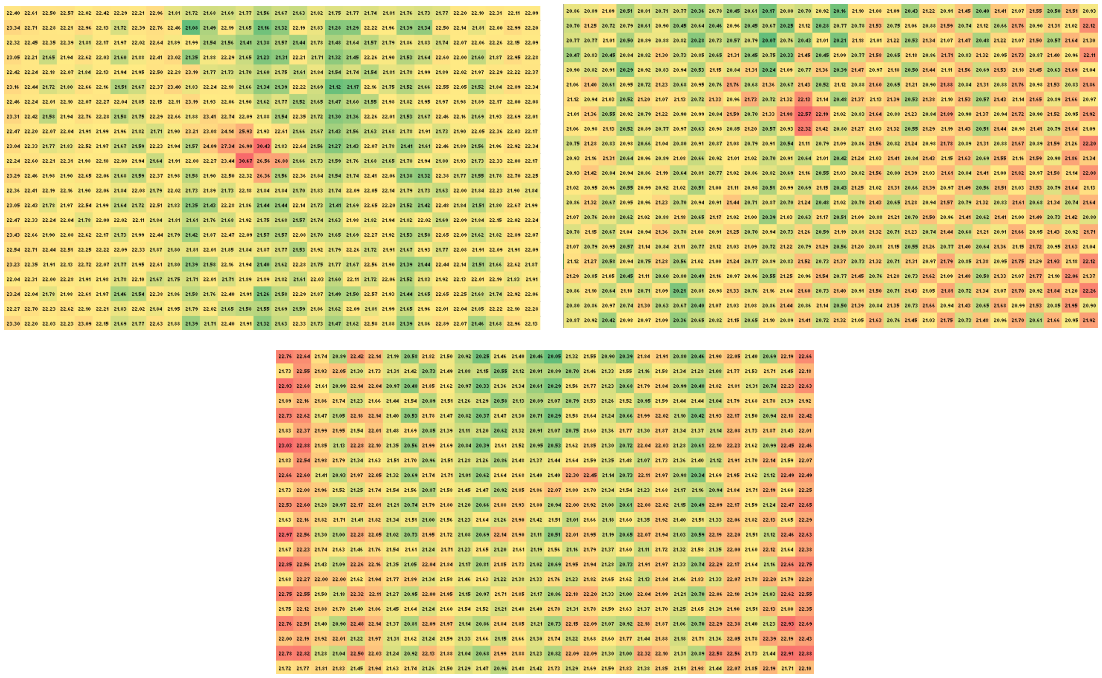


Figure 21 - Top left: 3 meters; Top right: 5 meters; Bottom: 7 meters

As can be observed in the previous images, the sensor's limit for a small flame is around 7 meters, where it blends in with environment readings. This limitation could possibly have been reduced through the correction of the subpage error, but the initial tests still proved the sensor's capability and allowed for an understanding of its functions, and thus it was decided that the time cost would not be worth the gains.

4.1.2- New Module implementation

The next phase of the project was assembling and programming a new version of the thermal mapping module. The previous version allowed for analysis only after copying the values sent to the serial port into an Excel table and conditionally formatting the cells, whereas this one would perform the transformation from values to image automatically.

The setup process in the software first starts up the serial communication, through which we receive messages indicating whether or not the module is working properly, then opens the communication between the microcontroller and both the OLED display and the sensor.

```
void setup()
   {
    Serial.begin(115200);
    Wire.begin();
    Wire.setClock(5000000); //speed at 5Mhz (max)
   if (isConnected() == false)
       {
        Serial.println("MLX90640 not detected at default I2C address. Please check wiring. Freezing.");
        while (1);
       }

    Serial.println("MLX90640 online!");

    //Get device parameters - We only have to do this once
    int status;
    uint16_t eeMLX90640[832];

    status = MLX90640_DumpEE(MLX90640_address, eeMLX90640);

    if (status != 0)
       Serial.println("Failed to load system parameters");

    status = MLX90640_ExtractParameters(eeMLX90640, &mlx90640);

    if (status != 0)
       {
        Serial.println("Parameter extraction failed");
        Serial.print(" status = ");
        Serial.println(status);
```

Figure 22-First part of the setup

Afterwards, we enable the reading of the values:

```
if (status != 0)
   {
    Serial.println("Parameter extraction failed");
    Serial.print(" status = ");
    Serial.println(status);
   }

//Once params are extracted, we can release eeMLX90640 array

MLX90640_I2CWrite(0x33, 0x800D, 6401);    // writes the value 1901 (HEX) = 6401 (DEC) in the register at position 0x800D to enable reading out the temperatures

MLX90640_SetRefreshRate(MLX90640_address, 0x05); //setting to 16Hz
```

Figure 23-MLX90640_I2CWrite declaration

Having concluded the setup for the IR sensor, the next phase is setting up the display. The following code draws the outline and frame of the display. As all elements here are static, they are defined in the setup section rather than inside the main loop.

This section first initializes the display with the "tft.begin()" function, then sets the orientation of the screen through "tft.setRotation(1)" call. The "tft.fillScreen" function creates the black background, with the following functions creating the header. "tft.fillRect()" takes two points on the screen as opposite corners of a rectangle then sets their color to the 5th variable, in this case red.

The four following functions are used to create the lettering of the header. They first define the starting pixel for the words, then set their size and color, then finally the content of the text is defined (Camara Termica foRESTER in this case). This driver does not support special characters, so unfortunately certain letters were not possible to draw unless more time was spent manually drawing pixel by pixel.

Lastly, the border is drawn in white through the use of the "tft.drawLine()" functions, along with the indicator for the temperature in the middle pixel.

```
pinMode(TFT_LED, OUTPUT);
digitalWrite(TFT_LED, HIGH);

tft.begin();

tft.setRotation(1);

tft.fillScreen(ILI9341_BLACK);
tft.fillRect(0, 0, 319, 13, TFT_RED);
tft.setCursor(100, 3);
tft.setTextSize(1);
tft.setTextColor(ILI9341_YELLOW);
tft.print("Camara Termica foRESTER");

tft.drawLine(250, 210 - 0, 258, 210 - 0, tft.color565(255, 255, 255));
tft.drawLine(250, 210 - 90, 258, 210 - 90, tft.color565(255, 255, 255));
tft.drawLine(250, 210 - 180, 258, 210 - 180, tft.color565(255, 255, 255));

tft.setCursor(80, 220);
tft.setTextColor(ILI9341_WHITE, tft.color565(0, 0, 0));
tft.print("T(+) = ");
```

Figure 24-Display outline code

After the outline is drawn the color scale, which allows for quick analysis of the value measurements, is drawn:

```
// drawing the colour-scale
// ========================

for (i = 0; i < 181; i++)
   {
    getColour(i);
    tft.drawLine(240, 210 - i, 250, 210 - i, tft.color565(R_colour, G_colour, B_colour));
   }
```

Figure 25-Color Scale code

After the screen setup, the BLE communication variables are created. These variables are the server, the service, and the two characteristics to transmit the two values described earlier.

```
//////BLE SETUP//////
BLEDevice::init("Thermal Mapping"); // nome do dispositivo bluetooth
// Create the BLE Server
BLEServer *server = BLEDevice::createServer(); //cria um BLE server
server->setCallbacks(new ServerCallbacks()); //seta o callback do server

// Create the BLE Service
BLEService *service = server->createService(SERVICE_UUID);

// Create a BLE Characteristic para envio de dados
characteristicTX = service->createCharacteristic(CHARACTERISTIC_UUID_TX,BLECharacteristic::PROPERTY_NOTIFY);
characteristicTX->addDescriptor(new BLE2902());
/////
characteristicRX = service->createCharacteristic(CHARACTERISTIC_UUID_RX,BLECharacteristic::PROPERTY_NOTIFY);
characteristicRX->addDescriptor(new BLE2902());
////
```

Figure 26-BLE communication setup

After setting the variables up, the service and communication are started:

```
service->start();
// Start advertising (descoberta do ESP32)
server->getAdvertising()->start();
} //fim setup
```

Figure 27-BLE service startup

Having concluded the setup, the main loop begins. This loop will run endlessly, though after connecting the central device to the module and then disconnecting reconnecting becomes impossible due to the "advertising" of the module only being run once throughout the program in the setup stage.

The first issue when developing the new software was the frame and subpage issue. The readings would not work at long distance if the frame issue seen in the short-range tests

persisted. As such, a different approach to the readings was taken, as seen in Figure 28. This approach does show the same errors, as the "MLX90640_CalculateTo()" function runs twice. Further short range testing using this approach could detect the same target at over 13 meters with no ambient error.

Figure 29 contains the first instance of code for the sensor reading function. This loop runs through every single value stored in the sensor's array (mlx90640To) to determine the highest and lowest temperature value read. These values are essential to the module as they set the scale for the color grading displayed on the screen.

Figure 30 contains the second instance of sensor reading, where the program assigns a color to a group of pixels. This color is selected by the "getColour" function, which classifies the color as one of six tiers using a value calculated at the beginning of the secondary "for" loop (see Figure 32 for tiers and color calculation). This value is a measurement of where the real temperature value fits within the boundaries set in the previous function. The "drawline" function serves to set the pixels between two coordinates to a specific color. The "fillRect" function receives the coordinates of a pair of pixels and creates a rectangle of a set color with those two points as opposite corners. Lastly, the "setCursor" and "print" functions allow for text to be displayed without needing to draw out each letter. The former sets the initial position of the text, and the second one writes it.

As these functions will always adapt to the sensor's readings, as long as the values read by the sensor show a high degree of accuracy, the module will always be able to detect hotspots even if the temperatures read are not accurate.

```
void loop(){
    for (byte x = 0 ; x < 2 ; x++) //Read both subpages
       {
        uint16_t mlx90640Frame[834];
        int status = MLX90640_GetFrameData(MLX90640_address, mlx90640Frame);

        if (status < 0)
           {
            Serial.print("GetFrame Error: ");
            Serial.println(status);
           }

        float vdd = MLX90640_GetVdd(mlx90640Frame, &mlx90640);
        float Ta = MLX90640_GetTa(mlx90640Frame, &mlx90640);

        float tr = Ta - TA_SHIFT; //Reflected temperature based on the sensor ambient temperature
        float emissivity = 0.95;

        MLX90640_CalculateTo(mlx90640Frame, &mlx90640, emissivity, tr, mlx90640To);
       }
```

Figure 28 - Frame Reading function

```
mlx90640To[1*32 + 21] = 0.5 * (mlx90640To[1*32 + 20] + mlx90640To[1*32 + 22]);    // eliminate the error-pixels
mlx90640To[4*32 + 30] = 0.5 * (mlx90640To[4*32 + 29] + mlx90640To[4*32 + 31]);    // eliminate the error-pixels

T_min = mlx90640To[0];
T_max = mlx90640To[0];

for (i = 1; i < 768; i++)
   {
    if((mlx90640To[i] > -41) && (mlx90640To[i] < 301))
       {
        if(mlx90640To[i] < T_min)
           {
            T_min = mlx90640To[i];
           }

        if(mlx90640To[i] > T_max)
           {
            T_max = mlx90640To[i];
           }
       }
    else if(i > 0)   // temperature out of range
       {
        mlx90640To[i] = mlx90640To[i-1];
       }
    else
       {
        mlx90640To[i] = mlx90640To[i+1];
       }
   }
```

