José Pedro Marques Curva

Licenciado em Ciências da Engenharia
Eletrotécnica e de Computadores

Infrared Fire Alarm for Vehicle Protection

Dissertação para obtenção do Grau de Mestre em

Engenharia Eletrotécnica e de Computadores

Orientador: Prof. Dr. Nuno Filipe Silva Veríssimo Paulino,
Prof. Auxiliar,
Universidade Nova de Lisboa

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Setembro, 2019

Infrared Fire Alarm for Vehicle Protection

Este documento foi gerado utilizando o processador (pdf)LaTeX, com base no template "novathesis" [1] desenvolvido no Dep. Informática da FCT-NOVA [2].
[1] https://github.com/joaomlourenco/novathesis [2] http://www.di.fct.unl.pt

To my beloved family and my other-half Maria

Acknowledgements

Firstly, I would like to thank Prof. Nuno Paulino for the opportunity, commitment and support during this thesis, as well as help during the aforementioned.

I would also like to thank my friends Bruno Ambrósio and Miguel Castilho for the support given during the whole project and help solving some problems in the development of the thesis.

A special thanks to my girlfriend Maria for all the support, love and help during the whole university time, but especially during the thesis. The moral support given was essential to keep going during the hardest times and to give the best of me to this project. A very sincere thanks for believing in me when I couldn't believe in myself.

Last but not least, I thank my family as well. I thank my mother and my father for the effort they made during the past 5 years and the moral support they gave me to conclude my degree and keep positive.

Abstract

The occurrence of wildfires is a matter of preoccupation, specially in Portugal. In those disasters, many fire fighters are deployed to try put out the flames. However, when driving to the fire or away from it, the vehicles carrying the fire fighters might be in a dangerous situation, whether it is from being surrounded by flames or because of a drop in the levels of oxygen. The aftermath of the wildfires is also a focal point, since it's hard for the fire fighters to verify its perimeter looking for possible reignition points.

To overcome these problems, two modules were created. The first behaves like an alarm for fire fighting vehicles, verifying the surrounding temperatures and the oxygen levels inside the vehicle. If the vehicle is found in a dangerous situation, a message of emergency is sent to a central server, with the coordinates of the vehicle and the type of danger, in order to someone send help. The second module works like a scanner installed in a vehicle, collecting an array of temperatures in different positions, to analyse the perimeter of the wildfire and detect possible reignition points.

This project will help central communication stations to acknowledge the safety state of fire fighting vehicles, as well as to have a more efficient aftermath analysis of wildfires.

Keywords: Wildfire Prevention, Infrared Thermal Sensor, Arduino, Oxygen Sensor, GNSS Receiver.

Resumo

A ocorrência de incêndios florestais é um assunto de preocupação, especialmente em Portugal. Durante estes desastres, vários bombeiros são chamados a combater as chamas. No entanto, na deslocação para o local dos incêndios ou na vinda do mesmo, os veículos que transportam os bombeiros podem deparar-se com situações de perigo, seja por ficarem cercados pelas chamas ou devido a baixos níveis de oxigénio dentro do veículo. O rescaldo de incêndios é também um ponto de foco, devido à dificuldade dos bombeiros verificarem todo o perímetro do incêndio à procura de focos de reacendimento.

Com vista a ultrapassar estes problemas, foram desenvolvidos dois módulos. O primeiro funciona como um alarme para veículos de combate a incêndios, verificando as temperaturas à sua volta e os níveis de oxigénio no seu interior. Se for detetada uma situação de perigo associada ao veículo, uma mensagem de emergência é enviada para um servidor central, com as coordenadas do veículo e a situação de perigo detetada, de modo a que alguém possa enviar ajuda. O segundo módulo funciona como um sistema de varrimento de imagens instalado num veículo, recolhendo matrizes de temperaturas em diferentes posições, permitindo analisar e detetar possíveis focos de reacendimento no perímetro do incêndio.

Este projeto ajudará as estações centrais de comunicação a terem conhecimento do estado de segurança dos veículos de combate a incêndios, assim como a realizar um rescaldo dos incêndios mais eficiente.

Palavras-chave: Prevenção de Incêndios Florestais, Sensor Térmico Infravermelho, Arduino, Sensor de Oxigénio, Receptor GNSS.

Contents

List of Figures

List of Tables

Listings

Glossary

Central Processing Unit: The component of a computer responsible for controlling data and other component's activities, by executing instructions and logical operations.

Development board: A printed circuit board including a MCU, I/O pins, LEDs and buttons designed to allow easy experimentations and creations with a certain MCU.

Integrated Development Environment: Set of programming tools and menus for coding.

Microcontroller Unit: A single chip containing a processor, a non-volatile memory for programs, a volatile memory for data processing, a clock and a controller unit for input and output. It is created to have a single and very specific purpose.

Sensor: A device capable of detecting and responding to certain environmental stimulus.

Acronyms

ADC Analog-to-Digital Converter

AGC Automatic Gain Controller

BoM Bill of Materials

BPF Band-Pass Filter

CPU Central Processing Unit

CSV Comma-Separated Values

DGPS Differential GPS

DIY Do It Yourself

DLL Delay Lock Loop

EGNOS European Geostationary Navigation Overlay System

FLL Frequency Lock Loop

FOV Field of View

GAGAN GPS-Aided Geo Augmented Navigation

GLONASS Global Orbiting Navigation Satellite System

GND Ground

GNSS Global Navigation Satellite System

GPIO General Purpose Input/Output

GPS Global Positioning System

HDMI High Definition Multimedia Interface

I/O Input/Output

I2C Inter-Integrated Circuit

IDE Integrated Development Environment

IF Intermediate Frequency

IoT Internet of Things

IR Infrared

LED Light-Emmitting Diode

LNA Low Noise Amplifier

LO Local Oscillator

MCU Microcontroller Unit

MEO Medium Earth Orbit

MSAS Multi-functional Satellite Augmentation System

NaN Not a Number

NAV Navigation Message

NVM Non-Volatile Memory

OLED Organic Light-Emitting Diode

PLL Phase Lock Loop

PRF Pulse Repetition Frequency

PRN Pseudo-Random Noise

PVT Position, Velocity and Time

RADAR Radio Detection And Ranging

RAM Random Access Memory

RF Radio Frequency

SAR Synthetic Aperture Radar

SBAS Satellite Based Augmentation System

SBL Static Base Location

SCL Serial Clock Signal

SDA Serial Data Signal

SoC System-on-a-Chip

SPIFFS Serial Peripheral Interface Flash File System

UART Universal Asynchronous Receiver-Transmitter

UAV Unmanned Aerial Vehicle

USB Universal Serial Bus

V_{CC} Voltage Common Collector

WAAS Wide Area Augmentation System

*

Chapter 1

Introduction

This chapter is to contextualize the purpose of this thesis. Its motivation will be explained firstly, followed by the target goals of this work.

1.1 Context and Motivation

During the last 15-20 years, fighting wildfires turned into a matter of major concern, with a large quantity of wildfires starting from non-natural causes. In June 2017, there was a huge wildfire in Pedrógão Grande, that killed more than 60 people (including fire fighters and villagers), injured more than 250 people and burnt around 50 thousand hectares of forest. This was a big pivotal point in taking drastic measures against wildfires, since the results of this fire were catastrophic, as well as scary [1, 2].

One of the first problems is the management in fire fighting. In the wildfire of Monchique, in 2018, some fire fighters reported that the management of people and machines to control the fire was a total disaster [3]. This poor management can put lives of people fighting the wildfire in question if help doesn't arrive when they're in a danger situation, like being in a vehicle surrounded by flames.

Also in the wildfire of Monchique in 2018, Portugal asked the European Union to use the satellite Copernicus to map the national territory, in order to obtain more information about the size of the fire, as well as where the main heat sources were [4]. This method is very effective, but may take some time to be able to use the satellite data for better management. However, this approach is not viable in the case of analysing the aftermath of the fire, since some possible reignition points might not be in the form of a visible flame.

1.2 Target Goals and Problem Approach

The goal of this thesis is to provide protection to the vehicle from the flames by adding an array of sensors that can detect said flames around the vehicle and being able to look for possible reignition points during the aftermath of the wildfire. The approach to this problem start s with the creation of a model that uses Infrared (IR) thermal sensors and Global Positioning System (GPS) coordinates to control the surroundings of the vehicle and map the perimeter of the wildfire. This way, a larger area of the fire can be covered much faster during the aftermath, as well as giving much more reliable and important information about the state of the region where the fire occurred. It will also allow to give a more efficient protection to the fire fighters located inside a vehicle in a situation of emergency. The module also needs to be relatively cheap, in order to produce it in large quantities and apply in many vehicles, without large amounts of money being spent. The third requirement of the module is to have an easy maintenance so, in case something breaks, the costs of repairing are kept low.

The module will be constituted by the following sub-modules:

- **Global Navigation Satellite System (GNSS) Receiver:** Will be used to receive the location of the user, in order to correctly locate a specific thermal data in the map. The location data will be sent to the Microcontroller Unit (MCU) via serial communication, like Inter-Integrated Circuit (I2C) or Universal Asynchronous Receiver-Transmitter (UART);

- **IR thermal sensor:** The temperature data of a certain object needs to be obtained somehow. This module will do so for different ambient points detected, with a certain period. The data obtained will be sent to the MCU via I2C.

- **Display:** In order to give more detailed information about the received data, a small Organic Light-Emitting Diode (OLED) display will be part of the module. The data received in the MCU will be displayed in a little OLED screen, making it easier to debug the module if any problems occur;

- **WiFi Connection:** The information received in the MCU needs to be sent to a server, so that it can be seen by the entities responsible for fire fighting management. Using WiFi connection, the MCU will be able to connect to a web server, delivering the information received by its sensors without the need of internet cables, making the module more versatile;

- **MCU:** Used to make everything work together. It will receive the data from both the GNSS receiver and the IR sensor, run the programmed script for the module and send the data received to the display, as well as the result of the script to a web server.

In Figure 1.1 is a simplified representation of the circuit to be implemented. Further, in the Chapter 3, will be shown the choices made to each one of the components, as well as the main reasons that support those choices.

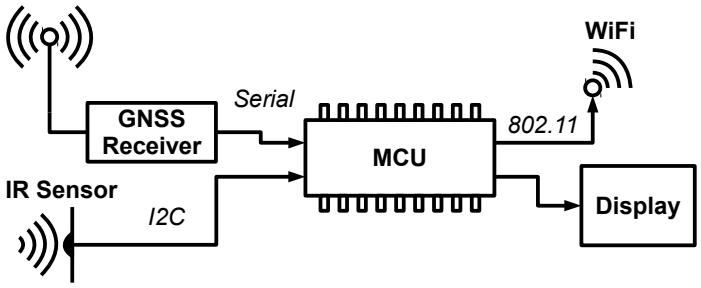


Figure 1.1: Simplified representation of the circuit to be implemented.

1.3 Main Contributions

This thesis will have a big impact regarding the safety of fire fighters and fire fighting vehicles, as well as being a great contribution to map wildfire perimeters.

Keeping fire fighters safe and aware of the surrounding dangers is critical. This project allows to inform the occupants of the vehicles in the area of the fire, showing them values of the temperatures around the vehicle and oxygen levels inside the car. This project is also capable of being applied in other areas, like monitoring if the engine of a vehicle has enough oxygen to start, which will prevent its occupants from being caught by the flames.

Another area of contribution is to protect the trailing machines used during the aftermath of the fire. Those trailing machines clear some zones where the fire occurred, preventing the propagation of possible reignition points. The alarm function of this project is also applicable to those vehicles, giving them protection and keeping central services informed about their state.

Regarding the mapping capabilities, this project will allow a fast identification of possible reignition points. By equipping a fire truck with the module capable of acquiring thermal pictures of the perimeter, and along side the already used trailing machines, the aftermath process of wildfires will be much faster and under control. When a fire truck detects a heat source in real time using the mapping module, it starts the process to put the fire out as fast as possible.

By applying the thermal mapping process during the fight of a wildfire, it will allow real time imaging of the perimeter, unlike satellite imaging, which is only available once every few days. These satellite images, unlike the data obtained using the result of this thesis, aren't real-time images and are quite heavier, regarding the size of the necessary data. The data transmitted using the module developed in this project is send in packets

with the size around a few bytes or kilobytes, while imagery from satellites are around megabytes, gigabytes or more.

Chapter 2

State of Art

This chapter provides an introduction about the modules used in this thesis and some environmental imaging techniques. The objective of this chapter is to give some background knowledge and information about how energy transmits through radiation, the components of a GNSS and the difference between active and passive sensors.

It will also be explained how some of the components of the module presented in the previous chapter, like the IR thermal sensor, the GNSS receiver and the MCU.

A section will be dedicated to Radio Detection And Ranging (RADAR) imaging which, despite not being directly present in this thesis, has a working principle similar to the one used to image fires using the module developed in this thesis.

2.1 Radiation Energy

When thermal radiation strikes an object, it is split into three components: transmitted radiation, absorbed radiation and reflected radiation. Each one of this components has a coefficient, with a range from 0 to 1, that defines how much radiation of such type is created by the object, and they are dependent on various aspects [5]. Figure 2.1 displays a representation of the various radiation components when an object receives electromagnetic energy.

Transmitted radiation energy depends on the composition of the object. Translucent objects like glass or crystal allow more energy through them, representing a higher transmission coefficient. On the other hand, opaque objects have a lower transmission coefficient [5].

Absorbed radiation energy is what makes the temperature of an object fluctuate. More absorbed energy means a higher temperature of the object. An object that absorbs all the energy is called blackbody and has an absorption coefficient of 1. In reality, blackbodies

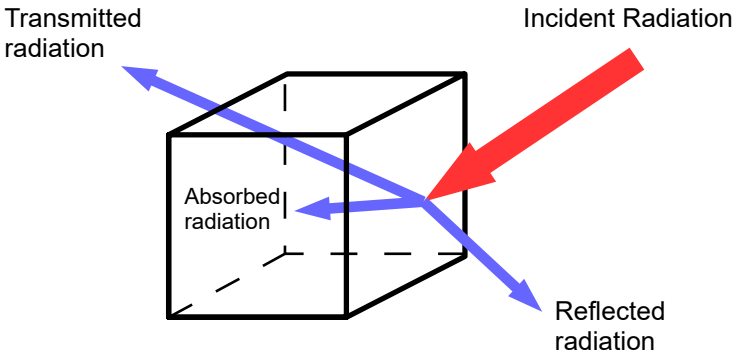


Figure 2.1: Visual interpretation of the effect of incident radiation energy on an object.

don't exist. Therefore, the absorption coefficient of a body falls between 0 an 1, without ever reaching the maximum value [5].