Figure 29 - Sensor reading function

```
for (i = 0 ; i < 24 ; i++)
   {
    for (j = 0; j < 32; j++)
       {
        mlx90640To[i*32 + j] = 180.0 * (mlx90640To[i*32 + j] - T_min) / (T_max - T_min);

        getColour(mlx90640To[i*32 + j]);

        tft.fillRect(217 - j * 7, 35 + i * 7, 7, 7, tft.color565(R_colour, G_colour, B_colour));
       }
   }

tft.drawLine(217 - 15*7 + 3.5 - 5, 11*7 + 35 + 3.5, 217 - 15*7 + 3.5 + 5, 11*7 + 35 + 3.5, tft.color565(255, 255, 255));
tft.drawLine(217 - 15*7 + 3.5, 11*7 + 35 + 3.5 - 5, 217 - 15*7 + 3.5, 11*7 + 35 + 3.5 + 5,  tft.color565(255, 255, 255));

tft.fillRect(260, 25, 37, 10, tft.color565(0, 0, 0));
tft.fillRect(260, 205, 37, 10, tft.color565(0, 0, 0));
tft.fillRect(115, 220, 37, 10, tft.color565(0, 0, 0));

tft.setTextColor(ILI9341_WHITE, tft.color565(0, 0, 0));
tft.setCursor(265, 25);
tft.print(T_max, 1);
tft.setCursor(265, 205);
tft.print(T_min, 1);
tft.setCursor(120, 220);
tft.print(T_center, 1);
tft.setCursor(265, 115);
tft.print(T_medium, 1);
```

Figure 30 - Image drawing function

```
MLX90640_SetRefreshRate(MLX90640_address, 0x05); //setting to(0x05 is 16Hz)
```

Figure 31 - MLX90640 Refresh rate

Through multiple experiments and combinations of performance settings, it was concluded that this module functions at a maximum 16 Hz, meaning it performs 16 readings per second, translating to the same number of frames per second on the display. The setting of

this value is shown in Figure 31. Further increasing the refresh rate causes the module to freeze frequently, likely due to the limited processing capabilities of the ESP32 board. It is possible that an implementation which makes use of both cores of the microcontroller will be able to handle an increase to 32 Hz, but since implementing an RTOS based solution has proven impossible as of yet, the maximum is capped at 16 Hz.

```
void getColour(int j)
   {
    if (j >= 0 && j < 30)
       {
        R_colour = 0;
        G_colour = 0;
        B_colour = 20 + (120.0/30.0) * j;
       }

    if (j >= 30 && j < 60)
       {
        R_colour = (120.0 / 30) * (j - 30.0);
        G_colour = 0;
        B_colour = 140 - (60.0/30.0) * (j - 30.0);
       }

    if (j >= 60 && j < 90)
       {
        R_colour = 120 + (135.0/30.0) * (j - 60.0);
        G_colour = 0;
        B_colour = 80 - (70.0/30.0) * (j - 60.0);
       }
```

```
    if (j >= 90 && j < 120)
       {
        R_colour = 255;
        G_colour = 0 + (60.0/30.0) * (j - 90.0);
        B_colour = 10 - (10.0/30.0) * (j - 90.0);
       }

    if (j >= 120 && j < 150)
       {
        R_colour = 255;
        G_colour = 60 + (175.0/30.0) * (j - 120.0);
        B_colour = 0;
       }

    if (j >= 150 && j <= 180)
       {
        R_colour = 255;
        G_colour = 235 + (20.0/30.0) * (j - 150.0);
        B_colour = 0 + 255.0/30.0 * (j - 150.0);
       }

   }
```

Figure 32 - Color tiers and calculation

For the BLE communication, some specific variables and processes must be initiated even before the setup. Firstly, each service and characteristic must have a declared UUID in order to communicate properly. As the UUID's used in this project are custom, an external program was needed to generate them. This means that when the project proceeds enough in the future to be mass produced, all the alarm modules services will have the same UUID. This should not be an issue, as the modules should be used with enough distance from each other that one central device will not detect two peripherals.

```
// BLE INCLUDES
#include <BLEDevice.h>
#include <BLEServer.h>
#include <BLEUtils.h>
#include <BLE2902.h>
//objeto responsável pela comunicação com o sensor IR
BLECharacteristic *characteristicTX; //através deste objeto iremos enviar dados para o client
bool deviceConnected = false; //controlo de dispositivo conectado

#define SERVICE_UUID "ab0828b1-198e-4351-b779-901fa0e0371e"

#define CHARACTERISTIC_UUID_TX "3b475608-7762-4e82-b133-a9ff5ab38aa4"
```

Figure 33 - BLE includes and UUID's

Having set up the variables, it is necessary to create a "class" type function to connect and disconnect the microcontroller and the smartphone.

```
class ServerCallbacks: public BLEServerCallbacks {
    void onConnect(BLEServer* pServer) {
      deviceConnected = true;
    };

    void onDisconnect(BLEServer* pServer) {
      deviceConnected = false;
    }
};
```

Figure 34 - Module connection function

Finally, having completed both the declaring of the variables and setup stages, the BLE section of the main loop can run. This section first checks if the device is connected, then creates an array of 20 characters, which is the maximum number of characters that can be displayed by the app currently in use. Afterwards, using the "dtostrf()" function, the float value of the center pixel is copied onto this array. The value of the characteristic is then set to the value of the character array, which then is updated through the "notify()" function. This process then repeats for the value of the maximum temperature.

```
    ////BLE COMMUNICATION/////
    if (deviceConnected) {
//recupera a leitura da temperatura do ambiente
float tempAmbiente = T_center;


char txString[20];
dtostrf(tempAmbiente, 2, 2, txString); // float_val, min_width, digits_after_decimal, char_buffer

characteristicRX->setValue(txString); //seta o valor que a caracteristica notificará (enviar)
characteristicRX->notify(); // Envia o valor para o smartphone

char txString2[20];
dtostrf(T_max, 2, 2, txString2);
characteristicTX->setValue(txString2); //seta o valor que a caracteristica notificará (enviar)
characteristicTX->notify(); // Envia o valor para o smartphone

    }
}
```

Figure 35 - BLE communication in the main loop

Currently, the implementation of BLE communication does not have descriptors for each characteristic, and therefore it is not clear which value is which without beginning the data stream. Future versions might be able to add descriptors but implementing them within schedule was impossible.

4.1.3-Field tests

The specific sensor model used in the following tests was the Sparkfun MLX90640 IR sensor breakout. Multiple tests in various conditions were performed, with the primary objective being determining the distance at which a more realistic flame would be reliably detected, and the secondary objective being how an environment with high thermal "noise" would affect the module's performance. To this effect, by repeating the test using the small alcohol burner, it was determined that the current version of the module could faintly detect the flame at a distance of 13 meters.

The tests were all performed using wood as a fuel, as previous testing was done with coal fires, which burn significantly hotter and are easier to detect. Considering how the temperature of a wood-fueled flame far surpasses the maximum temperature the IR sensor can register, and that the fires used in field tests would be significantly larger, the tests were done at distances between 20 and 50 meters. For additional data points, some tests at 20 meters were made from a point around 5 meters below the flames and with a somewhat obstructed view.

Throughout the tests it was concluded that the IR sensor's readings there was no evidence that visible radiation could influence the readings in any significant way. Targets that reflected a lot of visible light did not influence the readings, unlike darker obstacles at higher temperatures which did result in faraway hotspots being missed.

Unlike previous versions, this implementation of the module does not transmit any data from the ESP32 to the serial port; meaning that recording test results was only possible by way of photography. For the sake of ease of reading, some photos were cleaned and filtered, but their content was in no way changed.

4.1.3.1- First set of tests

The first set of tests was conducted on a partly cloudy day, using a barbecue pit containing various dried branches as fuel. The first test was a reading at 1,5 meters, which was then followed by various readings made from a lower point at 20 meters.

Figure 36 - Fire burning

Figure 37 - Measurement of the embers (295.7ºC)

Figure 38 - 20m test setup (flame is on the top right, behind the wood railing)

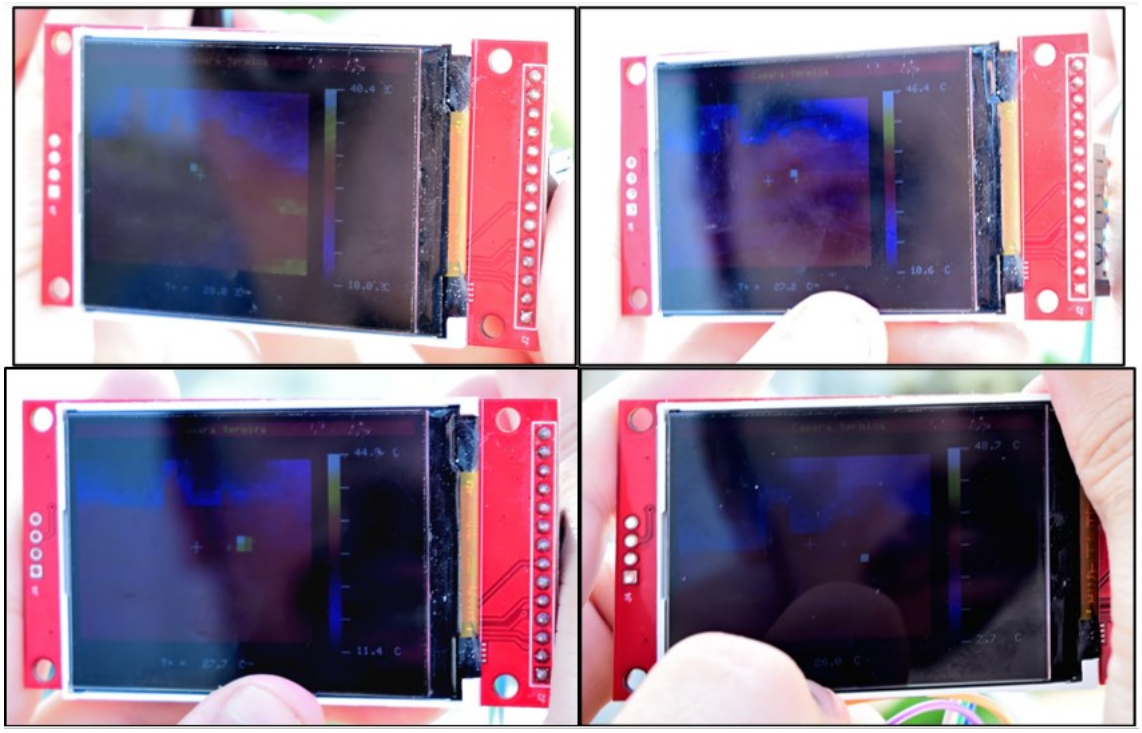


Figure 39 - Readings at 20m. Top left- Embers; Others- Flames burning

As can be observed, the results at 20 meters were quite satisfactory, as there always was one or a few pixels far hotter than their surroundings, represented by the white squares seen on the screen. These tests brought on the question which led to the secondary objective: with

a temperature of around 40 degrees at 20 meters, as seen in the first photo, is it possible that in situations of high ambient temperature it becomes impossible to detect small or medium flames due to thermal noise? To find out, a second set of tests was performed.

4.1.3.2- Second set of tests

The second set of tests was performed during clear sunny days with very high temperatures, as well as sandy ground which got very hot and provided a good source for thermal noise. The target flame was made in a barbecue pit, and dried wood was used as fuel.

The captures were made at distances between 26 and 50 meters. Taking the ambient temperature into account would always make the target considerably hotter than the air around it, the unlit source was measured at a distance of 1.5 meters for control purposes. As seen in the following image, the pit is considerably colder than the ground beneath it, thus it has no risk of false positives.

Figure 40 - Unlit source and reading

After this first control check, the flame was ignited for a second control test, in which the sensor captured temperatures close to its maximum range of 300ºC, at which point the measurements become somewhat unreliable. Fortunately, for the purposes of this module, the detection of incredibly high temperatures is more important than obtaining a very precise value, so any errors regarding the exact value read can be ignored.

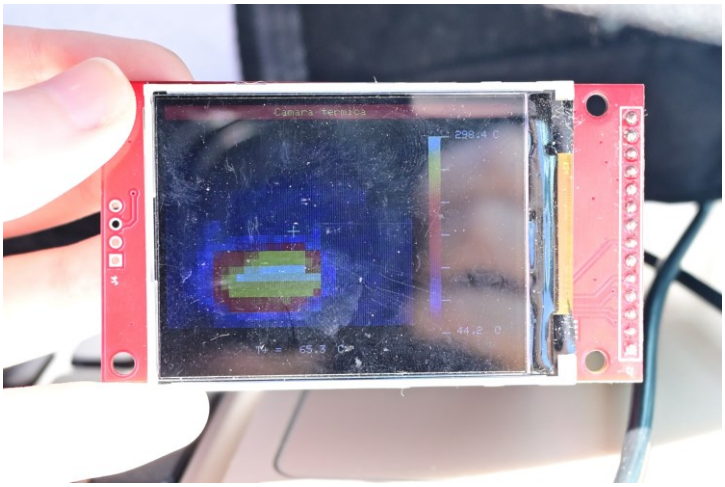


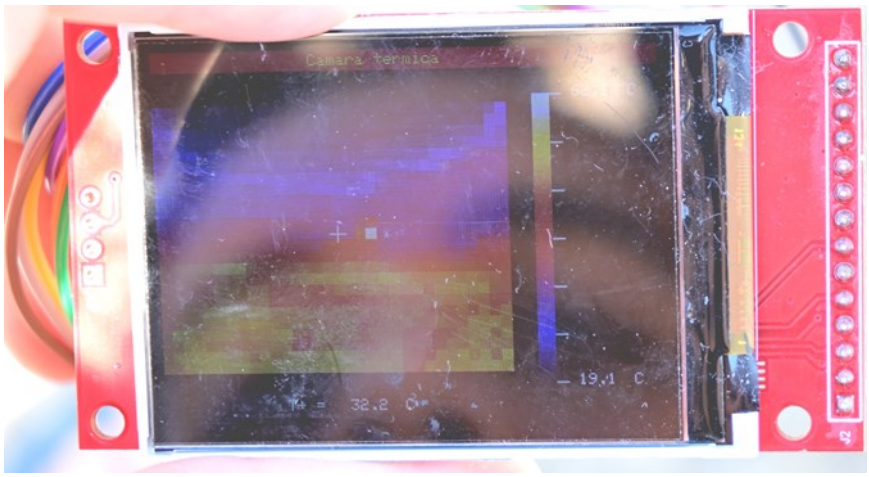
Figure 41 - Flame temperature readings

Following this second control reading, the long distance tests were performed. The first one was at 30 meters, and the testing environment is pictured below, with the target flame in the upper center of the image, just below the blue strip at the bottom of the white wall.

As can be seen in the image, the object is difficult to perceive by the naked eye, thus serving as a good study of how well small flames or reignition sources would be detected using the device.

Figure 42 - Testing environment

This series of tests showed very good results, as the module was able to reliably detect the flames consistently. However, at this range, the high noise levels from the ground made it somewhat difficult to detect the exterior of the barbecue pit reliably when the blaze did not crest above it.



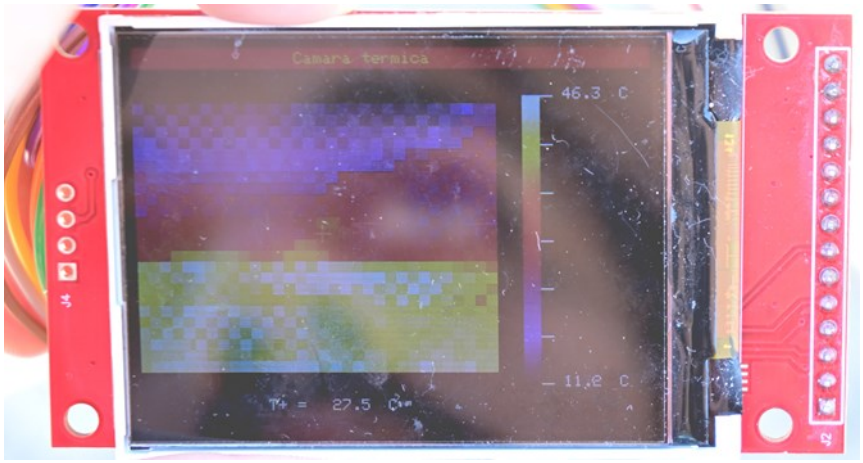


Figure 43 - Readings with (Top)and without (Bottom) flames above the rim of the pit

As seen in the previous images, an open flame registers up to 63ºC, while just the metal exterior registers around 46ºC and is virtually indistinguishable from the ground. After these measurements were taken, the outside of the pit's temperature was measured, showing around 215ºC, which is still a dangerous level of heat to which the user should be informed. Thus, it was concluded that in a sandy field with high thermal noise, this range was the upper limit for reliable detection of hotspots.

After the first set of tests allowed for an estimation for the upper limit for reliable readings, a second set of tests was performed at 50 meters, as this value was discussed as the maximum distance at which the sensor module would be used. These tests were somewhat positive, as it was possible to faintly detect the flame, but this was only possible due to prior knowledge of where the target was.

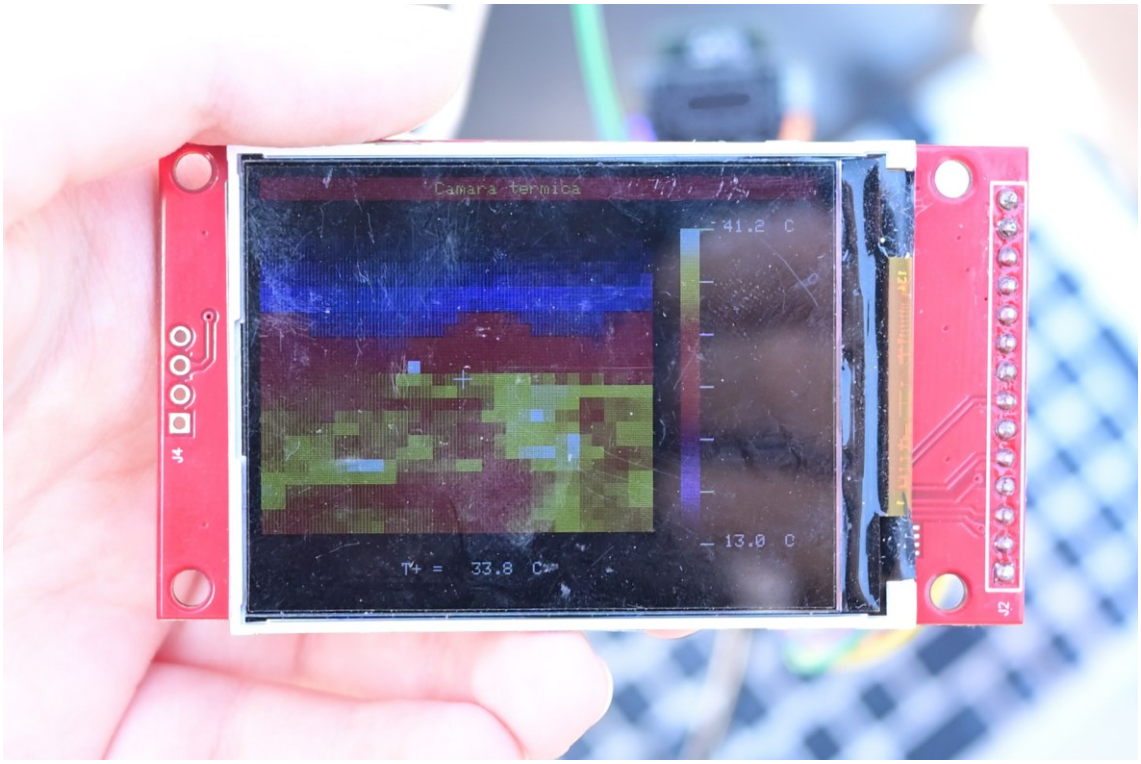


Figure 44 - Flame detected in the center-left of the screen

As the environmental conditions did not allow for testing at this range while there was less thermal noise, it is not possible to confirm the reliability of the sensor module at this range when under less extreme conditions. One can estimate, however, that a hotspot could be reliably detected at the maximum proposed range when the ambient temperature surrounding it is less than 30ºC.

The last set of tests was done at 26 meters and was done to see if vegetation would generate enough noise to obscure hotspots (Figure 45). The results were quite positive, as the flora generated far less infrared radiation than the ground and allowed for clear readings. These tests helped reinforce the idea that the sensor module can be reliable no matter the distance when used in high-vegetation areas, as the overall thermal noise around potential hotspots will be less intense.

Figure 45 - Environment and readings at 26 meters

Additional images as well as full sized versions of the of the various testing environments and readings are catalogued for further analysis available online through the following link: Thermal Mapping Tests, as including every single image would extend this chapter beyond practical lengths.

4.1.4 - BLE communication

BLE communication was the last function to be implemented, and its inclusion was done after the field tests were completed. Communication tests were done using the "BLE Scanner" app. Using this app requires having location turned on, as per the recent changes to BLE protocols.

Figure 46 - BLE Scanner Logo

When the app opens, all the available devices are displayed, along with their distance to the smartphone (Figure 47 (a)). The user can then connect to the Thermal mapping module, where the services are displayed. They can then open the Custom Service to see the characteristics available (Figure 47 (b)). Again, these are not immediately identifiable due to descriptor limitations.