Reflected radiation energy depends on the object's surface brightness. Surfaces more polished and shinier have a higher reflection coefficient than rough and mate surfaces [5]. The concept of an IR thermal sensor is based on the fact that if a body has a temperature above 0ºK, it emits energy in the form of radiation [5]. This energy can be represented by [6]:

$$P = e\sigma A(T_{obj}^4 - T_{amb}^4) \tag{2.1}$$

Where

P = energy radiated by a body, W;
e = emissivity of the object radiating;
σ = Stefan-Boltzmann constant = $5.6703 \times 10^{-}8$, W/m^2K^4;
T_{obj} = Temperature of the object radiating, K;
T_{amb} = Temperature of the surrounding environment, K;

Emissivity e is what defines the object's radiation capabilities [7]. It has a range from 0 to 1, with the greater value meaning it's a better emitter. When an object has an emissivity of 1, is also considered to be a blackbody [8].

2.2 Active Sensors and Passive Sensors

The technology solutions used nowadays to detect and monitor wildfires are mainly divided into two types: the ones based on active sensors (i.e. RADAR) and the ones based on passive sensors (i.e. IR sensors). Active sensors use a transmitter to send a signal to the environment and a receiver to detect the echo of the same signal. This echo is the signal reflected from the objects in the area illuminated by the sensor, which arrives with different characteristics when compared to the original signal. This allows to perceive

some characteristics of the target like its aspect, size and temperature. A passive sensor doesn't need a transmitter, since it receives signals of natural sources like environmental radiation [9].

Active sensors operate with wavelengths ranging from *cm* and upwards which means they are less affected by weather factors. It also means that they are used for long distance detection [9]. However, if the time between consecutive transmitted signals is too small, it creates aliasing, where the first echoes of the second transmitted signal arrive earlier to the receiver than the later echoes from the previous transmitted signal. This means that, the further away the target is, the lower the Pulse Repetition Frequency (PRF) has to be [10]. In [11] is an example of an active sensor applied in fire detection and monitoring.

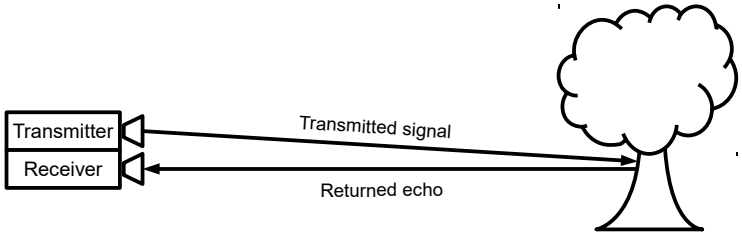


Figure 2.2: Principle of an Active Sensor.

Passive sensors use smaller wavelengths, around the μm's, making these sensors more suitable for short and medium range. Because of the smaller wavelength, they are also more affected by the weather condition and the atmosphere. Compared to the active sensors, passive sensors don't have the problem of signal aliasing [9]. In [12] is a project of a system for fire detection using a passive sensor.

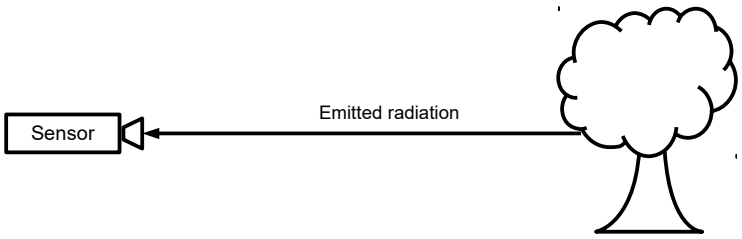


Figure 2.3: Principle of a Passive Sensor.

2.3 IR Thermal Sensor

2.3.1 What is an IR Thermal Sensor?

An IR thermal sensor, also known as IR thermometer, is a sensor that measures the temperature of an object by receiving its radiant energy [13]. As mentioned in the Section 2.1, when an object is hit by radiation energy, some part is absorbed, other part is transferred through the object and the rest is reflected by the object [14]. The reflected radiation from the object comes in the form of electromagnetic waves, the majority of them in

the IR region of the spectrum. This waves are caught by the sensor, which turns the electromagnetic wave signal into electrical energy and then into digital data [5].

2.3.2 How does the IR Thermal Sensor work?

The IR thermal sensors use the fact that the radiation emitted by an object is correlated the the 4^{th} power of its temperature, as shown in the equation (2.1), to obtain that value.

The radiated energy is received by the sensor through a lens that focus and filters the IR waves, in order to obtain the ones inside the wavelength range desired. The emitted radiation spectrum includes wavelengths from 1 mm to 0.7 μm. However, to obtain better results, systems using IR thermometers only use signals with wavelengths between 14μm and 7 μm, since that is the bandwidth where the strongest radiation occurs and where the atmospheric attenuation is the smallest [5]. Then, an IR detector receives the filtered radiation thermal energy and converts it to an electric signal. This signal is around the microvolt range, so a high-gain amplifier is needed to increase the signal strength. Despite the treatment that the wave got from the lens, the signal still has some fluctuations caused by the ambient temperature. Therefore, a signal-conditioning module is placed after the amplifier, stabilizing the temperature values and linearising the current signal. Last but not least, an Analog-to-Digital Converter (ADC) is placed, in order to turn the electrical signal into digital readable data. Hand-held models may contain a display to show the temperature from the ADC [5, 15]. In the Figure 2.4 we can see how the different modules inside the sensor are connected. Note that this scheme doesn't include a display or screen, since it's not a fundamental module for the sensor to work.

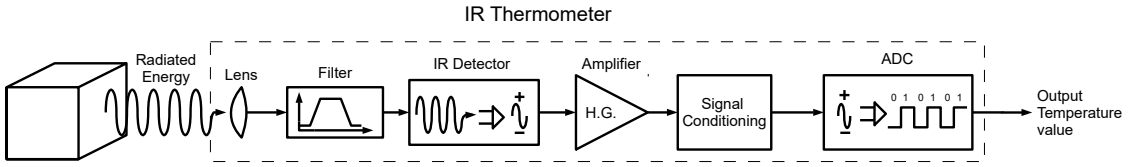


Figure 2.4: Scheme of an IR thermal sensor.

There are many different aspects to define an IR thermal sensors, like the number of wavelength bands, range of the wavelength bands and type of detector [5].

The IR detector can use single or double wavelength range. The difference is that in single wavelength, the detector measures the energy in a certain moment, with a pre-set emissivity and in one wavelength range only. Double wavelength uses two different readings in two different ranges, giving a more accurate reading when there's a big gap in the measurement of the emissivity or the energy emitted. The actual value read is the ratio between readings. Despite the greater accuracy, this type of wavelength range can be less accurate under certain circumstances and is more expensive than the single wavelength [5, 7].

Some sensors use a narrower range, some use a wider range. Narrow ranges are used to measure objects that have big variations in emissivity, like when there's smoke or steam covering the object. Wide ranges have less accuracy, but have a greater range of measured temperatures, which is useful when trying to detect hot objects in cold environments or vice versa [5].

Referring to IR detectors, there are 2 types of them: photodiodes and thermal detectors. A photodiode is a PN-junction diode that produces current from a light source, by letting go electrons when hit by photons. The more light received by the photodiode, the greater the flow of electrons will be [5, 16]. Thermal detectors change its properties depending on the incident radiation [5].

There are 3 types of thermal detectors:

- **Thermopile:** Generates electrical energy from thermal energy. Is composed of a pair of thermocouples connected in series or parallel. Is used when the output of a single thermocouple isn't large enough [17];
- **Bolometer:** Measures the intensity of the radiated energy with a varying resistance [18];
- **Pyroelectric:** Generates electric potential when heated or cooled [19].

2.3.3 Choosing the ideal IR Thermal Sensor

There are many different IR thermal sensors, from simple circuit modules up to high tech cameras and hand-held sensors. The price can vary a lot, with chipsets and modules being as cheap as 10€-50€, some hand-held modules around hundreds of euros, and the best cameras costing up to thousands of euros.

When looking for an IR thermal sensor, some specifications are more important than others. Some of the most important specs are [5, 13]:

- **Field of View (FOV):** How wide the sensor "sees";
- **Distance of measuring:** How far the sensor can measure;
- **Spectral band:** The range of wavelengths that the sensor can measure;
- **Accuracy:** How precise are the sensor's measurements;
- **Response time:** How fast the sensor can turn the radiation into an electrical signal;
- **Object temperature range:** The range of temperatures it can measure.

2.4 GNSS Receiver

2.4.1 What is a GNSS Receiver?

A GNSS receiver, usually just called GPS receiver, is a navigation device, used to obtain the current location of the device itself. The device has the ability to receive data from any of the satellites belonging to a constellation. After receiving this information, the device calculates its distance to a few satellites of the same constellation (usually 4 minimum). With the distance to each of the chosen satellites, the receiver has an area of its possible locations. By intercepting the area of possible locations to at least 4 satellites, the receiver gets its exact location on Earth. This process is called trilateration [20].

2.4.2 Satellite Constellation

A satellite constellation is a group of orbiting satellites positioned in a certain way, in order to achieve an objective by working together [21]. Therefore, a navigation constellation is a group of satellites used for navigation, working together to give receivers their exact positioning. Usually, navigation satellites are located in the Medium Earth Orbit (MEO), which is at an altitude of approximately 20,000 km, meaning these satellites aren't geostationary [22]. There are many navigation constellations, some regional and some global. The global navigation constellations are the most used, and some examples of them are:

- **Global Positioning System (GPS):** Founded in the USA, has 31 operational satellites, with at least 24 of them available for 95% of the time [22]. Has an accuracy of approximately 5 meters [23].
- **Global Orbiting Navigation Satellite System (GLONASS):** Founded in Russia, has 24 operational navigation satellites [24]. Has an accuracy of approximately 5 to 7 meters [23].
- **Galileo:** Founded in Europe. Is under civil control, unlike GPS or GLONASS, which are both under military control [25]. The team in charge of the Galileo GNSS is aiming to reach a constellation of 30 satellites by 2020, with the first 2 being sent to orbit in 2011 [26]. Has an accuracy of about 1 meter [23].

2.4.3 Trilateration

In order to obtain a reliable location, as mentioned in the Section 2.4.1, the receiver needs to connect to, at least, 4 satellites. Trilateration is the process of intercepting data received from each of the satellites, in order to obtain the correct location of the receiver.

When thinking about the 2-D location of something, 3 circular area would be enough to define the location of the receiver. However, the receiver needs to know its altitude too. In order to do so, a fourth satellite is used to define the altitude.

The way the sensor determines its distance to a satellite is using the data sent by each satellite. This data is sent in the form of a radio signal and is called Navigation Message (NAV) [23].

2.4.4 Navigation Message (NAV)

Each satellite is continuously sending a group of data, called NAV. The NAV contains different types of information, including the position of the satellite, atmospheric corrections to the signal, an almanac with estimated positioning of neighbour satellites of the constellation and a signal that identifies the satellite called Pseudo-Random Noise (PRN) [23]. The PRN code is a code that is continuously sent in the NAV. Each satellite has its own PRN code, which allows the receiver to identify the satellite sending the information. This code allows the receiver to measure its distance to the satellite, since the code has a period small enough to repeat it self many times per second. The way it works is simple: both the receiver and the transmitter start the code at the same time; when the receiver gets the code sent by the satellite, it arrives with a certain delay [20, 23]. Since the signal travels roughly at the speed of light in vacuum ($c = 299,792,458\ m/s$), the receiver can find the distance to the satellite in question, based on the equation

$$v = d/t \tag{2.2}$$

where

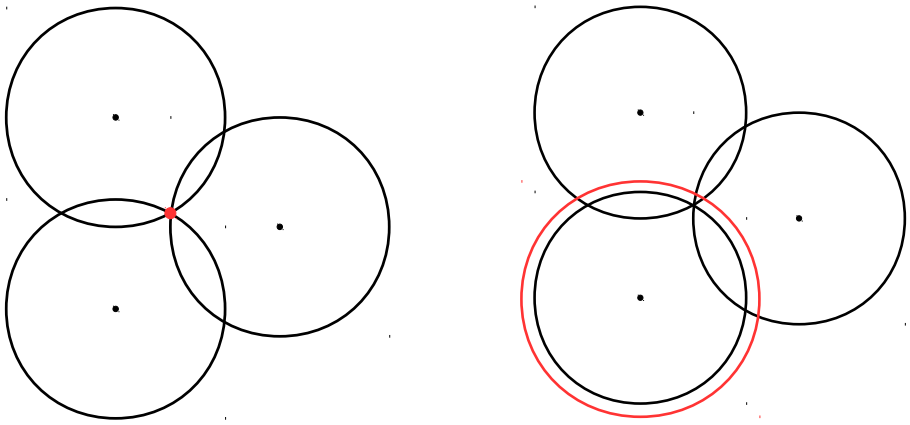
v = velocity, m/s;
d = distance travelled, m;
t = duration of travelling, s;

Knowing the distance to the satellite and its location, is possible for the receiver to perform the trilateration, explained in the Section 2.4.3.

2.4.5 Differential GPS (DGPS)

As seen in previous chapters, some errors can occur while the satellite transmits the signal to the receivers. Although satellites send atmospheric corrections, other errors can occur, like the deflection of signals in buildings, or the weather affecting the signal velocity. However, there is a system to help correct positional errors in GPS receivers, called Differential GPS (DGPS) [20, 27]. Considering a two-dimensional trilateration with 3 satellites, if there are no delays, the circles of positioning relative to each one of the satellites cross at exactly one point, like in the Figure 2.5a. But if the signal of one of the satellites has some unexpected delay during transmission, the three circles won't cross in any point, as shown in the Figure 2.5b.

The basic concept to correct positional errors is by getting the position of a nearby known location. These locations can be either a Static Base Location (SBL) or geostationary



(a) (b)

Figure 2.5: Two-dimensional comparation of trilateration with 3 satellites without delays (a) and with an unexpected delay marked in red (b).

Based on: [20, 27]

satellites in the form of Satellite Based Augmentation System (SBAS). SBL are stations located on Earth. These stations know their exact locations, so they can correct errors by themselves, by constantly getting their location from satellites. The station then calculates the difference between its correct location and the one obtained by satellite signals. This difference is then broadcasted to all the receivers equipped with DGPS, in order to obtain a more accurate positioning. SBAS is very similar to SBL, being composed by a set of geostationary satellites which receive the average correction in their area sent by base stations. These satellites then send the corrections to DGPS receivers, which can calculate their position more accurately. SBAS corrections aren't as accurate as SBL, but are faster and easier to use. SBAS is the term used for this type of systems in general. However, there are different names for the regional systems like [27]:

- **Wide Area Augmentation System (WAAS):** Covers the area of the United States of America.
- **European Geostationary Navigation Overlay System (EGNOS):** Covers the area of Europe, some part of Africa and a couple of spots in America;
- **Multi-functional Satellite Augmentation System (MSAS):** From Japan, operates in Asia;
- **GPS-Aided Geo Augmented Navigation (GAGAN):** Operates in the region of India.