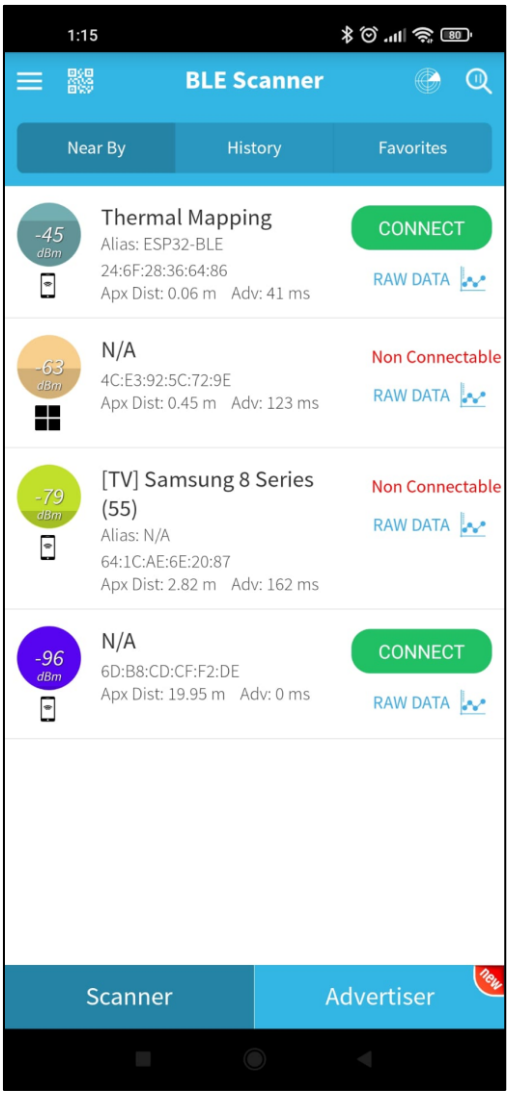


Figure 47 - (a) Available devices

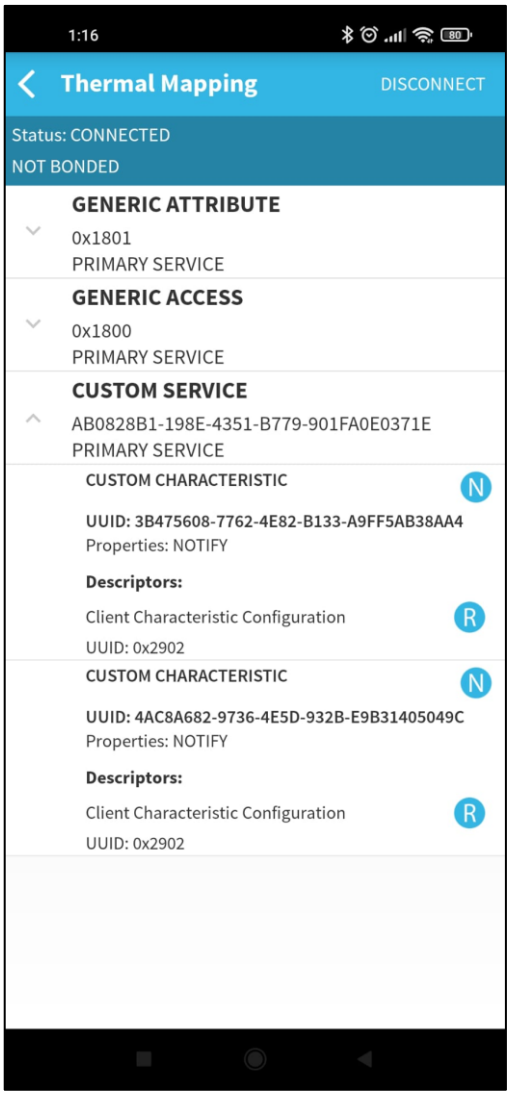


Figure 47 - (b) Characteristics available

Once the user presses the blue N button, the app begins retrieving data from the module, and the values will update continuously. Due to some added formatting to the values sent, it is now possible to discern which value is which, as seen in Figure 48.

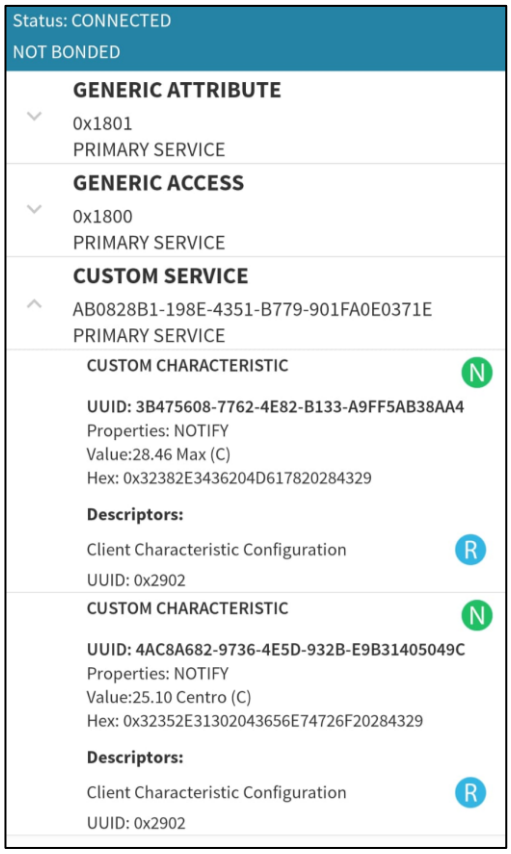


Figure 48 - Values being read

It is worth noting that while both values can keep updating without error, the app itself is not as reliable when it comes to begin the NOTIFY function. Once the first characteristic begins streaming, the button for the second one sometimes becomes unresponsive. This issue can be resolved easily by collapsing and reopening the Custom Service tab above the characteristics.

4.1.5- Final notes and module assessment

The thermal mapping module currently performs all required functions as per the original plan. The information obtained from the sensor is useful and easy to analyze, and will therefore assist the user in the field, even in extreme situations.

Due to the current state of the module, it has some hardware-related flaws which can impede its proper function. The wire connection to the IR sensor is somewhat sensible, which can cause it to disconnect. When this happens, the sensor's readings fall out of synch with the ESP32 and the system as a whole fails critically. This can be solved quickly by assuring the sensor is properly connected then restarting the module, but nevertheless poses a risk of impeding proper usage of the module in the field. Future versions of the model where the sensor

and screen are part of a PCB in order to create a single device with no movable wires would resolve this issue.

The OLED screen was selected due to being the most cost-effective model, and while the images it can display are clear and the results very satisfactory, it should be noted that in environments which are really bright it becomes difficult to discern some details. Implementing some way of blocking light coming from the sides would solve this issue.

4.2- Alarm module

The current version of the alarm module features only the GNSS receiver and the oxygen sensor and does not make use of the carbon monoxide sensor. Due to working environment constraints it was not possible to comprehensively test the carbon monoxide sensor in a controlled environment, so it was decided that its implementation into the module would be done in future developments.

Previous versions of this module [40] required extensive tests for both the GNSS receiver and the O2 sensor, so the focus on the development of this version was developing software and gaining a further understanding of how the GNSS receiver and the oxygen sensor function, as well as some limitations to their usage. The tests performed during development were concerned with response time and communication rather than precision, as both components had their readings validated for practical usage.

4.2.1- Oxygen Sensor

4.2.1.1- Sensitivity and Recovery tests

The first focus of the alarm module was studying the O2 sensor and, using some basic example code provided by the manufacturer [43], conduct some basic tests. The code used is identical to the one in the final version of the module, so to avoid redundancies all software explanations concerning this sensor will be presented at the end of the chapter. The test conducted sought to ascertain the response time to a quick drop in oxygen levels, the recovery time after returning to normal conditions, and responsiveness to smaller variations following large decreases in air quality. This was done to ensure that after being exposed to extreme conditions the sensor would still detect the presence of dangerous, albeit not as dire, conditions.

The setup was similar to tests done in previous versions of this project. A bottle was placed over a source of fire, creating an almost airtight environment, and thus depleting its oxygen at a rapid pace. The difference between prior tests and this one lies in the way the sensor enters the space. While in previous tests the sensor would always be on top of the container, and thus would record the lowering oxygen concentrations as the flame burned and eventually went out, this setup has transparent film place over the top of the bottle. This allows the creation of an oxygen-scarce environment which, through breaking the film, the oxygen sensor can enter quickly without allowing an intake of clean air. Testing in this manner can help measure how quickly the sensor readings evolve when the concentration of oxygen plummets quickly.

Figure 49 - Oxygen test setup

After the measured oxygen levels reach their lowest point, the sensor was taken out of the container and exposed to clean air again in order to measure recovery time. This was done twice, after which the sensor was exposed to human breath in order to measure a less intense but still dangerous environment. These variations were then repeated again, alternating between exposure to breath and flame.

Results were consistent, and even after extremely low readings the recovery and stabilization time was around 35 seconds, with the readings of the breath values still meeting the

correct estimates. The following images show a graph of one of the tests conducted, with the X axis being the number of readings done, which happen two times per second.

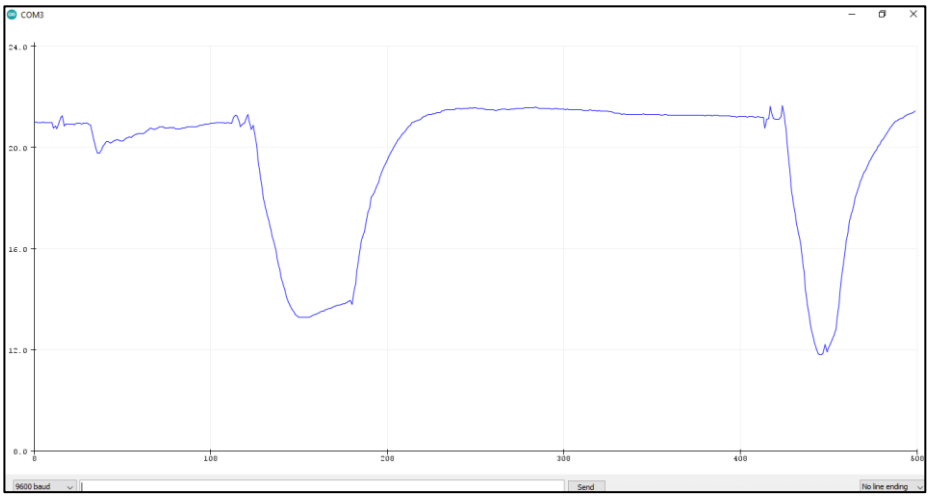


Figure 50 - Initial Flame readings

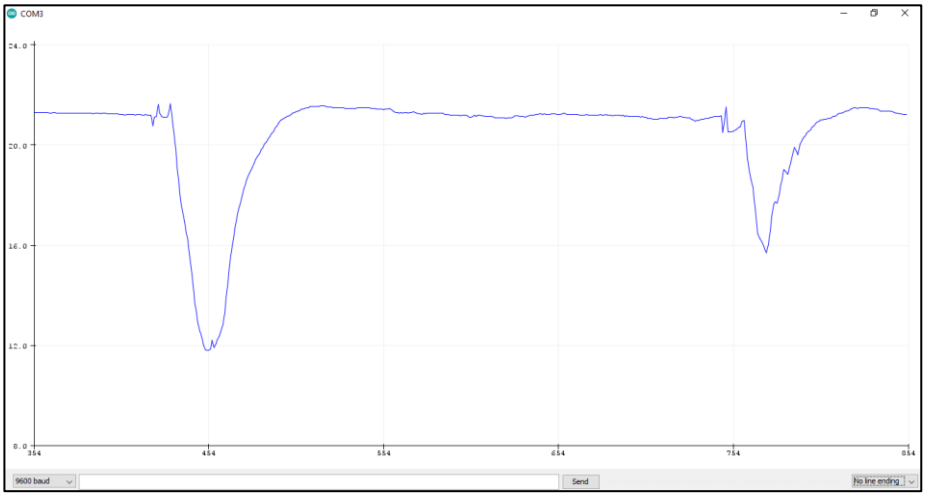


Figure 51 - Breath readings after flame

The oxygen levels drop as low as 12% when in a closed chamber, and dip just below 16% when exposed to a single prolonged human breath. From these tests it was concluded that the sensor can respond and recover from extreme changes, retaining the precision of the readings and ability to detect smaller variations in concentration. Additionally, it was observed that after a large drop in oxygen concentration, the sensor tends to overcorrect and reach a higher reading that initial. Seeing as this error lasts for only a few seconds, is of small magnitude (less than 5%) and the sensor still responds as required when exposed to low oxygen, the overcorrection can be ignored. Finally, it was concluded that it would be pertinent to create a control system to avoid false positives caused by users breathing too close to the alarm module.

4.2.1.2- Oxygen sensor functions

The proper calibration of the oxygen sensor's readings was done through some trial and error. This was due to there being no resistor between the ESP32 microcontroller and the ME2-O2 sensor in order to make the module more energy-efficient. Additionally, the initial testing software considers that pure air has an oxygen concentration of 21%, when the value is slightly lower. The final version of the software, detailed in this section, considers 20.5% to be the concentration in pure air.

The way this sensor works, as described in chapter 2, is through an electrochemical cell which produces current when exposed to oxygen. The more current it produces, the higher the measured voltage, which means a higher oxygen concentration. In order to calculate the readings, the microcontroller reads the output voltage of the sensor and then based on a reference voltage and a set of values based on calibration tests converts the voltage into an oxygen concentration estimate.

The reference voltage for this module is 3.3V, as the 5V pin of the ESP32 is dedicated to the GNSS receiver. The selected analog pin was number 13 due to convenience of assembly and can be swapped for any other pin if needed.

```
/**/ float O2_val = 0;                /**/
/**/ const float VRefer = 3.3;            /**/
/**/ #define pinAdc 13 //analog pin
```

Figure 52 - Reference Voltage and digital pin

Having declared the reference voltage, we can now begin extracting oxygen concentration values from the sensor. This is achieved through a combination of two functions, one to retrieve the value of the voltage read at the analog pin, and one which uses the previous value to calculate the oxygen concentration.

The first function first defines a "long" type variable onto which it will save the average of 128 readings. The averaging in this function is done by performing a binary shift with an operand of 7 on the sum of the values. This value is then multiplied by the reference voltage and finally divided by 1500, a default value.

```
float readO2Vout()
{
    long sum = 0;
    for(int i=0; i<128; i++)
    {
        sum += analogRead(pinAdc);
    }
    sum >>= 7;
    float MeasuredVout = sum * (VRefer / 1500.0);
    return MeasuredVout;
}
```

Figure 53 - Read Voltage function

The returned value of this function is then used by a secondary function, the values of which were manually set for the purpose of calibration. The value of the "MeasuredVout" variable is 3.1V under normal conditions (clear air in an open space), and the baseline oxygen concentration is assumed to be 20.5%.

```
float readConcentration()                                    /**/
{                                                          /**/
    float MeasuredVout = readO2Vout();
    float Concentration = MeasuredVout * 0.205 / 3.1;      /**/
    float Concentration_Percentage=Concentration*100;
    return Concentration_Percentage;                                /
}                                                          /**/
```

Figure 54 - Oxygen percentage calculation

4.2.2- GNSS Receiver

4.2.2.1- Sensor tests and validation

Due to issues with transportation and connectivity, as well as the hardware itself, tests using the NEO-7M receiver were substantially limited. Throughout the testing process, the GNSS receiver displayed a substantial inconsistency with start-up times, which ranged between 10 minutes and over 3 hours, with no change in setup. Currently, the assumption is that the waiting period before results can be extracted from the receiver is affected by how long it has been disconnected from a power source.

The GNSS receiver collects date, time, and location from the satellites. Testing shows that retrieval of the date and time is substantially quicker on start-up. Upon resetting the module, the retrieval of the date and time is immediate, unlike the GPS coordinates, which take at least 10 minutes to connect.

After the connection is established, the sensor's readings have a moderate degree of precision. The current software reads to the sixth decimal point, which corresponds to a distance of 0.111m, though in the hour of operation after establishing a position, the values oscillate with a somewhat significant range, as much as 50 meters over a period of 2 minutes, despite the module being stationary. This value tends to stabilize to some extent, going as low as 10 meters and averaging at 20 meters over the same timeframe.

For the purposes of practical usage, the precision of the receiver is acceptable, as the purpose of the alarm module's ability to convey coordinates is to allow for emergency support to know where the vehicle in danger is located so that assistance or rescue can be provided. A level of precision of 20 meters is sufficient for a user with knowledge of the road maps within the area to determine a more exact position, and once additional manpower arrives on the scene of danger it should be easy enough to locate the vehicle which triggered the alarm.

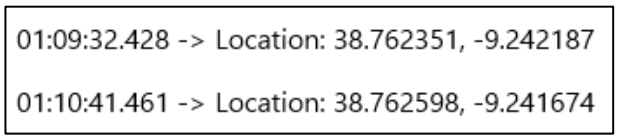


Figure 55 - Difference of almost 30 meters in little over 1 minute

Regarding the starting time for the receiver, current results leave much to be desired. The inability to establish a connection quickly after a long time without power means that practical usage will require establishing a dedicated source of power for the receiver to be operational within reasonable practical standards. Additionally, the stabilization time can be detrimental in some extreme cases where dangerous conditions manifest quickly. Improving these areas would be critical going forward, as it would make the alarm module more reliable and energy efficient.

The test performed on this receiver contradict the ones performed in [40], which is why it is recommended that further developments proceed with further testing in order to validate the usage of this particular receiver.

4.2.2.2- Software developed

The software developed to obtain the coordinates was simple, as the library required to use the NEO-7M receiver, TinyGPS++, comes with code examples to demonstrate basic functions. Very few alterations were done to the base software, as the inclusion of the serial communication and information regarding the time and date were crucial for testing and analysis purposes. The serial communication allowed the recording of values read over a large

period, and the time-related information served to ascertain if communication was happening during the waiting time of the GPS portion of the receiver.

As mentioned before, including the "TinyGPS++" library is necessary for the functions to work:

```
#include <TinyGPS++.h>
#include <SoftwareSerial.h>
```

Figure 56 - Libraries included

To read from the receiver, one most declare the RX and TX pins, which will serve as the communication ports for the GNSS receiver. These two port numbers cannot be changed, as they are the only ones in the ESP32 board. Each of these ports must also be connected to the opposing port in the receiver (TX->RX and RX->TX). One must also declare the GPS object onto which the functions will be executed.

```
// The TinyGPS++ object
TinyGPSPlus gps;

// The serial connection to the GPS device
SoftwareSerial ss(RXPin, TXPin);

static const int RXPin = 16, TXPin = 17;
static const uint32_t GPSBaud = 9600;
```

Figure 57 - GPS variable declarations

Having declared the variables, it is now possible to retrieve the data from the GNSS receiver. This is done through a function named "displayInfo" which is called from the main loop after checking if data is being collected from the receiver. If there is no data for more than 5 seconds, the program notified the user through the serial port and freezes. This checking feature is useful for testing connections in the assembly of new devices.

```
void loop()
{
  // This sketch displays information every time a new sentence is correctly encoded.
  while (ss.available() > 0)
    if (gps.encode(ss.read()))
      displayInfo();

  if (millis() > 5000 && gps.charsProcessed() < 10)
  {
    Serial.println(F("No GPS detected: check wiring."));
    while(true);
  }
}
```

Figure 58 - Main loop of the alarm module

The "displayInfo" function has 3 identical sections for collecting each of the data types from the NEO-7M receiver. Each one consists of first checking the validity of the information gathered, then, if it is valid, write those values into the serial port. This is done for the GPS coordinates, the time, and the date. This function was altered to include the declaration of two variables in order to store the values of latitude and longitude so they can be then transmitted through BLE. This change was required due to the original "displayInfo" function relying on the "gps.location"-type functions of the "TinyGPS++" library. It is also through the displayInfo function that the functions for collecting values from the oxygen sensor are called.

```
void displayInfo()
{
  Serial.print(F("Location: "));
  float local = gps.location.lat();
  float local2 = gps.location.lng();
 // Serial.println(local);
  if (gps.location.isValid())
  {
    Serial.print(gps.location.lat(), 6);
    Serial.print(F(","));
    Serial.print(gps.location.lng(), 6);
  }
  else
  {
    Serial.print(F("INVALID"));
  }
```

Figure 59 - Display GPS info

```
Serial.print(F("  Date/Time: "));
if (gps.date.isValid())
{
  Serial.print(gps.date.day());
  Serial.print(F("/"));
  Serial.print(gps.date.month());
  Serial.print(F("/"));
  Serial.print(gps.date.year());
}
else
{
  Serial.print(F("INVALID"));
}
```

```
Serial.print(F(" "));
if (gps.time.isValid())
{
  if (gps.time.hour() < 10) Serial.print(F("0"));
  Serial.print(gps.time.hour());
  Serial.print(F(":"));
  if (gps.time.minute() < 10) Serial.print(F("0"));
  Serial.print(gps.time.minute());
  Serial.print(F(":"));
  if (gps.time.second() < 10) Serial.print(F("0"));
  Serial.print(gps.time.second());
  Serial.print(F("."));
  if (gps.time.centisecond() < 10) Serial.print(F("0"));
  Serial.print(gps.time.centisecond());
}
else
{
  Serial.print(F("INVALID"));
}
```

Figure 60-Display date and time

After all these functions are called, a variable to store the oxygen concentration is created:

```
else
{
  Serial.print(F("INVALID"));
}

float local3= readConcentration();
```

Figure 61-Get Oxygen concentration

4.2.3- BLE communication

Like the thermal mapping module, the alarm module serves as a peripheral device connected to a central device through Bluetooth Low Energy, with the central device being the user's smartphone. Once collected, the data from the sensors is sent to the smartphone through a custom BLE service in the form of NOTIFY-type BLE characteristics.

Another similarity to the thermal mapping module, due to the BLE communication start being declared in the setup section of the software, once the device connects and disconnects from the central device, it must be restarted manually before it can be used again.

The characteristics do not possess descriptors due to the previously mentioned implementation issue, so in order to avoid confusion a small indicative string is added to the end of the string which contains the values. The text added indicates if the characteristic contains the latitude (Lat), longitude (Long), or oxygen concentration percentage (%).

```
////BLE COMMUNICATION/////
  if (deviceConnected) {

  char txString[20];
  dtostrf(local, 2, 6, txString); // float_val, min_width, digits_after_decimal, char_buffer
  strcat(txString, " Lat ");
  characteristicRX->setValue(txString); //seta o valor que a caracteristica notificará (enviar)
  characteristicRX->notify(); // Envia o valor para o smartphone

  char txString2[20];
  dtostrf(local2, 2, 6, txString2);
  strcat(txString2, " Long");
  characteristicTX->setValue(txString2); //seta o valor que a caracteristica notificará (enviar)
  characteristicTX->notify(); // Envia o valor para o smartphone

  char txString3[20];
  dtostrf(local3, 2, 2, txString2);
  strcat(txString2, " %");
  characteristicVX->setValue(txString2); //seta o valor que a caracteristica notificará (enviar)
  characteristicVX->notify(); // Envia o valor para o smartphone
  }
Serial.println();
}
```

Figure 62 - BLE communication for the alarm module

The communication of values works reliably even with an added characteristic, with the previously encountered error where the app ceases to respond manifesting with the same frequency and being solvable with the same solution. These results bode well for future iterations of the project, which should see the addition of the carbon monoxide sensor data and therefore an additional characteristic. The following figures show the alarm module ready to connect and the transmitted data when the sensor is at 38.762360537373056, -9.24299000680553 [44].