2.4.6 GNSS Receiver Architecture

The generic architecture of a GNSS receiver is composed by an Antenna, a Radio Frequency (RF) Front End, a Baseband Processing module and an Application Processing

module [28].

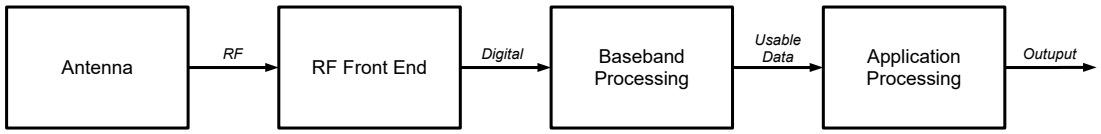


Figure 2.6: Generic architecture of a GNSS receiver.

Based on: [28]

- **Antenna:** Is composed by the antenna itself and a Low Noise Amplifier (LNA). Since the signal received by the antenna comes in RF, it has very low power. So the amplifier right after the antenna allows for a better reading of the signal [28];

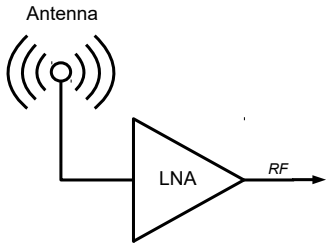


Figure 2.7: Generic architecture of the Antenna module of a GNSS receiver.

Based on: [28]

- **RF Front End:** This block receives the amplified RF signal. This signal comes with noise and unwanted frequencies. So, firstly, there is a Band-Pass Filter (BPF), which allows the receiver to process only the desired frequencies of the signal. Then, to compensate any transmission losses, the BPF is followed by a LNA. After this process, the signal needs to be down-shifted in frequency. This is called down-conversion, where the signal is multiplied by a signal from a Local Oscillator (LO), in order to have a signal with Intermediate Frequency (IF) instead of RF. When the 2 signals are multiplied in time domain, it means they are convoluted in the frequency domain. The product of the two signals is a representation of the signal centred in 2 different frequencies: the sum of both signal, and the difference of both signals. After this process, the signal is filtered again, allowing only the lower-frequency representation of the signal. This step can be repeated as many times as the manufacturer wants [29].

  Then, the circuit is followed by an Automatic Gain Controller (AGC). This is critical in RF circuits because the signal's energy decreases as the distance between the receiver and the transmitter increases. Therefore, the signal strength fluctuates a lot, and needs to be as constant as possible, in order for the signal processing be as efficient as possible [30]. The AGC does so, allowing the ADC to successfully output the IF signal to digital;

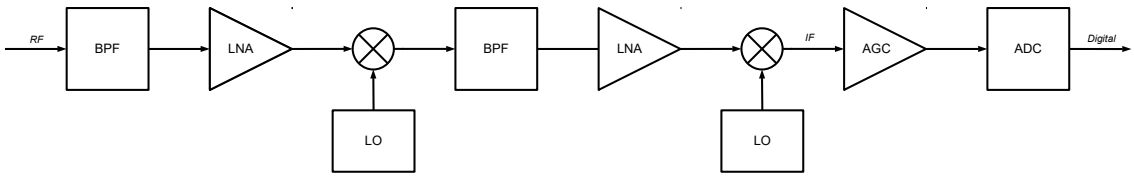


Figure 2.8: Generic architecture of the RF Front End module of a GNSS receiver.

Based on: [29]

- **Baseband Processing:** The main objective of this block is to acquire each signal and track its code delay and carrier phase measurements [28]. As mentioned in the Section 2.4.4, each satellite has its own PRN, which is true for satellites from GPS or Galileo constellation. So, in order to keep track of each satellite, receivers have a specific channel for each different satellite, running different PRN codes. Constellations like GLONASS use the same PRN code for every satellite, but with different frequencies. So channels associated to GLONASS satellites compare the frequency of the signal as well as the PRN [31]. This allows the receiver to calculate the signal delay of a specific satellite [32]. First, the signal passes through a Doppler frequency removal module, since the signal suffers from Doppler effect caused by the satellite movement. Then, there's a block which correlates the received signal with the code calculated by the receiver. This allows the receiver to verify if the estimated PRN code delay and Doppler frequency are correct [31]. Then, the incoming signal is sent to a Delay Lock Loop (DLL), a Phase Lock Loop (PLL) and a Frequency Lock Loop (FLL), to verify and adjust the PRN code delay, the carrier signal misalignment and Doppler frequency misalignment respectively, produced by the signal generator [33]. This estimation is sent back to both the Doppler removal block and the local PRN code generator block from time to time [31]. Before sending out the result signal, the incoming signal goes through a demodulating and processing block, which basically decomposes the full message into small separated messages that include information like the location of the satellite, atmospheric delays, and so on [31];

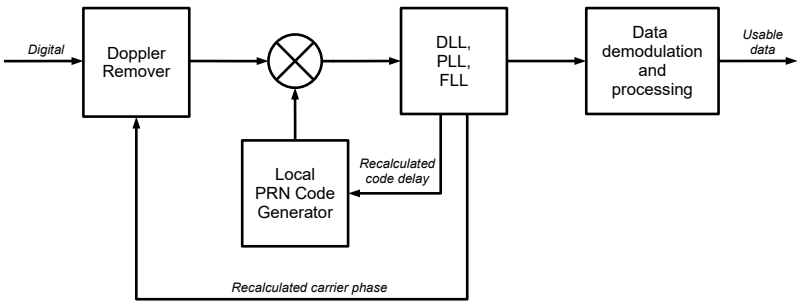


Figure 2.9: Generic architecture of the Baseband Processing module of a GNSS receiver.

Based on: [31, 32]

- **Application Processing:** Here, the receiver extracts the message from the Baseband module of each channel in order to calculate relevant information like the Position, Velocity and Time (PVT). Depending on the purpose of the receiver, other calculations can be made with PVT or other data included in the message [28].

2.4.7 Choosing a GNSS Receiver

Like it happens when choosing an IR Thermometer, there are also lots of GNSS receivers. Some of them are more expensive, some cheaper, but all have a few characteristics worth looking at, like [34]:

- **Precision:** How precise is the receiver when calculating its position;
- **Acquisition time:** The time it takes for the receiver to acquire data from the satellite. There are 3 types of signal acquisition:
  - **Cold start:** In this case, the receiver doesn't have any type of information related to current time or satellite positions;
  - **Warm start:** The receiver knows the current time and satellite positions, but doesn't know which satellites are in its sight;
  - **Hot start:** This situation is the ideal case for the receiver, where it knows the current time, the position of the satellites and the satellites in its sight.
- **Sensitivity:** How sensible the receiver is to catch the signal when acquiring for the first time, reacquiring and tracking the signal;
- **Power Consumption:** How much current the receiver asks of the power source. Since it's a small circuit, its consumption is around the milliamperes;
- **DGPS:** Sensors equipped with DGPS systems have higher accuracy;
- **Receiver version:** Like happens with everything in technology, new versions of receivers come once in a while. Sometimes it's not worth to take the latest version of a GNSS receiver, and previous versions are usually a lot cheaper than newer versions.

2.5 Microcontroller Unit (MCU)

2.5.1 What is a MCU?

A MCU is a type of computer, contained inside a single chip and with a specific objective. Like a desktop computer or a laptop, a MCU contains [35, 36]:

- **Central Processing Unit (CPU):** Used to execute the programs saved in the MCU;

- **Non-Volatile Memory (NVM):** Capable of saving data without being connected to a source of power, holds data for the programs to be executed. Can be ROM, EEPROM or Flash memories [37];
- **Internal Clock:** To keep everything in synchrony;
- **Random Access Memory (RAM):** Unlike NVM, RAM isn't capable of holding memory without a source of power. However, is very useful to hold variables;
- **Input/Output (I/O) pins:** In order to communicate with the real world, the MCU needs an I/O interface, whether it is to receive instructions form a keyboard, receive data from sensors or send instructions to actuators.

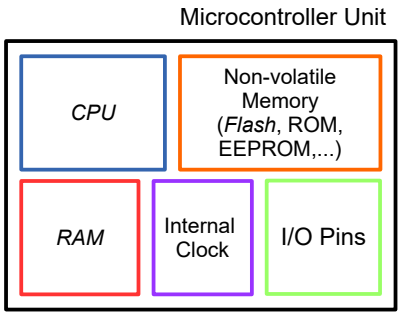


Figure 2.10: Generic architecture of a MCU.

However, unlike desktop and laptop computers, MCUs have some special characteristics [35]:

- **Embedded system:** Usually, MCUs are embedded in other circuits or devices. For development of devices related to the Internet of Things (IoT), we can find MCUs included in what is called System-on-a-Chip (SoC). Some examples of programmable SoCs used to create small devices are the Arduino Uno, the NodeMCU and the Raspberry Pi. The latter is more powerful that the other two, hence being used to more complex functions. However, the Arduino and the NodeMCU are very popular for creating devices that interact with sensors;
- **Single Purpose:** This type of computers are made to have a single purpose. It can be to control a DVD player or be part of the engine control unit of a car as an example, but they are not expected to run certain programs used in desktops, like Microsoft Office;
- **Low Power Consumption:** Since they are a single chip and used for a specific operation, MCUs are built to run on low power. They can consume power around the milliWatts, unlike desktops, which require much more power;
- **Small Size and Low Cost:** MCUs are usually used in many products, so they are made to be as cheap as possible. Small size comes from the fact that MCUs are a

single chip used to run a single purpose, and since they are usually within in other devices, smaller MCUs reflect on more compact devices.

2.5.2 Some MCUs on the market

There are some MCUs in the market right now that are seen as the ones to watch. Some of them are made by companies with more experience on this market and have been producing controlling units for a long time, others are newer in this field and are trying to compete by bringing modern and/or cheaper concepts. Looking to the development of small devices and Do It Yourself (DIY) projects some known brands are [38]:

- **Arduino:** Probably the number 1 name when it comes to development boards, Arduino has over 10 different types of boards for sale and lots of add-on modules. Arduino development boards prices start at around €20 new, but in 2^{nd} hand shops the prices can get even lower. The base model Arduino Uno is perfect for small projects which don't demand complex circuits. Alongside the whole set of boards and modules, Arduino offers a free Integrated Development Environment (IDE), allowing its customers to write and compile their Arduino codes easily [39];

- **Raspberry Pi:** Raspberry Pi created their first prototype in 2006, launching the first board in 2012, being a fairly recent company [40]. One of the main goals of this company is to make computer boards accessible to everyone, which require the creation of powerful but cheap boards. Raspberry already has at least 4 versions of the board: the Pi Zero, the Pi 1, the Pi 2 and the Pi 3. The Raspberry Pi 3 is the most recent and has a lot of good features like 1GB of RAM, microSD slot, High Definition Multimedia Interface (HDMI) port, WiFi module, 40 I/O pins and 4 Universal Serial Bus (USB) ports [41]. Alongside all this, the prices are below €40, which is very cheap for such computational power [42];

- **NodeMCU:** This project started with the objective of upgrading a chipset called ESP8266, which is a very small but very capable controlling board. The two most attractive characteristics of this chipset are its size (around the same size of a coin) and its capability of WiFi connection. In order to make the ESP8266 more user-friendly, a team found a way to program the ESP8266 through the Arduino IDE. Later, the NodeMCU project was created, making the ESP8266 into a single board, with additional components like more I/O pins and a microUSB to serial connection [43]. The price of the NodeMCU is also very appealing, since they are one of the cheapest boards (if not the cheapest) in the market, with prices below the €5 barrier [44]. Small size, free IDE for coding, Arduino compatibility, WiFi connection and cheap prices make the NodeMCU a strong competitor in the market of development boards for small and simple task devices.

2.6 Synthetic Aperture Radar (SAR)

2.6.1 What is a SAR?

A Synthetic Aperture Radar (SAR) is a system used for of ground imaging, where a RADAR with smaller aperture is linked to an airplane or a satellite and collects information along a certain distance. By putting the RADAR in motion, it is possible to cover a bigger distance than it's possible with a stationary RADAR, giving it a "synthetic aperture" greater than its real aperture [45]. In the Figure 2.11 is a scheme of a SAR system. The SAR is flying at a given altitude with a constant velocity. The SAR flies along the azimuth axis, perpendicular to the range axis. The RADAR has a certain angle θ_{inc} relative to the altitude axis. The ground range is the range from the azimuth axis to the center of the range. The RADAR has a range aperture of θ_{range}, which defines the swath of the imaging, and an azimuth aperture θ_{azi}, which defines the azimuth resolution of the RADAR. The range coordinate closer to the azimuth axis is called near range, while the further range coordinate from the azimuth axis is the far range. The swath is the width of the image caught by the RADAR on the ground. The azimuth resolution is the length along the azimuth axis caught by the RADAR with a certain PRF [45, 46].

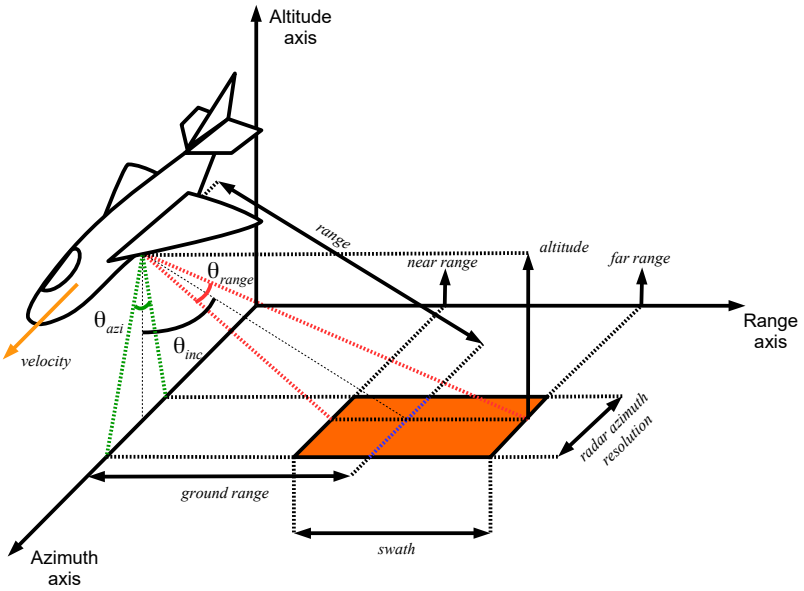


Figure 2.11: Imaging geometry of a SAR system on an airplane.

The azimuth resolution of the RADAR is small, usually with a narrow range, due to it being dependent on the antenna size and signal frequency. The resolution is increased by increasing the size of the antenna or by increasing the frequency of the signal. Since the antenna size is limited by the size of the airplane/space shuttle and the frequency can't be very high not to suffer signal losses, the resolution of the RADAR is necessarily small in the azimuth. However, the azimuth resolution of the whole image is increased by moving the RADAR in a forward motion, catching a greater ground area. This is, effectively, the

synthetic aperture of the RADAR [46].