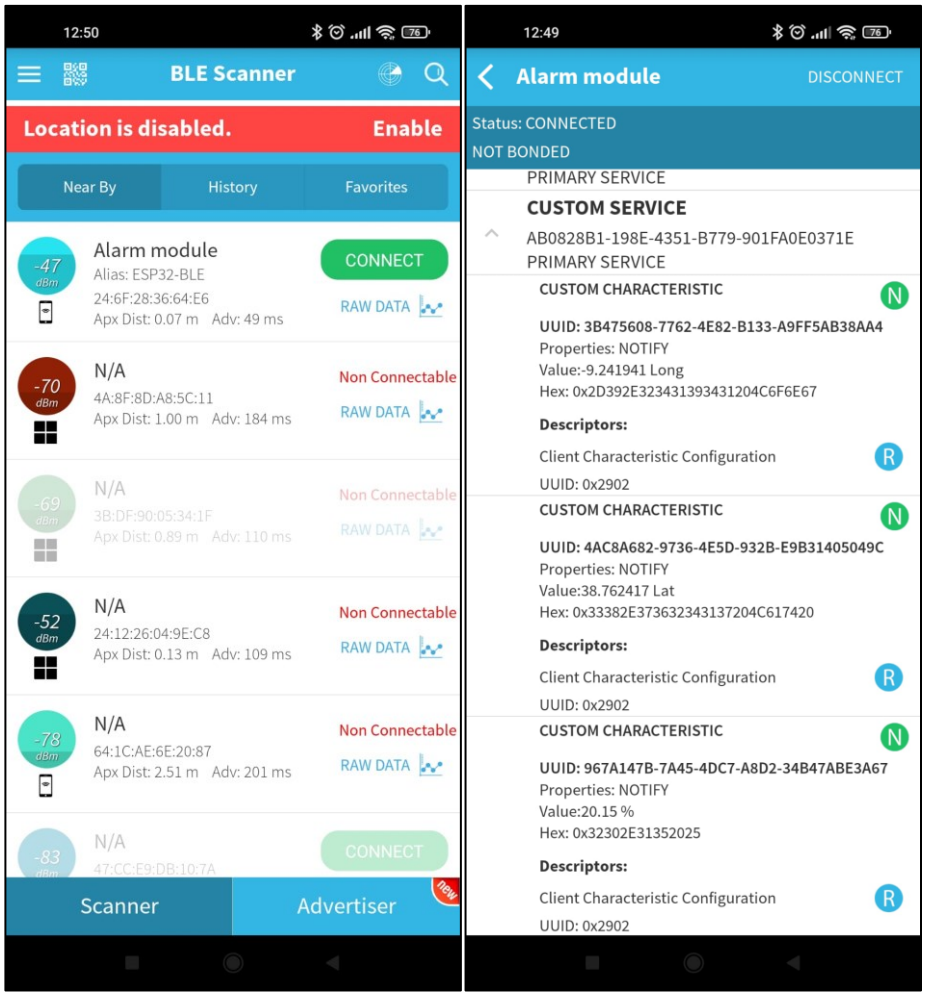


Figure 63-Alarm module available and transmitted data

4.3-Bill of materials:

The following tables consist of the various components and their cost, in order to obtain the overall cost of each of the modules.

Thermal Mapping Module:

Table 1 - Thermal Mapping Module BoM

Component	Model	Cost	Total Cost
Microcontroller	ESP32 DevkitC	9 €	
Multi-pixel IR sensor	Sparkfun MLX90640	62.10 €	
LCD Display	ILI9341	15.33 €	
			86 €

Alarm Module:

Table 2 - Alarm Module BoM

Component	Model	Cost	Total Cost
Microcontroller	ESP32 DevkitC	9 €	
GNSS Receiver	Neo-7M	33.00 €	
O2 Sensor	Grove ME2-O2	50.00 €	
CO sensor	MQ-9	7.26 €	
			99 €

5

5. Conclusion and Future Work

5.1- Conclusion

As stated in the target goals, the Thermal Mapping and Alarm modules were designed, developed, and tested. The Thermal Mapping module was a complete success, with every objective being met. As for the Alarm module, while the implementation of the carbon monoxide sensor was not possible, the oxygen values and GPS coordinates are great assets.

This project was treated not as a standalone endeavor but as the continuation of a process. A large amount of the effort was put into creating system models to ease replication, as well as establishing which objectives should be prioritized and which should be left in preliminary stages for future integration.

The two new alarm modules developed over the course of this project have already proven their practical use, with the thermal mapping module showing great results even in extreme conditions, and the alarm module having a good degree of accuracy in all readings. Most importantly, the implementation of BLE communication will allow for the usage of a smartphone as a transmission device will allow for better communication with the central ArcGIS platform.

The biggest obstacle in development was understanding and working around the limitations of the hardware due to lack of documentation or conflicting resources, particularly regarding the CO sensor, which is why it was left out the current version of the alarm module, and the GNSS receiver, the functioning of which was not completely comprehended.

Overall, the work developed and the value produced were substantial. The current state of the modules can already be used to some extent, and through thorough modelling and planning they can be easily used as a base for future developments which could include the mass production and distribution of the modules.

5.2- Future Work

There are several areas in which the project as a whole and the modules themselves can evolve. For the communication aspect, the BLE Scanner app is insufficient. As such, an app capable of either receiving the data through BLE and posting it to the ArcGIS server by itself or which can work in conjunction with the BLE Scanner to read the values from it and then forward them to the server. Additionally, the implementation of RTOS technology will enable the usage of the ESP32's two cores in order to make the modules' ability to process information significantly faster.

Regarding the thermal mapping module, the change from an ESP32 board with the sensor and screen connected by wires to a PCB connecting them would remove the errors caused by connection interruptions. Additionally, a protective box with a way to provide shade around the OLED screen would prevent glare and make it easier to read. Another possibility would be using a different screen to increase resolution, but this would be a very big undertaking due to screens with a resolution large enough to justify changing the screen model often having unique software.

The alarm module is where the most notable improvements can be implemented. The addition of the carbon monoxide sensor and the establishment of a proper functioning GNSS receiver would be major developments in making it as valuable as possible. It is possible that the NEO-7m model might not be the appropriate hardware to use, so it would be pertinent to test other models to determine if the problems experience during development were due to the sensor itself or due to how it is currently being used. Either way, further testing of the GNSS receiver is required. Additionally, developing alarm thresholds for the gas sensors is required for ease of use.

References

[1]- M.R. Hall, D. Allinson, in Materials for Energy Efficiency and Thermal Comfort in Buildings, 2010. Accessed: 11/2020

[2]- J. R. Gyorki. Understanding the Infrared Temperature Sensor. Sept. 2009. url: https://www.sensortips.com/temperature/infrared-temperature-sensor/.
Accessed: 11/2020

[3]- Stefan-Boltzmann Law. url: http://hyperphysics.phy-astr.gsu.edu/hbase/thermo/stefan.html. Accessed: 11/2020.

[4]- Fluke. What is Emissivity? url: https://www.flukeprocessinstruments.com/en-us/service-and-support/knowledge-center/infrared-technology/whatis-emissivity. Accessed: 11/2020

[5]- Definition: active sensor. url: https://internetofthingsagenda.tech-target.com/definition/active-sensor Accessed: 11/2020.

[6] P. Jacobs. Thermal infrared characterization of ground targets and backgrounds. 2nd ed. Jan. 2006, pp. 1–184. doi: 10.1117/3.651915. Accessed: 11/2020

[7]- OMEGA. Introduction to Non- contact Infrared Thermometers. url: https://sea.omega.com/my/prodinfo/infraredthermometer.html.
Accessed: 11/2020.

[8]- E. TollBox. Radiation Heat Transfer. 2003. url: https://www.engineer-ingtoolbox.com/radiation-heat-transfer-d_431.html. Accessed: 11/2020.

[9]- Photodiode Working Principle, Characteristics and Applications. url: https://www.elprocus.com/photodiode-working-principle-applications/.
Accessed: 11/2020.

[10]- Dictionary of the English Language. 5th ed. pyroelectricity. Houghton Mifflin Harcourt Publishing Company, 2016. url: https://www.thefreedictionary.com/pyroelectricity. Accessed: 11/2020.

[11]- How does an Oxygen sensor work? – url: https://gaslab.com/blogs/articles/how-does-an-oxygen-sensor-work. Accessed: 11/2020

[12]- The Different Types of Oxygen Sensor – url: https://www.azosensors.com/article.aspx?ArticleID=1708. Accessed: 11/2020

[13]- Oxygen Sensor Types – url: https://aoi-corp.com/articles/oxygen-sensor-types/. Accessed: 11/2020

[14]- Sieber, Arne (18–20 May 2012). Vann, Richard D.; Denoble, Petar J.; Pollock, Neal W. (eds.). Rebreather Forum 3 Proceedings. Durham, North Carolina. url: http://media.dan.org/RF3_web.pdf Accessed: 11/2020

[15]- W. H. Nernst (1864-1941)

[16]- How does a Zirconia Oxygen Sensor Work? url: https://www.co2meter.com/blogs/news/how-does-a-zirconia-oxygen-sensor-work. Accessed: 11/2020

[17]- Bylund Automotive Educative Products. url: http://www.bylund-automotive.com/educative/. Accessed: 11/2020

[18]- UL Standard 2075, Standard for Gas and Vapor Detectors and Sensors. Accessed: 11/2020

[19]- Clifford, Paul K., and Michael G. Dorman. Test Protocols for Residential Carbon Monoxide Alarms, Phase 1. Gas Research Institute, 1996.

[20]- Penney, David G., ed. Carbon monoxide toxicity. CRC Press, 2000.

[21]- Chris Woodford, Carbon monoxide detectors. url: https://www.explainthatstuff.com/carbonmonoxidedetectors.html. Accessed 12/2020

[22]- U.S. Patent 5,055,688. url: https://patents.google.com/patent/US3898462. Accessed: 12/2020

[23]- M. Brain and T. Harris. "How GPS Receivers Work." In: (Sept. 2006). url: https://electronics.howstuffworks.com/gadgets/travel/gps.htm. Accessed: 12/2018.

[24]- TechTarget. Satellite Constellation. Mar. 2017. url: https://whatis.techtarget.com/definition/satellite-constellation. accessed: 12/2020.

[25] GPS.gov. Space Segment. url: https://www.gps.gov/systems/gps/space/. Accessed: 12/2020.

[26] A. Brierley-Green. "Global Navigation Satellite System Fundamentals and Recent Advances in Receiver Design." In: IEEE Long Island Section. Maxim Integrated Inc., Sept. 2017. url: https://www.ieee.li/pdf/viewgraphs/gnss_fundamentals.pdf. Accessed: 12/2020.

[27] Galileo: a constellation of navigation satellites url: https://www.esa.int/Applications/Navigation/Galileo/Galileo_a_constellation_of_navigation_satellites. Accessed: 12/2020.

[28] RACELOGIC. How does DGPS (Differential GPS) work? url: https://racelogic.support/01VBOX_Automotive/01General_Information/Knowledge_Base/How_Does_DGPS_(Differential_GPS)_Work%3F.
Accessed: 12/2020.

[29] ESA. System Design Details. 2014. url: https://gssc.esa.int/navipedia/index.php/Receiver_Block_Diagram. Accessed: 12/2020.

[30] ESA. Front End. 2011. url: https://gssc.esa.int/navipedia/index.php/Front_End Accessed: 12/2020

[31] ESA. Digital Signal Processing. 2014. url: https://gssc.esa.int/navipedia/index.php/Digital_Signal_Processing. Accessed: 12/2020.

[32] ESA. Tracking Loops. 2014. url: https://gssc.esa.int/navipedia/index.php/Tracking_Loops. Accessed: 12/2020.

[33] M. Brain. How Microcontrollers Work. Apr. 2000. url: https://electronics.howstuffworks.com/microcontroller1.htm. Accessed: 12/2020.

[34] PC MAG. Definition of microcontroller. url: https://www.pcmag.com/encyclopedia/term/46924/microcontroller. Accessed: 12/2020.

[35] Collins English Dictionary – Complete and Unabridged. 12th ed. nonvolatile memory. HarperCollins, 2014. url: https://www.thefreedictionary.com/nonvolatile+memory. Accessed: 12/2020.

[36] CATHERINE BARRAT, HGH INFRARED SYSTEMS INC. IR Sensor Calibration: When Extreme Uniformity Is Not Boring. url: https://www.photonics.com/Articles/IR_Sensor_Calibration_When_Extreme_Uniformity_Is/a60621.
Accessed: 02/2021

[37] Richard E. Fairfax. "Clarification of OSHA's requirement for breathing air to have at least 19.5 percent oxygen content". url: https://www.osha.gov/laws-regs/standardinterpretations/2007-04-02-0. Accessed: 02/2021

[38] Liang Shan, Huiyun Huang, Bo Hong, Jun Zhao, Daodang Wang, and Ming Kong. "Temperature Measurement Method of Flame Image Fusion with Different Exposures". doi:10.3390/en13061487. Accessed: 02/2021

[39] DAVID L. HALL. "An Introduction to Multisensor Data Fusion" Accessed: 02/2021

[40] José Pedro Marques Curva. "Infrared Fire Alarm for Vehicle Protection" Accessed: 11/2020

[41] MQ9 Carbon Monoxide Sensor Datasheet. https://www.haoyuelectronics.com/Attachment/MQ-9/MQ9.pdf Accessed: 01/2021

[42] Official Bluetooth Website. https://bluetooth.com Accessed 06/2021

[43] Grove O2 Gas Sensor page. https://wiki.seeedstudio.com/Grove-Gas_Sensor-O2/ Accessed: 04/2021

[44] Google maps: https://www.google.pt/maps/@38.7623684,-9.2426708,20z

[45] M. Oliveira. "Bombeiros criticam estratégia de combate ao incêndio de Monchique." Público (Aug. 2018). url: https://www.publico.pt/2018/08/08/sociedade/noticia/bombeiros-criticam-estrategia-de-combate-no-incendio-de-monchique-1840416. Accessed: 11/2021

[46] MQ9 Gas sensor datasheet. https://www.electronicoscaldas.com/datasheet/MQ-9_Hanwei.pdf Accessed: 04/2021

[47] ILI9341 Datasheet. https://www.digikey.com/htmldatasheets/production/1640716/0/0/1/ili9341-datasheet.html Accessed 03/2021

[48] Gas sensors-Connection details. Accessed: 03/2021 https://alselectro.wordpress.com/2011/09/18/gas-sensorsconnection-details/

I. Annex 1- Coding

Code for the Thermal Mapping Module: /*#include <Adafruit_ILI9341.h>

```
/*#include <Adafruit_ILI9341.h>

#include <Adafruit_GFX.h>
#include <Adafruit_GrayOLED.h>
#include <Adafruit_SPITFT.h>
#include <Adafruit_SPITFT_Macros.h>
#include <gfxfont.h>*/


#include <Wire.h>
#include "MLX90640_API.h"
#include "MLX90640_I2C_Driver.h"
#include <SPI.h>
//#include <Adafruit_GFX.h> //(changed)
//#include "Adafruit_ILI9341.h" //(changed)
#include <TFT_eSPI.h> // Hardware-specific library


#define TFT_CS   15
#define TFT_DC    2
#define TFT_MOSI 13
#define TFT_CLK  14
#define TFT_RST  26
#define TFT_MISO 12
#define TFT_LED  27


TFT_eSPI tft = TFT_eSPI(); //(changed)

const byte MLX90640_address = 0x33; //Default 7-bit unshifted address of the
MLX90640
```

```
#define TA_SHIFT 8 //Default shift for MLX90640 in open air

static float mlx90640To[768];
paramsMLX90640 mlx90640;

int xPos, yPos;
int R_colour, G_colour, B_colour;
int i, j;
float T_max, T_min, T_medium;
float T_center;


// BLE INCLUDES
#include <BLEDevice.h>
#include <BLEServer.h>
#include <BLEUtils.h>
#include <BLE2902.h>
//objeto responsável pela comunicação com o sensor de temperatura infraver-
melho
BLECharacteristic *characteristicTX; //através deste objeto iremos enviar
dados para o client
BLECharacteristic *characteristicRX; //através deste objeto iremos enviar
dados para o client

bool deviceConnected = false; //controlo do dispositivo

#define SERVICE_UUID "ab0828b1-198e-4351-b779-901fa0e0371e"
#define CHARACTERISTIC_UUID_RX "4ac8a682-9736-4e5d-932b-e9b31405049c"
#define CHARACTERISTIC_UUID_TX "3b475608-7762-4e82-b133-a9ff5ab38aa4"

class ServerCallbacks: public BLEServerCallbacks {
    void onConnect(BLEServer* pServer) {
      deviceConnected = true;
    };

    void onDisconnect(BLEServer* pServer) {
      deviceConnected = false;
    }
};



// ***************************************
// *************** SETUP ***************
// ***************************************

void setup()
   {
    Serial.begin(115200);
    Wire.begin();
    Wire.setClock(5000000); //speed at 5Mhz (max)
   if (isConnected() == false)
       {
        Serial.println("MLX90640 not detected at default I2C address. Please
check wiring. Freezing.");
        while (1);
       }
```

```
    Serial.println("MLX90640 online!");

    //Get device parameters - We only have to do this once
    int status;
    uint16_t eeMLX90640[832];

    status = MLX90640_DumpEE(MLX90640_address, eeMLX90640);

    if (status != 0)
       Serial.println("Failed to load system parameters");

    status = MLX90640_ExtractParameters(eeMLX90640, &mlx90640);

    if (status != 0)
       {
        Serial.println("Parameter extraction failed");
        Serial.print(" status = ");
        Serial.println(status);
       }

    //Once params are extracted, we can release eeMLX90640 array

    MLX90640_I2CWrite(0x33, 0x800D, 6401);    // writes the value 1901 (HEX)
= 6401 (DEC) in the register at position 0x800D to enable reading out the
temperatures

    MLX90640_SetRefreshRate(MLX90640_address, 0x05); //setting to 16Hz


    pinMode(TFT_LED, OUTPUT);
    digitalWrite(TFT_LED, HIGH);

    tft.begin();

    tft.setRotation(1);

    tft.fillScreen(ILI9341_BLACK);
    tft.fillRect(0, 0, 319, 13, TFT_RED);
    tft.setCursor(100, 3);
    tft.setTextSize(1);
    tft.setTextColor(ILI9341_YELLOW);
    tft.print("Camara Termica foRESTER");

    tft.drawLine(250, 210 - 0, 258, 210 - 0, tft.color565(255, 255, 255));
    tft.drawLine(250, 210 - 90, 258, 210 - 90, tft.color565(255, 255, 255));
    tft.drawLine(250, 210 - 180, 258, 210 - 180, tft.color565(255, 255, 255));

    tft.setCursor(80, 220);
    tft.setTextColor(ILI9341_WHITE, tft.color565(0, 0, 0));
    tft.print("T(+) = ");


    // drawing the colour-scale
    // ========================

    for (i = 0; i < 181; i++)
       {
        getColour(i);
```

```
        tft.drawLine(240,  210  -  i,  250,  210  -  i,  tft.color565(R_colour,
G_colour, B_colour));
       }


  ///////BLE SETUP//////
  BLEDevice::init("Thermal Mapping"); // nome do dispositivo bluetooth
  // Create the BLE Server
  BLEServer *server = BLEDevice::createServer(); //cria um BLE server
  server->setCallbacks(new ServerCallbacks()); //seta o callback do server
  // Create the BLE Service
  BLEService *service = server->createService(SERVICE_UUID);
  // Create a BLE Characteristic para envio de dados
  characteristicTX       =       service->createCharacteristic(CHARACTERIS-
TIC_UUID_TX,BLECharacteristic::PROPERTY_NOTIFY);
  //BLEDescriptor *pDescriptor1 = new BLEDescriptor("temperature");
  characteristicTX->addDescriptor(new BLE2902());
  //characteristicTX->addDescriptor(pDescriptor1);
  /////
  characteristicRX       =       service->createCharacteristic(CHARACTERIS-
TIC_UUID_RX,BLECharacteristic::PROPERTY_NOTIFY);
  characteristicRX->addDescriptor(new BLE2902());
  ////



  service->start();
  // Start advertising (descoberta do ESP32)
  server->getAdvertising()->start();
  } //fim setup



// ********************************
// ************** LOOP **************
// ********************************

void loop()
   { //}//closed the loop (added)

  // void TaskCamera(void *pvParameters)  // This is a task. (added)
   // { //added
    //(void) pvParameters; //added
    //Serial.print("teste1");
    for (byte x = 0 ; x < 2 ; x++) //Read both subpages
       {
        uint16_t mlx90640Frame[834];
        int status = MLX90640_GetFrameData(MLX90640_address, mlx90640Frame);

        if (status < 0)
           {
            Serial.print("GetFrame Error: ");
            Serial.println(status);
           }

        float vdd = MLX90640_GetVdd(mlx90640Frame, &mlx90640);
```

```
        float Ta = MLX90640_GetTa(mlx90640Frame, &mlx90640);

        float tr = Ta - TA_SHIFT; //Reflected temperature based on the sensor
ambient temperature
        float emissivity = 0.95;

        MLX90640_CalculateTo(mlx90640Frame,   &mlx90640,   emissivity,   tr,
mlx90640To);
       }


    // determine T_min and T_max and eliminate error pixels
    // ==================================================

    mlx90640To[1*32 + 21] = 0.5 * (mlx90640To[1*32 + 20] + mlx90640To[1*32 +
22]);    // eliminate the error-pixels
    mlx90640To[4*32 + 30] = 0.5 * (mlx90640To[4*32 + 29] + mlx90640To[4*32 +
31]);    // eliminate the error-pixels

    T_min = mlx90640To[0];
    T_max = mlx90640To[0];

    for (i = 1; i < 768; i++)
       {
        if((mlx90640To[i] > -41) && (mlx90640To[i] < 301))
           {
            if(mlx90640To[i] < T_min)
               {
                T_min = mlx90640To[i];
               }

            if(mlx90640To[i] > T_max)
               {
                T_max = mlx90640To[i];
               }
           }
        else if(i > 0)   // temperature out of range
           {
            mlx90640To[i] = mlx90640To[i-1];
           }
        else
           {
            mlx90640To[i] = mlx90640To[i+1];
           }
       }


    // determine T_center
    // ==================

    T_center = mlx90640To[11* 32 + 15];
    T_medium = ((T_max + T_min)/2);
    // drawing the picture
    // ===================

    for (i = 0 ; i < 24 ; i++)
       {
        for (j = 0; j < 32; j++)
```

```
            {
             mlx90640To[i*32 + j] = 180.0 * (mlx90640To[i*32 + j] - T_min) /
(T_max - T_min);

             getColour(mlx90640To[i*32 + j]);

             tft.fillRect(217 - j * 7, 35 + i * 7, 7, 7, tft.color565(R_colour,
G_colour, B_colour));
            }
       }

    tft.drawLine(217 - 15*7 + 3.5 - 5, 11*7 + 35 + 3.5, 217 - 15*7 + 3.5 +
5, 11*7 + 35 + 3.5, tft.color565(255, 255, 255));
    tft.drawLine(217 - 15*7 + 3.5, 11*7 + 35 + 3.5 - 5, 217 - 15*7 + 3.5,
11*7 + 35 + 3.5 + 5,  tft.color565(255, 255, 255));

    tft.fillRect(260, 25, 37, 10, tft.color565(0, 0, 0));
    tft.fillRect(260, 205, 37, 10, tft.color565(0, 0, 0));
    tft.fillRect(115, 220, 37, 10, tft.color565(0, 0, 0));

    tft.setTextColor(ILI9341_WHITE, tft.color565(0, 0, 0));
    tft.setCursor(265, 25);
    tft.print(T_max, 1);
    tft.setCursor(265, 205);
    tft.print(T_min, 1);
    tft.setCursor(120, 220);
    tft.print(T_center, 1);
    tft.setCursor(265, 115);
    tft.print(T_medium, 1);

    tft.setCursor(300, 25);
    tft.print("C");
    tft.setCursor(300, 205);
    tft.print("C");
    tft.setCursor(300, 115);
    tft.print("C");
    tft.setCursor(155, 220);
    tft.print("C");

    delay(20);


    ////BLE COMMUNICATION/////
    if (deviceConnected) {
//recupera a leitura da temperatura do ambiente
float tempAmbiente = T_center;


char txString[20];
dtostrf(tempAmbiente, 2, 2, txString); // float_val, min_width, digits_af-
ter_decimal, char_buffer
strcat(txString, " Centro (C)");
characteristicRX->setValue(txString); //seta o valor que a caracteristica
notificará (enviar)
characteristicRX->notify(); // Envia o valor para o smartphone

char txString2[20];
dtostrf(T_max, 2, 2, txString2);
```

```
strcat(txString2, " Max (C)");
characteristicTX->setValue(txString2); //seta o valor que a caracteristica
notificará (enviar)
characteristicTX->notify(); // Envia o valor para o smartphone

    }
}



// ================================
// ===== determine the colour ====
// ================================

void getColour(int j)
   {
    if (j >= 0 && j < 30)
       {
        R_colour = 0;
        G_colour = 0;
        B_colour = 20 + (120.0/30.0) * j;
       }

    if (j >= 30 && j < 60)
       {
        R_colour = (120.0 / 30) * (j - 30.0);
        G_colour = 0;
        B_colour = 140 - (60.0/30.0) * (j - 30.0);
       }

    if (j >= 60 && j < 90)
       {
        R_colour = 120 + (135.0/30.0) * (j - 60.0);
        G_colour = 0;
        B_colour = 80 - (70.0/30.0) * (j - 60.0);
       }

    if (j >= 90 && j < 120)
       {
        R_colour = 255;
        G_colour = 0 + (60.0/30.0) * (j - 90.0);
        B_colour = 10 - (10.0/30.0) * (j - 90.0);
       }

    if (j >= 120 && j < 150)
       {
        R_colour = 255;
        G_colour = 60 + (175.0/30.0) * (j - 120.0);
        B_colour = 0;
       }

    if (j >= 150 && j <= 180)
       {
        R_colour = 255;
        G_colour = 235 + (20.0/30.0) * (j - 150.0);
        B_colour = 0 + 255.0/30.0 * (j - 150.0);
       }
```

```
    }


//Returns true if the MLX90640 is detected on the I2C bus
boolean isConnected()
    {
     Wire.beginTransmission((uint8_t)MLX90640_address);
     Serial.print("looking for i2c");
     if (Wire.endTransmission() != 0)
        return (false); //Sensor did not ACK

     return (true);
    }
```