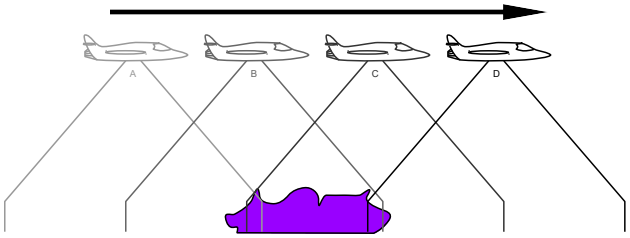


Figure 2.12: Visual representation of the SAR imaging process of an object on the ground using an airplane.

Based on: [47]

As the Figure 2.12 shows, when the RADAR flies over an object, its aperture covers the object in different positions. So, in order to obtain a correct imaging of the object and with a good resolution, the different signals need to be processed together [47].

2.6.2 SAR Modes

There are different ways for the SAR to work while airborne. The 3 most common modes are [45]:

- **Stripmap:** The distance between the near range and the far range is constant during the whole trajectory, allowing the SAR to make an image with a good azimuth resolution and length. The greater the frequency of PRF without originating aliasing, the better is the resolution. This is the most common mode as well as the simplest. This process can be visualized in the Figure 2.13;

- **Scan:** In this mode, the antenna changes its incidence angle periodically, in order to obtain an image larger in the range axis, as represented in the Figure 2.14. However, this comes with the cost of a worse azimuth resolution when compared with the Stripmap mode, since the illumination period of each footprint on the ground is smaller and the footprint is changing along two axis;

- **Spotlight:** The Spotlight mode works similarly to the Scan mode, but instead of varying the incident angle relative to the range axis, it varies the incident angle relative to the azimuth axis. What this means is that the SAR basically fixes a point in the ground and keeps sending signals to the same point, as it moves along the azimuth axis. The result of this mode is a image with great resolution, given the long period of imaging the same spot. However, the resulting image is smaller in the azimuth axis than the one obtained through the Stripmap mode. In the Figure 2.15 is a representation of the principle of the Spotlight mode.

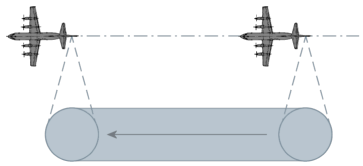


Figure 2.13: Representation of the SAR Stripmap mode.

Source: http://www.radartutorial.eu/20.airborne/ab08.en.html

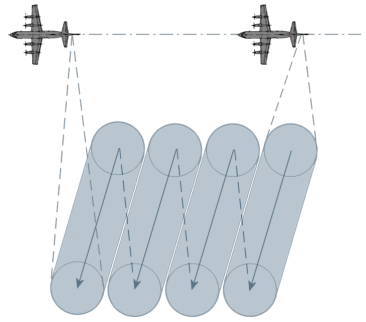


Figure 2.14: Representation of the SAR Scan mode.

Source: http://www.radartutorial.eu/20.airborne/ab08.en.html

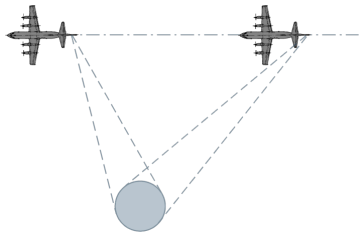


Figure 2.15: Representation of the SAR Spotlight mode.

Source: http://www.radartutorial.eu/20.airborne/ab08.en.html

2.6.3 Applications of SAR

SAR imaging has both military and civilian applications. Some of those applications are [48]:

- **Battlefield recognition:** The military services use SAR imaging to acquire more knowledge of big battlefield areas;
- **Mining and Geology:** SAR imaging allows to find deposits of minerals of the ground;
- **Fire detection:** SAR is able to identify any source of fire almost in real time. It is also capable of detecting fire sources at night with a temperature greater than 39ºC and with high accuracy. In fact, the RADAR used in the SAR detects the smoke produced by the fire and no the flames or heat of the fire. This is due to the smoke being able to backscatter the signal emitted by the RADAR to the surface [49]. In

the Figure 2.16 is possible to see the resulting image of a SAR applied to a wildfire scenario;

- **Evaluation and monitoring of agricultural crops:** Different types of crops can identified using SAR imaging, as shown in the Figure 2.17;
- **Oil Spill Monitoring:** SAR can see through the water, which is good to identify such disasters like oil spills like in the Figure 2.18.

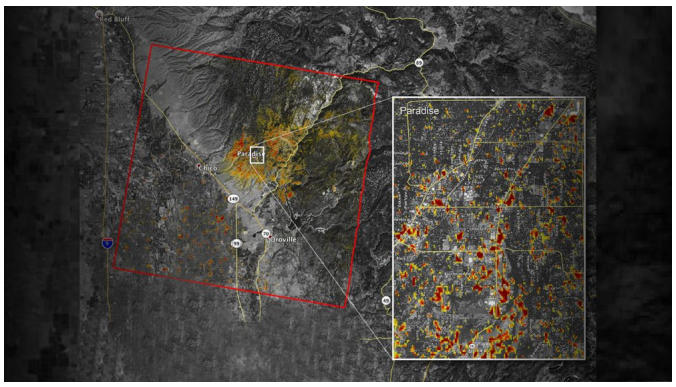


Figure 2.16: Example of wildfire monitoring using SAR.

Source: https://gogeomatics.ca/devastating-california-wildfires-sar-images-from-the-copernicus-sentinel-1/

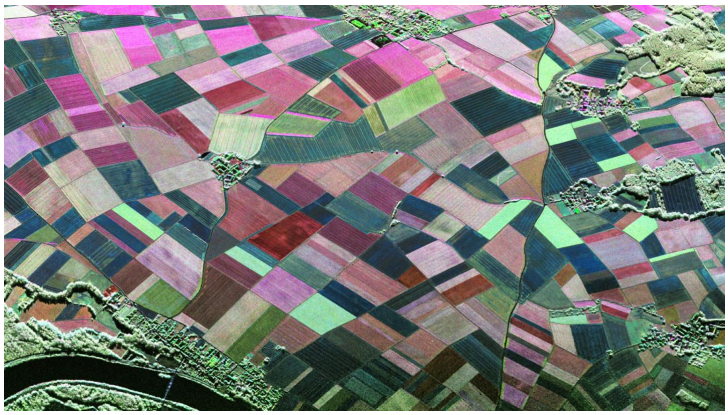


Figure 2.17: Example of agriculture crops monitoring using SAR imaging.

Source: https://www.dlr.de/dlr/en/desktopdefault.aspx/tabid-10293/427_read-11609/year-all/#/gallery/16494

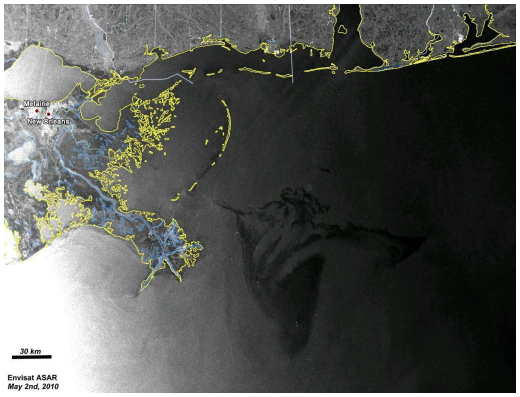


Figure 2.18: Example of oil spill monitoring using SAR imaging.

Source: http://www.esa.int/spaceinimages/Images/2010/05/Radar_image_acquired_on_2_May

Chapter 3

Hardware Selection

This chapter is dedicated to explain the structure of development of the project. The different steps in the project will be explained.

3.1 Development Plan

The first thing to notice is that the module will have two main functions. The first and main function is to work as an alarm module. The second function is to work similar to a SAR system, in order to help spot locations with high chance of reignition during the in-field aftermath analysis of the fire.

This division within the project was made to develop the module faster and more easily, as well as to not compromise the development of a certain function of the module. This way, if problems occur during the project, it's possible to focus in creating one fully functional module with only one function instead of developing a non-functional module with different possible functions.

However, the module is going to be based on the one presented in the Chapter 1. The components will vary depending on the needs for the task. It's also worth mentioning that the coding will be made through the Arduino IDE, which means that the chosen components will preferably have Arduino libraries already developed and/or coding examples. This will make it easier to understand how the components work, as well as making the code simpler.

3.2 Alarm Module

The alarm module is the part of the project that will perceive the dangers surrounding a fire fighting vehicle, as well as the conditions inside.

This module will measure the temperature outside of the vehicle, the oxygen levels inside the vehicle and collect the GPS coordinates of the current position. If the module detects dangerous levels of temperature surrounding the vehicle and/or dangerous levels of oxygen concentration in the air, the alarm will send the values to a server, as well as the current GPS coordinates of the vehicle and a warning message with the type of problem occurring in the referred vehicle.

In order to learn better how each component works, the module was split into four versions. The first version will be simpler, while the latest ones will be more complex and closer to the final complete module.

3.2.1 First Alarm Version

The first version will be composed only by an IR thermal sensor and an MCU, since those are two of the most important components.

The MCU chosen is the ESP8266, included in a development board called NodeMCU ESP-12e. This is a very capable development board for the task, very compact, cheap and it comes equipped with an integrated WiFi module. Since the task that the MCU is going to perform is not very complex, a more powerful development board like the Raspberry Pi was out of question because it is much more expensive and requires more power. When compared to an Arduino board in the same level of the NodeMCU, the latter outperforms the Arduino one, especially in the domains of processing speed, price and WiFi capability.

As for the IR thermal sensor, the module to be used is the MLX90614 series. This sensor measures ambient and object temperature with one pixel, comes factory calibrated, has a FOV of 90º and has a precision of ±0.5 ºC. The current consumption in these sensors is less than 23 mA, which is very important to create a small and compact device.

The circuit will be powered through a microUSB port within the ESP-12e development board, with a voltage of 5 V. The communication between the sensor and the development board will occur through I2C. In the Figure 3.1 is displayed the scheme for the first prototype of the alarm module.

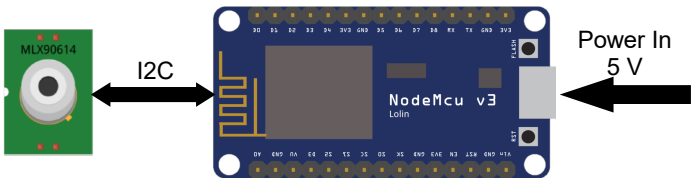


Figure 3.1: First Alarm Module version.

3.2.2 Second Alarm Version

For the second version of the alarm, a GNSS receiver is added to the previous version, as a form of evolution. Obtaining the GPS coordinates of the vehicle is also very important

for the alarm, therefore the presence of this module in the second stage of this module.

The chosen GNSS Receiver is the NEO-7M series module. This module comes equipped with the u-blox 7 chipset, an antenna, a signal processing unit and a UART connection. The reasons to choose the GNSS module over just the chipset are the price (which is fairly low for both of them), not to have to create a signal processing module for the GNSS receiver and, lastly, the fact it comes with a UART connection. This way, it is possible to use the data acquired by the u-blox 7 much more easily. This GNSS receiver is very popular in many compact DIY projects related to Earth positioning. The current consumption of this module is also very low (around 20 mA), which also fits the low consumption requirement of the module.

In the Figure 3.2 is shown the scheme for the second version of the alarm module, an evolution from the previous one.

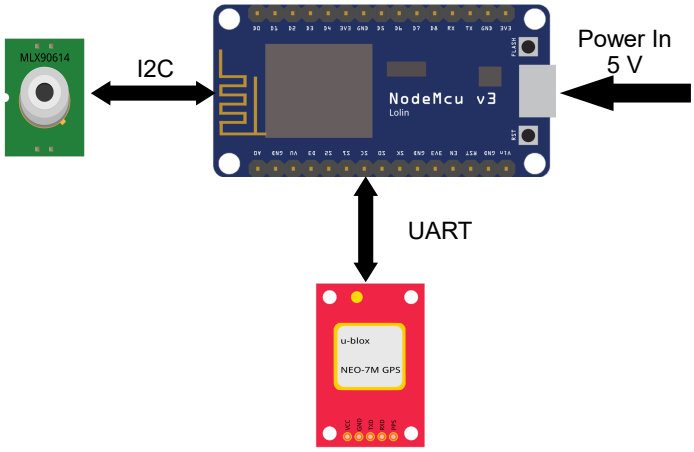


Figure 3.2: Second Alarm Module version.

3.2.3 Third Alarm Version

During wildfires, the oxygen levels can get dangerously low. So, it's necessary to keep the firemen alert for this invisible problem. The solution is to equip the module with an O_2 sensor. This sensor is able to measure the concentration of O_2 in the air. When this concentration reaches levels below safe, the alarm goes off, sending a warning message regarding the levels of O_2 inside the vehicle.

The used O_2 sensor will be the ME2-O2 by Seeed. This is a compact sensor that's capable of measuring O_2 levels between 0% and 25%. It communicates through an analog port, similar to a potentiometer. This O_2 sensor comes calibrated from factory, which makes it easier to use. It also doesn't need much heating time, since the sensor correctly reads values after a few seconds of heating up. This sensor is quite costly, with prices within 50€ and 60€, but is one of the few available in the market, having a size and performance that fits the purpose of the alarm. The Figure 3.3 shows the scheme of the third evolution of the module.

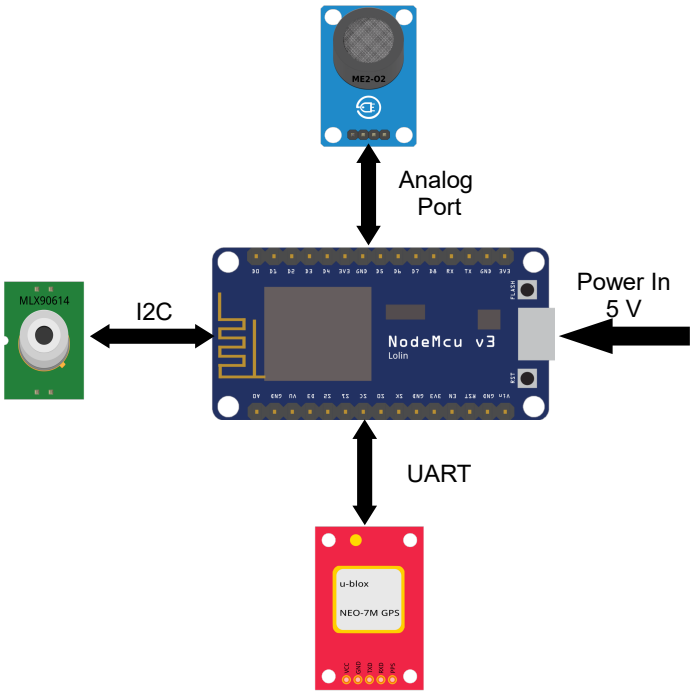
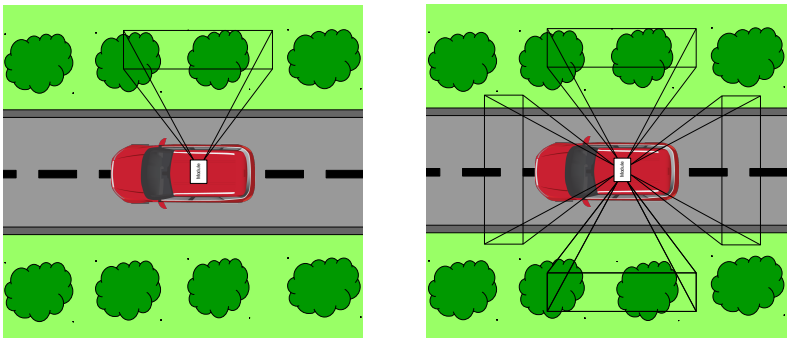


Figure 3.3: Third Alarm Module version.