Code for the Alarm Module

```
#include <TinyGPS++.h>
#include <SoftwareSerial.h>

/**/ float O2_val = 0;                /**/
/**/ const float VRefer = 3.3;             /**/
/**/ #define pinAdc 13 //analog pin

// The TinyGPS++ object
TinyGPSPlus gps;

static const int RXPin = 16, TXPin = 17;
static const uint32_t GPSBaud = 9600;

SoftwareSerial ss(RXPin, TXPin);

// BLE INCLUDES
#include <BLEDevice.h>
#include <BLEServer.h>
#include <BLEUtils.h>
#include <BLE2902.h>

BLECharacteristic *characteristicTX;
BLECharacteristic *characteristicRX;
BLECharacteristic *characteristicVX;


bool deviceConnected = false;
#define SERVICE_UUID "ab0828b1-198e-4351-b779-901fa0e0371e"
#define CHARACTERISTIC_UUID_RX "4ac8a682-9736-4e5d-932b-e9b31405049c"
#define CHARACTERISTIC_UUID_TX "3b475608-7762-4e82-b133-a9ff5ab38aa4"
#define CHARACTERISTIC_UUID_VX "967a147b-7a45-4dc7-a8d2-34b47abe3a67"



class ServerCallbacks: public BLEServerCallbacks {
    void onConnect(BLEServer* pServer) {
      deviceConnected = true;
```

```
    };

    void onDisconnect(BLEServer* pServer) {
      deviceConnected = false;
    }
};


void setup()
{
  Serial.begin(9600);
  ss.begin(GPSBaud);


  ///////BLE SETUP//////
  BLEDevice::init("Alarm module"); // nome do dispositivo bluetooth
  // Create the BLE Server
  BLEServer *server = BLEDevice::createServer(); //cria um BLE server
  server->setCallbacks(new ServerCallbacks()); //seta o callback do server
  // Create the BLE Service
  BLEService *service = server->createService(SERVICE_UUID);
  // Create a BLE Characteristic para envio de dados
  characteristicTX = service->createCharacteristic(CHARACTERIS-
TIC_UUID_TX,BLECharacteristic::PROPERTY_NOTIFY);
  characteristicTX->addDescriptor(new BLE2902());
  /////
  characteristicRX = service->createCharacteristic(CHARACTERIS-
TIC_UUID_RX,BLECharacteristic::PROPERTY_NOTIFY);
  characteristicRX->addDescriptor(new BLE2902());
  ////
  characteristicVX = service->createCharacteristic(CHARACTERIS-
TIC_UUID_VX,BLECharacteristic::PROPERTY_NOTIFY);
  characteristicVX->addDescriptor(new BLE2902());
  ////
  service->start();
  // Start advertising (descoberta do ESP32)
  server->getAdvertising()->start();
  } //fim setup


void loop()
{
  while (ss.available() > 0)
    if (gps.encode(ss.read()))
      displayInfo();

  if (millis() > 5000 && gps.charsProcessed() < 10)
  {
    Serial.println(F("No GPS detected: check wiring."));
    while(true);
  }
}


float readO2Vout()                                              /**/
{
    long sum = 0;                                             /**/
    for(int i=0; i<128; i++)                                        /**/
```

```
    {                                                       /**/
        sum += analogRead(pinAdc);                                  /**/
    }
    sum >>= 7;                                              /**/
    float MeasuredVout = sum * (VRefer /
1500.0);                                        /**/
    return MeasuredVout;                                        /**/
}
////////////////////////////////////////////////////////////////////////////
////////////////////////// /**/
float readConcentration()                                       /**/
{                                                   /**/
    float MeasuredVout = readO2Vout();                              /**/
    float Concentration = MeasuredVout * 0.205 / 3.1;       /**/
    float Concentration_Percentage=Concentra-
tion*100;                           /**/
    return Concentration_Percentage;                            /**/
}                                                   /**/
////////////////////////////////////////////////////////////////////////////
////////////////////////// /**/

void displayInfo()
{
  Serial.print(F("Location: "));
  float local = gps.location.lat();
  float local2 = gps.location.lng();
 // Serial.println(local);
  if (gps.location.isValid())
  {
    Serial.print(gps.location.lat(), 6);
    Serial.print(F(","));
    Serial.print(gps.location.lng(), 6);
  }
  else
  {
    Serial.print(F("INVALID"));
  }

  Serial.print(F("  Date/Time: "));
  if (gps.date.isValid())
  {
    Serial.print(gps.date.day());
    Serial.print(F("/"));
    Serial.print(gps.date.month());
    Serial.print(F("/"));
    Serial.print(gps.date.year());
  }
  else
  {
    Serial.print(F("INVALID"));
  }

  Serial.print(F(" "));
  if (gps.time.isValid())
  {
    if (gps.time.hour() < 10) Serial.print(F("0"));
    Serial.print(gps.time.hour());
    Serial.print(F(":"));
```

```
    if (gps.time.minute() < 10) Serial.print(F("0"));
    Serial.print(gps.time.minute());
    Serial.print(F(":"));
    if (gps.time.second() < 10) Serial.print(F("0"));
    Serial.print(gps.time.second());
    Serial.print(F("."));
    if (gps.time.centisecond() < 10) Serial.print(F("0"));
    Serial.print(gps.time.centisecond());
  }
  else
  {
    Serial.print(F("INVALID"));
  }

  float local3= readConcentration();
  ////BLE COMMUNICATION/////
    if (deviceConnected) {

    char txString[20];
    dtostrf(local, 2, 6, txString); // float_val, min_width, digits_af-
ter_decimal, char_buffer
    strcat(txString, " Lat ");
    characteristicRX->setValue(txString); //seta o valor que a caracteris-
tica notificará (enviar)
    characteristicRX->notify(); // Envia o valor para o smartphone

    char txString2[20];
    dtostrf(local2, 2, 6, txString2);
    strcat(txString2, " Long");
    characteristicTX->setValue(txString2); //seta o valor que a caracteris-
tica notificará (enviar)
    characteristicTX->notify(); // Envia o valor para o smartphone

    char txString3[20];
    dtostrf(local3, 2, 2, txString2);
    strcat(txString2, " %");
    characteristicVX->setValue(txString2); //seta o valor que a caracteris-
tica notificará (enviar)
    characteristicVX->notify(); // Envia o valor para o smartphone
    }
  Serial.println();
  }
```