3.2.4 Fourth Alarm Version

The last planned evolution for the alarm is the closest to a working functional fire alarm. The module will have four MLX90614 IR thermal sensors (three more than the previous version), one ME2-O2 oxygen sensor, one u-blox NEO-7M GNSS receiver, one NodeMCU ESP-12e development board and one generic 0.96" OLED display.

As shown in the Figure 3.4a, using just one IR thermal sensor, only one side of the vehicle is under vigilance of the module. The additional thermal sensors will allow coverage from both sides, front and back of the vehicle, as represented in the Figure 3.4b.



(a) (b)

Figure 3.4: Coverage of the vehicle using one sensor (a) and using four sensors (b).

The OLED display will be implemented to allow the users to receive information regarding the values of the sensors. For instance, if the vehicle is in a dangerous situation,

a message is displayed at the same time that the alarm goes off, communicating with the server. In the case of low oxygen levels, it's very important that the user is informed, since it's a danger that is not visible and is hard to detect by human sense.

In the Figure 3.5 is shown the scheme of the fourth and last planned evolution of the alarm module.

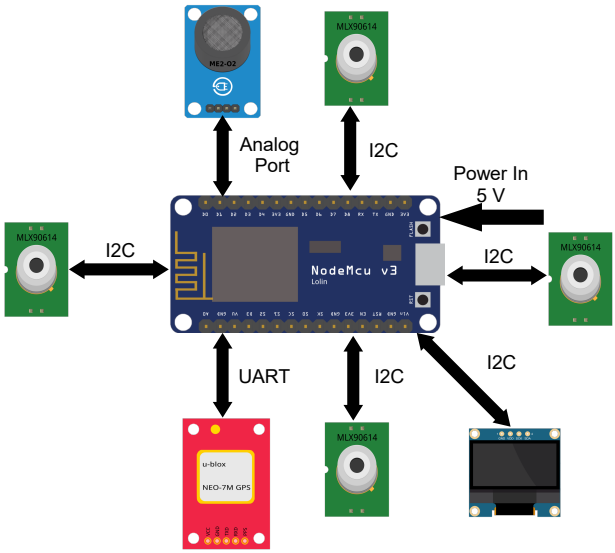


Figure 3.5: Fourth Alarm Module version.

3.3 Thermal Mapping Module

The Thermal Mapping module will be important to check possible reignition points after the wildfire being put out. Some bits of wood, bushes, trees and other combustible materials in the area might not be on fire, but they can be at very high temperatures, making them possible points of reignition. To provide better control over the conditions of the fire affected area, the aforementioned points need to be detected before they ignite. To do so, the area needs to be scanned using IR thermal sensors, to mark possible dangerous points by collecting the GPS coordinates of the place.

3.3.1 Scanning Strategy

The principle for scanning fires is identical to the one used in SAR systems. The vehicle equipped with the module will travel along a road, with the sensors fixed to the car. If the vehicle goes on a perfectly straight road, it will work like the Stripmode in SAR systems explained in the Section 2.6.2.

However, roads aren't always straight and information may overlap. When certain points in the map are overlapped, there are two possibilities: the first is to ignore one of the assigned values, keeping the remaining one; the second one is to do the mean between the values read.

Unlike a SAR system, a IR thermal sensor is going to be used instead of a RADAR antenna. This means that the Doppler effect won't occur while reading temperatures, since the IR thermal sensor is a passive sensor. This is a great advantage because the signal received doesn't need Doppler correction.

Another great advantage is the size of the sensor relative to the RADAR antenna. The RADAR antenna is huge, with dimensions in the range of metres. On the other hand, the IR thermal sensor is very small, having dimensions as small as 5 cm^2.

One thing to take into account is the time between two consecutive readings of the environmental temperature. Roads aren't always straight: some spots can be read more than one time (like the inside of a corner), others can be missed (like the outside of a corner). The second situation is not good, unlike the first situation, which is indifferent. Being unable to read a certain area means that a reignition point might be missed.

The frequency of reading needs to be high enough not to miss spaces, but not too big, to avoid excessive overlapping. The ideal frequency would be the one that wouldn't leave blank spaces nor overlap. This would be possible if the road was always straight. With curvy roads, it's necessary to ensure that no blank spaces will happen, without caring too much about overlap. The frequency of reading is going to be directly related to the speed of the vehicle and will depend on the processing power of the module. High vehicle speeds allow to cover a perimeter faster, but may create blank spaces between readings. Slow vehicle speeds avoid blank spaces, but result in a slow reading, which might not be good to cover large areas. A combination between the processing speed of the module and stable readings is the best way to go, by adapting the vehicle speed to the capabilities of the MCU.

3.3.2 Thermal Mapping Module Scheme

This module will be composed of less components than the alarm, but the general view of the module will be very similar. Since the majority of the components to be used have already been tested, this prototype will have just one evolution.

The module will have one multi-pixel IR thermal sensor, one MCU built in a development board, one GNSS receiver and one OLED display. The GNSS receiver and OLED display will be similar to the ones implemented in the Sections 3.2.2 and 3.2.4, respectively.

The development board will be the ESP32 DevKitC, by Espressif. This board is an evolution of the one used in the Alarm Module, the ESP-12e. This upgraded development board is important because the complexity of the module is increasing. Since one of the objectives is to merge the Alarm Module with the Thermal Mapping Module and to keep the whole module working fine, the MCU needs to be capable to coordinate all the components. In the Table 3.1 is a comparison between the two development boards. The ESP32 DevKitC is far more superior than the ESP-12e and, with similar prices, it is worth making the upgrade. The most important characteristics to take into account are

the number of General Purpose Input/Output (GPIO) ports, the CPU clock speed and the number of I2C and UART ports.

Table 3.1: Comparison of specifications between ESP32 DevKitC and NodeMCU-12e.

[50–52]

Development Board (SoC)	**ESP32 DevKitC (ESP32-WROOM-32)**	**ESP-12e (ESP8266)**
CPU	Tensilica Xtensa LX6 32 bit Dual-Core at 160/240 MHz	Tensilica L106 32 bit at 80 MHz (up to 160 MHz)
SRAM (KB)	520	36
Flash (MB)	4 (Max: 64)	4 (Max: 16)
Average Operating Current (mA)	80	80
Operating Voltage (V)	2.7 ~ 3.6	2.5 ~ 3.6
Wi-Fi	Yes	Yes
Bluetooth®	Yes	No
GPIO	32	17
I2C	2	1
UART	3	2
Average Price (€)	8	6

The IR thermal sensor used will be the Melexis MLX90640, a multi-pixel sensor based on the MLX90614 single-pixel IR thermal sensor used in the Alarm Module. Having more pixels allow for a more accurate thermal detection, since more sections of environment temperatures will be detected by a single sensor, in a similar range of a one-pixel sensor. The MLX90640 can measure temperatures ranging from -40 ºC to 300 ºC, which is a lower range when compared to the MLX90614 (from -70 ºC to 380 ºC). However, this loss in range is compensated by the greater number of pixels in the sensor. The MLX90640 has a 32x24 array of pixels, resulting in a resolution of 768 pixels, as shown in the Figure 3.6.

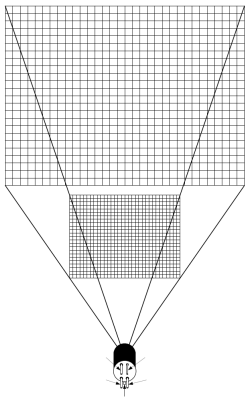


Figure 3.6: Field of View of the MLX90640.

Source: [53]

The chosen version of the MLX90640 comes with a FOV of 55ºx35º, which is the

narrower version. This version will be more precise than the wide version because it has the same number of sensors for a smaller aperture. However, the width of the FOV will be smaller, covering less distance at a time.

The interface of communication is I2C, just like with the MLX90614. Factory calibration and current consumption bellow the 23 mA are two very appealing factors of this sensor [53]. With prices very similar to the MLX90614, the MLX90640 is a very good option for the module in development. The main unknown question with both sensors is the processing speed, which might be much slower in the MLX90640 compared to the MLX90614. The Table 3.2 is a summary comparison of the specs of both IR thermal sensors.

Table 3.2: Comparison between the MLX90614 and the MLX90640 IR thermal sensors.

	MLX90614	**MLX90640**
Resolution	1 pixel	24x32 pixel array
Object Temperature (ºC)	-70 ~380	-40 ~300
Precision (ºC)	±0.5	±1.0
Factory Calibration	Yes	Yes
Current Consumption (mA)	23	23
Price (€)	~37	~41

In the Figure 3.7 is displayed the scheme for the Thermal Mapping Module. It's possible to see that, in the visual aspect, is a much simpler circuit than the one from Section 3.2.4. However, its software complexity might not be so different.

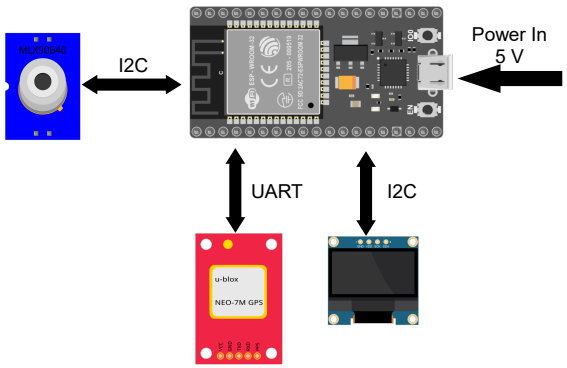


Figure 3.7: Thermal Mapping Module scheme.

Chapter 4

Project Development

In this chapter, the software development and respective testing will be reported. Some changes and difficulties may be noted, as well as some work for further development. Some of the characteristics explored might not be implemented yet, but will possibly be in a close future. It's worth mentioning that, for the scheme figures, the colours of the wiring are as follows:

- **Red:** Represents Voltage Common Collector (V_{CC}), whether it is 3.3 V or 5 V;
- **Black:** Represents 0V or Ground (GND);
- **Orange:** Represents connections from the transmitter pin of a component to the receiver pin of the development board;
- **Yellow:** Represents connections from the receiver pin of a component to the transmitter pin of the development board;
- **Blue:** Represents Serial Clock Signal (SCL) connections;
- **Green:** Represents Serial Data Signal (SDA) connections;
- **Brown:** Represents analog connections;
- **Purple:** Represents digital connections, mainly Light-Emmitting Diode (LED) connections.

4.1 IR Single-Pixel Thermal Sensor

The first components available were the NodeMCU ESP-12e development board and the Melexis MLX90614 IR thermal sensor. With the 2 components, some characteristics of both the development board and the IR thermal sensor were explored.

To obtain IR thermal data, the two components were connected as shown in Figure 4.1. The MLX90614 contains two pwer-related pins and two communication pins. The power-related pins are for the 3.3 V source and the GND and they are connected to the 3.3 V output voltage and the GND pins of the ESP-12e, respectively. The two communication pins of the IR thermal sensor are used to send the SCL and the SDA, since the communication protocol used by the sensor is the I2C. The SCL and SDA pins of the sensor are connected to the D1 and D2 pins of the ESP-12e, respectively.

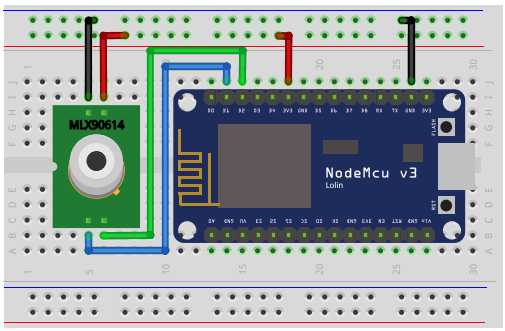


Figure 4.1: Diagram of the circuit for thermal data acquisition.

After connecting both components, the Arduino IDE was set up. This included the installation of the necessary drivers and libraries to program the development board and obtain data from the IR sensor.

Adafruit, a company specialized in hardware development and coding, created a dedicated library for the MLX90614 to be used in the Arduino IDE called "Adafruit_MLX90614.h". This library comes with a coding example to obtain data from the sensor, showed in Listing I.1, and was used to verify the connection between the IR thermal sensor and the development board. In the code below it's possible to see that the sensor measures two types of temperatures. The ambient temperature is the one surrounding the sensor, while the object temperature is the one perceived from where the sensor is pointing to. This library also makes the conversion of the values to degrees Celsius and Fahrenheit. Throughout this project, the value mostly used is going to be the object temperature, in degrees Celsius.

When setting up the MLX90614 to run in the Arduino IDE, the first thing to do is to declare a variable of type "Adafruit_MLX90614", shown in the Listing 4.1.

Listing 4.1: MLX90614 initialization.

```
4 Adafruit_MLX90614 mlx = Adafruit_MLX90614();
```

When declaring a variable of type "Adafruit_MLX90614", an optional parameter can be passed. This optional parameter is the address of the sensor. If the address isn't declared, the program will assume that the sensor has its standard factory address, which is 0x5A. This is an important aspect for later development, since the module will run

many sensors of the same type with the same address from factory. Some of the sensors will have their address changed to a different one, as it's explored in Section 4.3, in order to identify each sensor separately.

To initialize the sensor, the begin command is called, as showed in Listing 4.2, inside the setup function of the Arduino code.

Listing 4.2: MLX90614 begin command.

```
9  mlx.begin();
```

This function is only called once for each sensor. This is necessary for the sensor to start sending data from the environment to the MCU in the development board.

In the loop function is where the data is received from the sensor. In the Listing 4.3 is shown how to output to the serial monitor the ambient temperature (line 14) and the object temperature (line 16) in degrees Celsius.

Listing 4.3: MLX90614 readings in Celsius.

```
14  Serial.print(mlx.readAmbientTempC());
15  [...]
16  Serial.print(mlx.readObjectTempC());
```

Similarly, in the Listing 4.4, is shown how to output to the serial monitor the ambient temperature (line 18) and the object temperature (line 20) in degrees Fahrenheit.

Listing 4.4: MLX90614 readings in Fahrenheit.

```
18  Serial.print(mlx.readAmbientTempF());
19  [...]
20  Serial.print(mlx.readObjectTempF());
```

4.2 Sending Data to Web Page

In order to analyse the data received during a period of time, it has to be sent to the computer through a web page. To do so, firstly, the data needs to be saved to the memory of the development board. Then, the stored data is sent to a web page, available for the user to download in a text file. Then, the data is put into an Excel sheet to display in a graph.

To save data to the flash memory of the NodeMCU, it is used a file system called Serial Peripheral Interface Flash File System (SPIFFS). This file system gives access to the flash memory of the board and allows the user to create, read and write to its memory. To used the capabilities of the SPIFFS in the Arduino IDE, a library called "FS.h" is used.

The aforementioned library allows to create new files, read files and write to files inside the flash memory of the development board. To save the data to a text file, it's necessary to create a folder inside the Arduino project folder, called "data". This data folder is where the files that are going to be uploaded to the flash memory are put into. After creating this folder, the desired files to send to the flash memory are put into the folder. Next step, is to upload the files to the memory. The "FS.h" library gives an option in the Arduino IDE to upload data files from a certain project to the memory of the board. After uploading all the necessary files, it's possible to use the library functions to read from and write to files. In the Listing I.2 is the representation of the code found in [54]. This code was used as a base to create a program that saves data received by the IR thermal sensor to the SPIFFS. The specific code runs for around 20 seconds, fetching data with a frequency of 100 milliseconds after pressing a button. After those 20 seconds, the system saves the data to the SPIFFS and then idles. The code is represented in the Listing I.3. The diagram used to run this program is represented in the Figure 4.2.

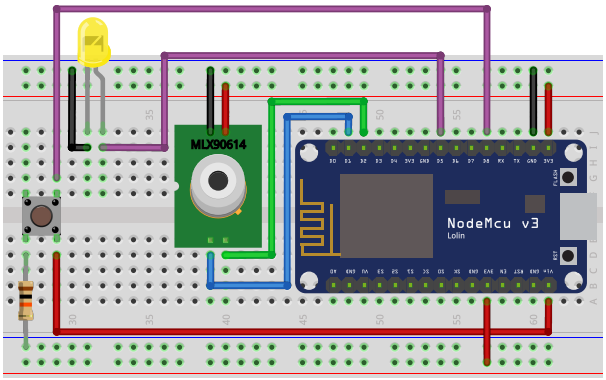


Figure 4.2: Diagram of the circuit for saving thermal data using SPIFFS.

To show the data in a web page, the functionality of the SPIFFS was combined with the capability of the NodeMCU to connect to the internet. This development board comes equipped with a WiFi module, allowing the board to connect to the internet. The base to display the data in a web server is an example from the library "ESP8266WebServer.h", one of the four libraries necessary to run the code. The remaining libraries used were the "ESP8266Wifi.h", the "WifiClient.h" and the "ESP8266mDNS.h". The coding example used is shown in the Listing I.4. To show the text file containing the data from the sensor, the only thing missing was to create an handle routine to write the data from the memory of the development board to the web page. This routine was created based on a solution presented in [55]. The resulting program is shown in Listing I.5.

4.3 Address Changing

The ESP-12e development board comes with an I2C channel, which allows to connect multiple devices using such protocol. The working principle is simple, and can be used

to receive data from devices (reading) or control other devices (writing). In the case of reading a sensor, the master device (in this case, the development board) sends a request to a slave device (which can be any connected sensor) with a specific address to read from a certain register, identified with the slave's address as well. Then, the master device connects to the slave device asking it to write the data in the SDA line, by sending the slave's address and setting the Read/Write bit to 1, which means the master will be reading the SDA line. Then, the slave device starts sending data for the master device to read. In the Figure 4.3 is a visual representation of the process of a master device reading a slave device using I2C.

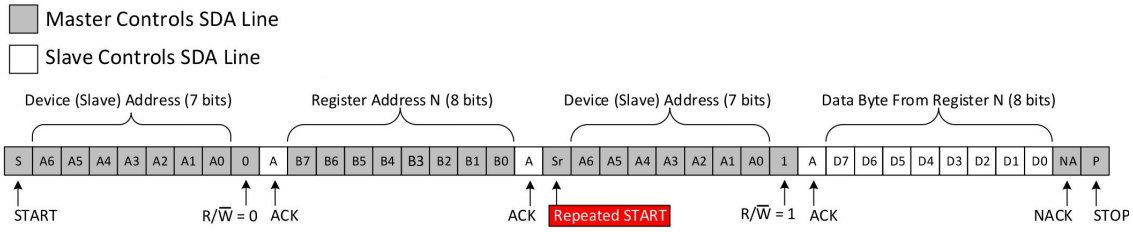


Figure 4.3: Time diagram of a master device reading a slave device's register using I2C.

Source: [56]

However, the ESP-12e only has one I2C channel, which means that two different sensors cannot be connected to the same channel using the same address. In the Figure 4.4 is an example of how to connect two sensors that communicate through I2C. As mentioned in the Section 4.1, the two defined ports of the ESP-12e for the SCL and SDA are the D1 and D2 pins, respectively. Therefore, any sensors that use I2C to communicate to the ESP-12e have to be connected to these two ports.

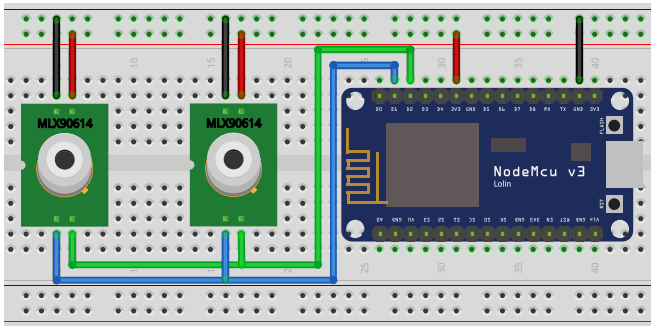


Figure 4.4: Example of two sensor connecting to ESP-12e using I2C communication.

As mentioned in the Section 4.1, the MLX90614 IR thermal sensor comes with a factory address, which in this case is 0x5A. Since there's only one I2C channel available and all MLX90614 sensors come with the same address from factory, the addresses of the used MLX90614 sensors need to be changed to different values from each other, in order to the development board being able to receive data from every connected sensor. For

instance, if four sensors are used to create a 360º fire alarm, all four sensors must have different addresses.

The concept is based on rewriting the address of the sensor, replacing it with a new one. The code shown in Listing I.6 is based on the one found in [57], and it changes the factory address to any given address, which in this case is 0x50. However, the NodeMCU development board isn't capable of running the aforementioned code, since an error related to the "i2cmaster.h" library occurs. This has to be with the architecture of the board it self, which is different from the Arduino boards. Since the code to change the address only needs to be ran once, an Arduino Uno was used to do so. In the Figure 4.5 is the scheme to change the address.

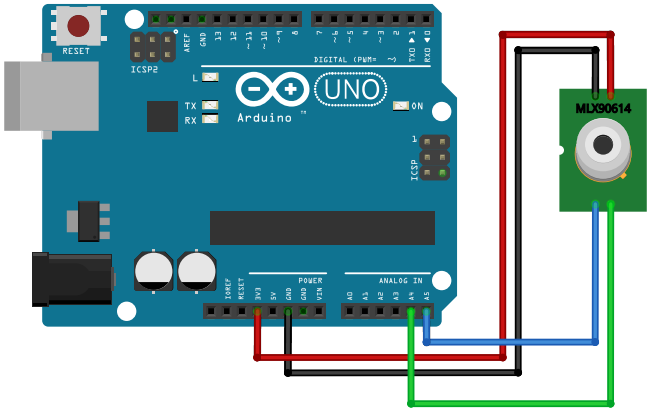


Figure 4.5: Diagram of the circuit used to change the address of a MLX90614 sensor.

4.4 GPS Data Acquisition

To acquire GPS coordinates, a GNSS receiver was necessary in the module. The component used is a module using the uBlox NEO-7M chipset and with the respective UART architecture, which means this is a "plug-and-play" type of component. As mentioned before, and unlike the IR thermal sensor, this GPS receiver communicates with the development board through UART. In the Figure 4.6 is the architecture with the GPS receiver and all the previously used components. As shown in the figure, the connections represented in orange and yellow are the UART connections.

To run the GPS receiver within the whole module, firstly, it was tested isolated. There is a library available for the Arduino IDE called "TinyGPS++.h", created to be compatible with any UART GPS modules. This library is a simplified version of a previous library called "TinyGPS.h" and comes with some example codes attached. To start exploring the GPS receiver, the "TinyGPS++.h" library was installed and an example code was tested, which is present in the Listing I.7. As shown in the code, the Baud rate of the GPS receiver needs to be specified, which is fixed at 9600 from the factory for the used module. The communication pins of the development board, Rx and Tx, also need to be specified. In the case of the NodeMCU ESP-12e, Rx is in the GPIO pin 13 and Tx in the GPIO pin 12.

Also, a constant for the GPS receiver needs to be specified. Lastly, the serial connection needs to be specified, using a constant with the Rx and Tx pins as parameters. All this initializations are shown in the Listing 4.5.

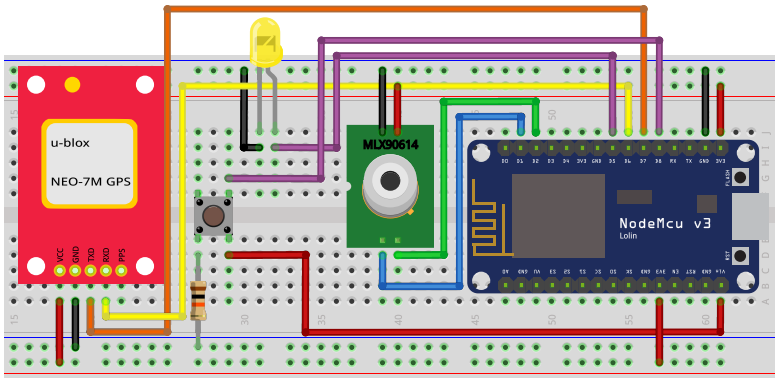


Figure 4.6: Diagram of the circuit for collecting thermal data and GPS coordinates.

Listing 4.5: GPS constant definitions.

```
4  static const int RXPin = 13, TXPin = 12;
5  static const uint32_t GPSBaud = 9600;
6  TinyGPSPlus gps;
7  SoftwareSerial ss(RXPin, TXPin);
```

In the setup function, the serial communication is initialized in an identical way as the serial monitor, by running the "begin()" method and using the Baud rate as an input parameter.

In the loop function, another function called "displayInfo()" is called. This function is self-explanatory, since all it does is displaying information about the GPS receiver.

Inside the aforementioned functioned, the information displayed is the location, the date and the time. The data received about the location are the latitude and longitude coordinates, displayed in decimal degrees. The example code prints to the serial monitor both latitude and longitude, respectively, only when the informations received is valid, as shown in the Listing 4.6. The coordinates are valid when the receiver is able to look at, at least, 3 satellites. This means that the GPS coordinates are only sent to the MCU when the receiver is locked-on. The receiver is locked-on when it is finally able to locate itself in the planet Earth, which means it's able to see enough satellites.

A similar process works for both date and time, where it's only displayed the information when such information is valid. The process to print the date to the serial monitor is quite straight forward, as shown in the Listing 4.7, since the functions called have unambiguous names.

Listing 4.6: GPS coordinates.

```
37  if (gps.location.isValid())
38  {
39     Serial.print(gps.location.lat(), 6);
40     Serial.print(F(","));
41     Serial.print(gps.location.lng(), 6);
42  }
```

Listing 4.7: GPS acquired date.

```
49  if (gps.date.isValid())
50    {
51      Serial.print(gps.date.month());
52      Serial.print(F("/"));
53      Serial.print(gps.date.day());
54      Serial.print(F("/"));
55      Serial.print(gps.date.year());
56    }
```

Same goes for time, although some tricks need to be made. As shown in the Listing 4.8, it needs to be checked if the values of the hours, minutes, seconds and centiseconds are bellow 10. If so, a 0 is added to the left of the number. An example is as follows: if the current time is 08:06:03, and the aforementioned condition isn't applied, the displayed time will be 8:6:3.

Listing 4.8: GPS acquired time.

```
63  if (gps.time.isValid())
64  {
65    if (gps.time.hour() < 10) Serial.print(F("0"));
66    Serial.print(gps.time.hour());
67    Serial.print(F(":"));
68    if (gps.time.minute() < 10) Serial.print(F("0"));
69    Serial.print(gps.time.minute());
70    Serial.print(F(":"));
71    if (gps.time.second() < 10) Serial.print(F("0"));
72    Serial.print(gps.time.second());
73    Serial.print(F("."));
74    if (gps.time.centisecond() < 10) Serial.print(F("0"));
75    Serial.print(gps.time.centisecond());
76  }
```

After displaying the GPS information, a function called "smartDelay()", found in [58], ensures that the receiver keeps being fed with information from the satellites, making the communication smoother and more constant.

To test the GNSS receiver and the IR thermal sensor together, a code was created, based on the Listings I.7 and I.3. The result is shown in the Listing I.8. The concept is

very similar to both previously mentioned codes, but, in this case, the data is directly sent to the serial monitor, instead of saving it using SPIFFS. The program starts by initializing all variables and components. The next step is to wait for the GNSS receiver to find and lock enough satellites. Then, the program waits for the user to give permission to collect data, which is performed through a push button. When the permission is given, the module starts collecting 200 lines of data values, which corresponds to a period of about 1 minute. This period depends on how constant the receiver is when finding satellites and receiving their data, as well as the speed of the IR thermal sensor when collecting data.

In the Figure 4.7 is shown the path walked during and experiment to collect data, while in the Figure 4.8 is represented a graph that displays object temperatures along the time of the day. The path was created using the website in [59]. It is possible to see that in the Figure 4.8 there are multiple points for a single value in the X axis. This is due to the GNSS receiver not being able to collect the centiseconds from the library function. The reason for such thing to happen is due to the frequency of data acquisition of the GNSS receiver, which is 1 Hz.

Figure 4.7: Image of the trajectory acquired using a tracking frequency of 1 Hz.

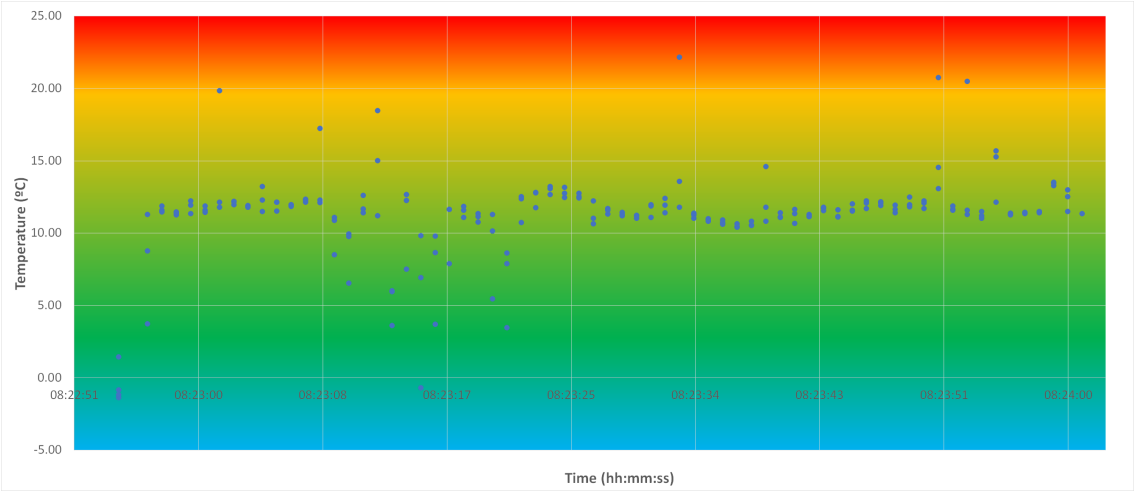


Figure 4.8: Graph of temperatures through time using a tracking frequency of 1 Hz.

To overcome this problem, the GNSS receiver was connected to a software called "u-center", developed by u-blox. This software allows the user to extensively customize the characteristics of the GNSS receiver. Following the manual in [60], the modifications applied were:

- **Baud rate:** From 9600 to 115200;
- **Measurement period:** From 1000 ms (1 Hz) to 100 ms (10Hz).

Those changes allowed the GNSS receiver to collect the centiseconds of the time sent from the satellite, since the period of signal acquirement is smaller than 1 second.

Since the working period of the module is going to be more than just a couple of minutes and the measurements are going to be made while riding a vehicle, a long range test using a car was made. This test had a duration of around 40 minutes, covering around 40 km of distance and using a period between readings of 150 ms. This test was good to verify the consistency of the data received along the time, as well as its frequency and accuracy, therefore, the circuit was downgraded to much simpler version, including only the development board and the GNSS receiver. The code used for testing is displayed in the Listing I.9, and is based on the Listing I.7.

After running the test, the log file from the Serial port was obtain, and it contained information about the GPS coordinates, the date and the time, as shown in the Figure 4.9.

gps_log_longtrip.txt

```
38.789026,-9.342990; 6/22/2019; 08:01:31.20
38.789021,-9.342987; 6/22/2019; 08:01:31.40
38.789018,-9.342985; 6/22/2019; 08:01:31.50
38.789013,-9.342982; 6/22/2019; 08:01:31.70
38.789007,-9.342979; 6/22/2019; 08:01:31.90
38.789004,-9.342977; 6/22/2019; 08:01:32.00
38.788999,-9.342974; 6/22/2019; 08:01:32.20
38.788997,-9.342972; 6/22/2019; 08:01:32.30
38.788992,-9.342967; 6/22/2019; 08:01:32.50
38.788989,-9.342965; 6/22/2019; 08:01:32.60
38.788983,-9.342959; 6/22/2019; 08:01:32.80
38.788979,-9.342956; 6/22/2019; 08:01:32.90
38.788972,-9.342950; 6/22/2019; 08:01:33.10
38.788969,-9.342947; 6/22/2019; 08:01:33.20
38.788963,-9.342940; 6/22/2019; 08:01:33.40
38.788956,-9.342933; 6/22/2019; 08:01:33.60
38.788952,-9.342929; 6/22/2019; 08:01:33.70
38.788944,-9.342922; 6/22/2019; 08:01:33.90
38.788940,-9.342919; 6/22/2019; 08:01:34.00
38.788932,-9.342912; 6/22/2019; 08:01:34.20
38.788927,-9.342908; 6/22/2019; 08:01:34.30
38.788920,-9.342899; 6/22/2019; 08:01:34.50
38.788916,-9.342895; 6/22/2019; 08:01:34.60
38.788907,-9.342885; 6/22/2019; 08:01:34.80
38.788899,-9.342875; 6/22/2019; 08:01:35.00
38.788895,-9.342870; 6/22/2019; 08:01:35.10
38.788886,-9.342859; 6/22/2019; 08:01:35.30
38.788882,-9.342853; 6/22/2019; 08:01:35.40
38.788872,-9.342841; 6/22/2019; 08:01:35.60
38.788867,-9.342835; 6/22/2019; 08:01:35.70
38.788857,-9.342823; 6/22/2019; 08:01:35.90
38.788848,-9.342811; 6/22/2019; 08:01:36.10
38.788843,-9.342805; 6/22/2019; 08:01:36.20
38.788834,-9.342792; 6/22/2019; 08:01:36.40
38.788829,-9.342786; 6/22/2019; 08:01:36.50
38.788819,-9.342774; 6/22/2019; 08:01:36.70
38.788814,-9.342769; 6/22/2019; 08:01:36.80
38.788804,-9.342757; 6/22/2019; 08:01:37.00
38.788800,-9.342751; 6/22/2019; 08:01:37.10
```

Figure 4.9: Portion of the log file of the GNSS receiver long trip test.

Then, the data from the file above was put into an Excel sheet, and turned into three different columns, each one with the three different information available. To analyse the tracking and the positions obtain, the website found in [59] was used, as before. The resulting image is seen in the Figure 4.10.

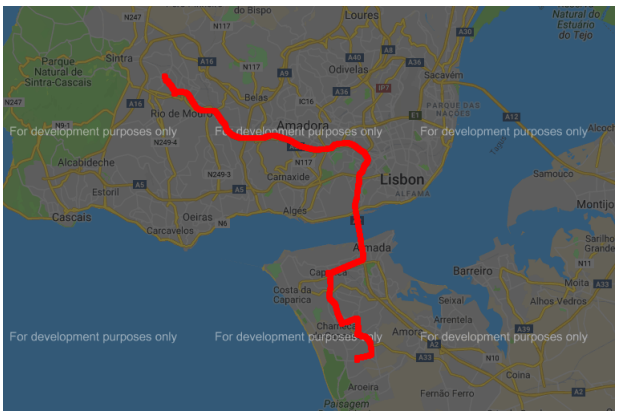


Figure 4.10: Resulting track using the obtained GPS coordinates.

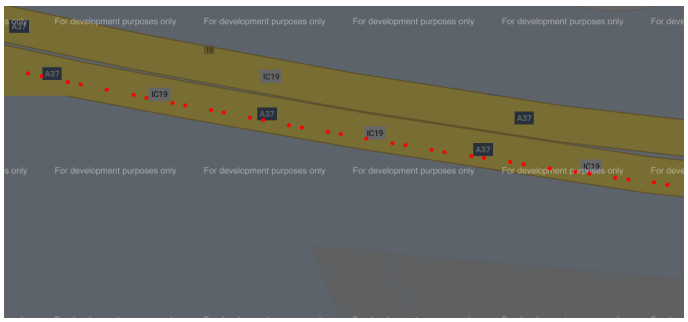
One thing that was verified with this experiment was the influence of high vegetation and building density in data acquisition.

In the Figure 4.11a is a sample of data obtained at a speed of approximately 90 km/h, in a highway, with clear view of the sky. It's possible to see that, despite the space between readings being greater (around 5 meters between consequent readings), the accuracy is much greater. This is due to the GNSS signal not suffering many deflections in buildings and vegetation.

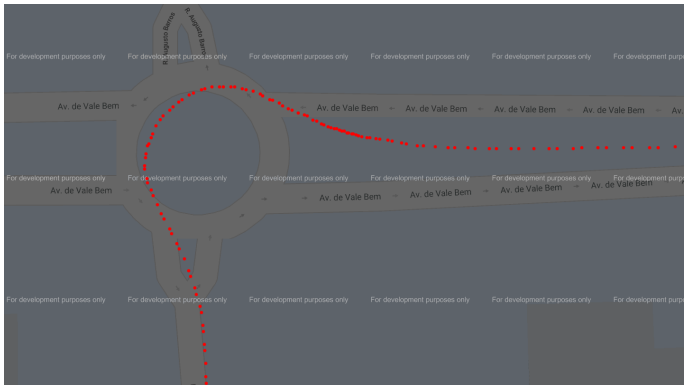
On the other hand, in the Figure 4.11b is a sample of readings obtained in a suburb area, with high vegetation and building density. While the speed in these regions is slower (between 40 and 60 km/h), which makes the distance between consequent readings shorter, the accuracy is lower. Despite the decrease in accuracy, the margin of error is around 2 - 3 meters.

One problem that was found after the previous experiment is that the GNSS module isn't capable of keeping the changes made to its baud rate or reading frequency. This is due to the module having a small capacitor working as a battery after a power cycle is done to the module. Taking this problem into account and the fact that both the alarm and the scanner won't require readings speeds with a faster period than 1 second, the following decisions were made:

- Keep the GNSS baud rate and reading frequency in the factory values (9600 and 1 Hz, respectively);
- The period between updates of both the Alarm Module and Thermal Mapping Module must be higher than 1 second;
- In the case of the Thermal Mapping Module, the period must be as close to 1 second as possible and the speed of the vehicle as to be adapted to avoid unmeasured areas. The speed must be low enough to create a bit of overlap between consequent readings and high enough to cover as much area as possible in the minimum time possible.



(a)



(b)

Figure 4.11: Comparison between data obtained in a open space area (a) and an area with many buildings and vegetation (b).

4.5 O_2 Sensor

To protect the fire fighters inside the vehicle, it's necessary to check if the oxygen levels in the air inside the vehicle are not dangerously low. To do so, an O_2 sensor is used.

The chosen one is the ME2-O2 from Seeed. This is a sensor which connects to the development board via analog input and is ready to program using the Arduino IDE, as the base code used to obtain the percentage of O_2 in the air is found in the library in [61]. In this library comes a C++ file "Oxygen.cpp" and its header file "Oxygen.h", shown in the Listings I.10 and I.11 respectively, and this pair of files composes the function to obtain the concentration levels of oxygen.

The code above is small and easy to understand, so it was quickly adapted to a code for testing the concentration of oxygen inside a container. The circuit was simply composed by the O_2 sensor, one IR single-pixel thermal sensor and one ESP-12e development board, as the Figure 4.12 suggests.

The test itself consisted of a small bottle of water, working as a container for a small candle. The bottom of the bottle was cut open, to try and cover the candle close to the table. The cap of the bottle was cut with the diameter of the O_2 sensor, in order to avoid air entering or exiting the container, as shown in the Figure 4.13. The Figure 4.14 shows how the container was made.

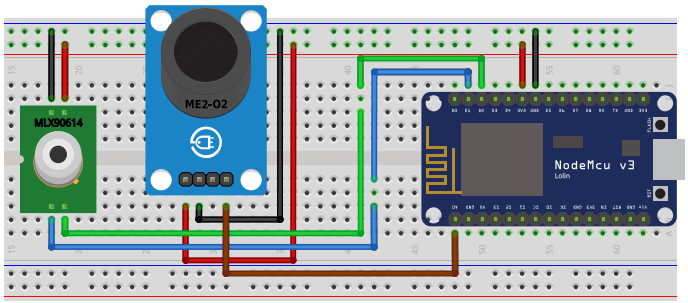


Figure 4.12: Scheme for testing the Oxygen sensor.

(a) (b)

Figure 4.13: Cut of the bottle cap (a) and sensor placed in cap, creating isolation (b).

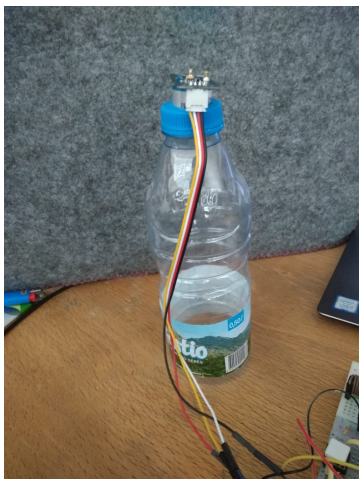


Figure 4.14: Full model of the container for testing O_2 concentration.

To make the test, a candle is lit and covered by the container. The IR thermal sensor will measure the temperature of the container. The expected result is that the oxygen concentration levels will drop when the candle is covered by the container, and the temperature of the container, which increases in the beginning, will start to slowly decrease once the candle goes out. The Figure 4.15 shows how the test occurred, with the bottle covering the candle, with the oxygen sensor on the top pointing inside the bottle and the IR thermal sensor pointing to the bottle to measure the temperature. The code used is shown in the Listing I.12.

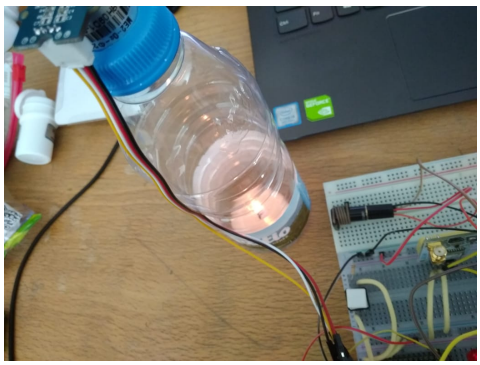


Figure 4.15: Test of the O_2 sensor using a candle inside a container.

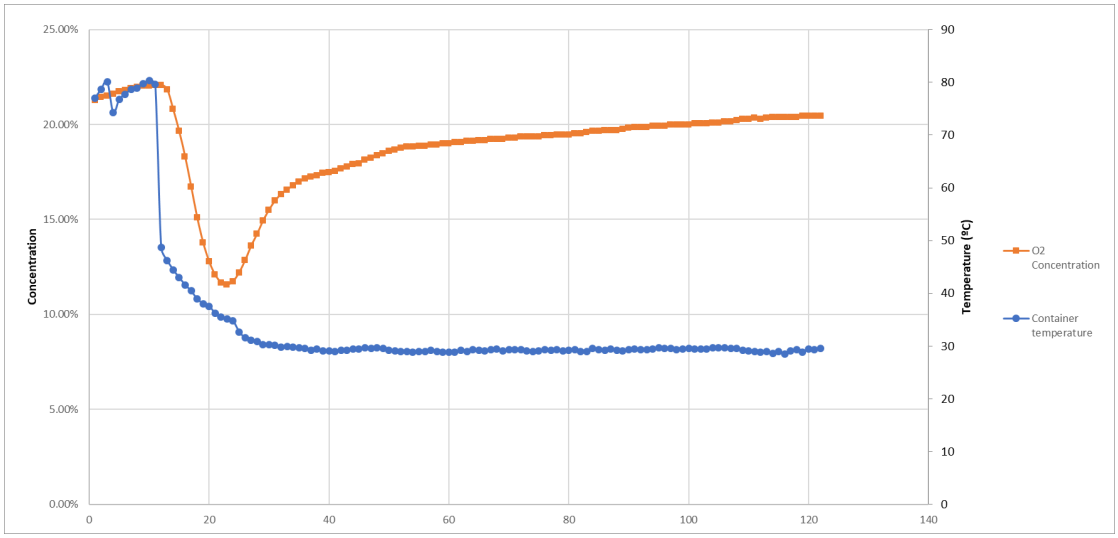
Running the test, the graphs obtained are in the Figure 4.16. In the first figure, is possible to see a big drop in the container temperature. That is because the IR thermal sensor was pointing directly to the candle and it was then covered by the container. So, the read temperature is no longer related to the candle but to the container, which justifies the drop in temperature. At the same time that this drop in temperature occurs, a drop in O_2 concentration also occurs, caused by the container creating an environment similar to vacuum. The candle starts to run out of oxygen until it goes off. The temperature then stabilizes after the candle is put out, as the comparison in the Figure 4.16b suggests. However, in the Figure 4.16a, we see that the oxygen concentration slowly goes up to normal levels. This is due to the container not being well isolated from the outside, therefore occurring small transferences of air between the inside and the outside of the container.

The lowest value of oxygen concentration obtained was 11.56%, which means that it corresponds to the level when the candle was put out. However, safe levels of oxygen concentration in air are higher than that value. As mentioned in [62], atmospheres with a O_2 concentration lower than 19% start to be detrimental to people's health. The Table 4.1 shows some of the effects of low oxygen concentration levels to health.

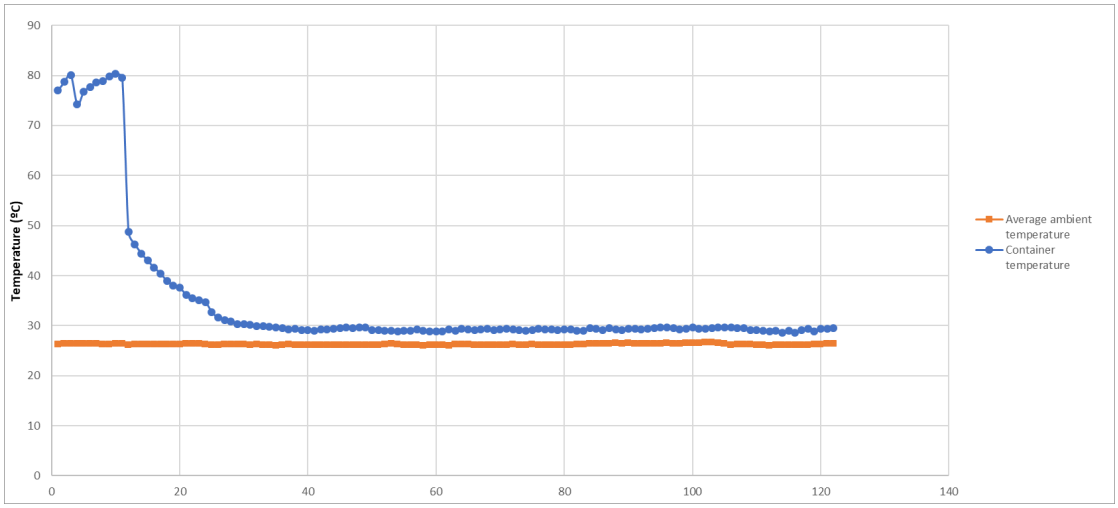
Table 4.1: Effects of oxygen-deficient exposure.

Source: [62]

Oxygen Concentration (% vol)	**Health effects of people at rest**
19	Some adverse physiological effects occur, but they may not be noticeable.
15 - 19	Impaired thinking and attention. Increased pulse and breathing rate. Reduced coordination. Decreased ability to work strenuously. Reduced physical and intellectual performance without awareness.
12 - 15	Poor judgement. Faulty coordination. Abnormal fatigue upon exertion. Emotional upset.
10 - 12	Very poor judgement and coordination. Impaired respiration that may cause permanent heart damage. Possibility of fainting within a few minutes without warning. Nausea and vomiting.
<10	Inability to move. Fainting almost immediate. Loss of consciousness. Convulsions. Death.



(a)



(b)

Figure 4.16: Graphs of candle experiment comparing container temperature with oxygen concentration (a) and comparing container temperature with ambient temperature (b).

4.6 IR Multi-pixel Thermal Sensor

In order to obtain a more accurate measurement of the temperatures, a sensor with more pixels was tested. In this case, the IR thermal sensor used was the Melexis MLX90640, which has a 32x24 pixel array.

This sensor has an aperture of 55ºx35º. To determine the width and height range of the sensor, some calculation was necessary. As shown in the Figure 4.17, it was considered that the sensor was one meter away from the object, in order to make calculations easier. To apply the trigonometric equations, the two apertures were split into right triangles. Taking this into account, the tangent of half the aperture results in half of the range at a distance of one meter.

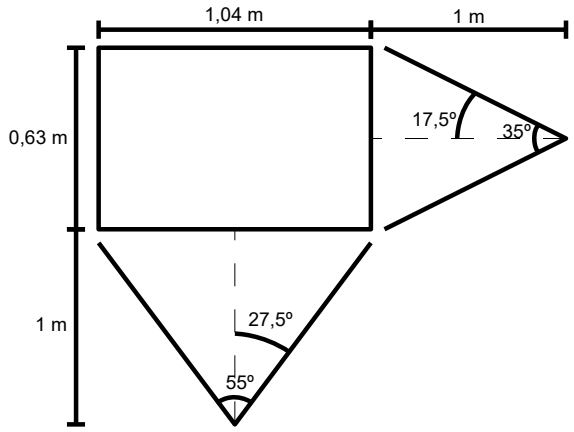


Figure 4.17: Visual representation of the aperture, width and height of the sensor at a certain distance.

So, as follows, the half width (hw) and half height (hh), respectively, are:

$$\begin{aligned} \tan(27.5^\circ) = hw/1 \Leftrightarrow hw = \tan(27.5^\circ) \Leftrightarrow hw \approx 0.521 \\ \tan(17.5^\circ) = hh/1 \Leftrightarrow hh = \tan(17.5^\circ) \Leftrightarrow hh \approx 0.315 \end{aligned} \tag{4.1}$$

By multiplying the results of the equation (4.1) by the factor of 2, the width and height at a distance of 1 meter, respectively, are:

$$\begin{aligned} width \approx 1.04\ m \\ height \approx 0.63\ m \end{aligned} \tag{4.2}$$

With the results of the equation (4.2), it's possible to calculate the approximate resolution of each pixel of this sensor.

$$\begin{aligned} pixel\ width &= \frac{104\ cm}{32} = 3.25\ cm \\ pixel\ height &= \frac{63\ cm}{24} = 2.625\ cm \\ resolution &= \frac{pixel\ width}{pixel\ height} = \frac{26}{21} \end{aligned} \tag{4.3}$$

After exploring the more technical part of the sensor, some tests were made with the sensor, connected to the ESP32 DevKitC board. Sparkfun provides a library with examples ready to use with the MLX90640 sensor, and that was the base for this tests. The aforementioned library is found in [63] and comes with 3 different examples. The example that was mainly used was the example 2, found in Listing I.13. Each example comes with two ".h" and two ".cpp" files, corresponding one header file to one ".cpp" file. One pair of files has functions related to the I2C communication between the sensor

and the MCU. The other pair has functions related to the extraction of parameters of the sensor.

When the code was run for the first time, the 768 values came all in the same line, as the code in the Listing 4.9 shows. So, the first modification was to print the values in the serial port in the form of a matrix. The Listing 4.10 shows the solution proposed to substitute the previous coding lines, which is printing a breakline after 32 values in a row. However, the value "x", that represents the position of the array of temperatures needs to point to the correct position. To do so, the calculations in the equation 4.4 were made, being c the column number, between 0 and 31, and l the line number, between 0 and 23. That means that, if you want to know the temperature in the column 3 of the line 10, you have to extract the value of the position 303 of the array.

$$Array\ Position = c + l \times 32 \tag{4.4}$$

Listing 4.9: Display temperatures in single array.

```
93   for (int x = 0 ; x < 768 ; x++)
94   {
95     Serial.print(mlx90640To[x], 2);
96     Serial.print(",");
97   }
98   Serial.println("");
```

Listing 4.10: Display temperatures in matrix.

```
93  int breakline = 0;
94  String s = "";
95  for (y = 0; y < 24; y++){
96    k = y*32;
97    for (int x = 0+k ; x < 32+k ; x++){
98      float valor = mlx90640To[x];
99      breakline++;
100     if(breakline == 32){
101       breakline = 0;
102       s=s+String(valor,2)+"\n";
103     }
104     else{
105       s=s+String(valor,2)+",";
106     }
107   }
108 }
```

With these changes applied, the values came in the form of a matrix, which makes it easier to visualize in the form of an image. By picking one frame as an example and

copying it to Excel, with the help of conditional formatting, it was discovered that the values were being printed in a mirrored way. This is due to the sensor reading from the top right corner, line by line, down to the bottom left corner. To correctly display the image, part of the code was substituted to the one represented in the Listing 4.11. The only modification was in the way the line is read, so instead of reading from the column 0 towards column 31, it now reads the other way around.

Listing 4.11: Display temperatures in matrix without mirror.

```
95  for (y = 0; y < 24; y++){
96      k = y*32;
97      for (int x = 31+k ; x >= 0+k ; x--){
98          float valor = mlx90640To[x];
99          if(breakline == 32){
100           breakline = 0;
101           s=s+String(valor,2)+"\n";
102         }
103         else{
104           s=s+String(valor,2)+",";
105         }
106     }
107 }
```

After applying the aforementioned change, it was noticed that the image was quite grainy. To make a smoother image, a non-uniform mean filter was applied to the pixels, to give the aspect of a smoother image.

The way this was made is shown in the image 4.18. The central pixel, marked in red, has the greater weight when calculating its new mean value, while directly adjacent pixels have less weight and diagonal pixels have even less weight.

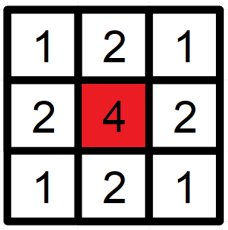


Figure 4.18: Non-uniform mean filter using 8 adjacent pixels.

For the example above, and assuming each pixel in the form of *pixel(line, column)*, the value of the pixel (2,2) is:

$$pixel(2,2) = \frac{pixel(2,2)\times 4+(pixel(1,2)+pixel(2,1)+pixel(2,3)+pixel(3,2))\times 2+pixel(1,1)+pixel(1,3)+pixel(3,1)+pixel(3,3)}{16} \quad (4.5)$$

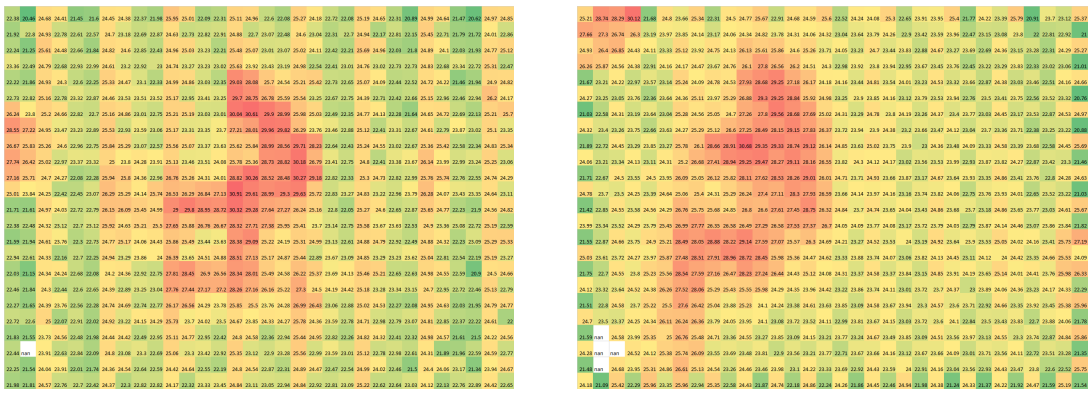
Applying the equation (4.5) to the code, the previous version of the printed matrix is substituted by the code in the Listing 4.12. It's important to underline that this change

was not applied to the margins, since the code would become too complex and slow. This is done in the lines 98, 99 and 101, by verifying if the current pixel isn't in either the first line, the last line, the first column or last column.

Listing 4.12: Display temperatures in matrix with non-uniform mean filter.

```
95  for (y = 0; y < 24; y++){
96    k = y*32;
97    for (int x = 31+k ; x >= 0+k ; x--){
98      int div1 = (x+1)%32;
99      int div2= x%32;
100     float valor;
101     if(x < 734 && x > 32 && div1!=0 && div2 !=0){
102       valor = ((mlx90640To[x]*4) + (mlx90640To[x+1]*2) + (mlx90640To[x-1]*2) +
              ↪ (mlx90640To[x+32]*2) + (mlx90640To[x-32]*2) + mlx90640To[x-31] +
              ↪ mlx90640To[x-33] + mlx90640To[x+31] + mlx90640To[x+33]) / 16;
103     }else{
104       valor = mlx90640To[x];
105     }
106     breakline++;
107     if(breakline == 32){
108       breakline = 0;
109       s=s+String(valor,2)+"\n";
110     }else{
111       s=s+String(valor,2)+",";
112     }
113   }
114 }
```

This change turns out to be great for visualization, as the comparison on the Figure 4.19 shows, but its importance to fire detection is still dubious.



(a) (b)

Figure 4.19: Comparison between the original thermal image (a) and a similar thermal image with a non-uniform mean filter (b) of a sitting person.

On the other hand, this change allowed to discover a problem, related with possible failures of a pixel in the sensor. In the Figure 4.19a is possible to see the original thermal

image, created with conditional formatting in Excel. On the bottom left corner of the image, a pixel is broken and its value is considered Not a Number (NaN). In the Figure 4.19b, after running the non-uniform mean filter, it's possible to see that the broken pixel propagates the NaN value to other pixels. Therefore, a function to correct those values was created and inserted in the code, shown in the Listing 4.13 in the lines 101 to 103. That portion of code just looks to the values and, if it's a NaN, the value of the current pixel will become the mean value between the previous and the next value of the array.

Listing 4.13: Correction of NaN values in pixels.

```
98  int div1 = (x+1)%32;
99  int div2= x%32;
100 float valor;
101 if (isnan(mlx90640To[x])) {
102   mlx90640To[x] = (mlx90640To[x+1] + mlx90640To[x-1])/2;
103 }
```

After tweaking the original code towards a more capable and compact code for the functionality needed, some tests regarding its range were made. A simple test, using an halogen heater, was run to determine the longest distance to detect a potential reignition focal point, using the code in the Listing I.14. This code was run with the ESP32 DevKitC development board, because this sensor is meant to be connected to this board. The scheme for the circuit is shown in the Figure 4.20.

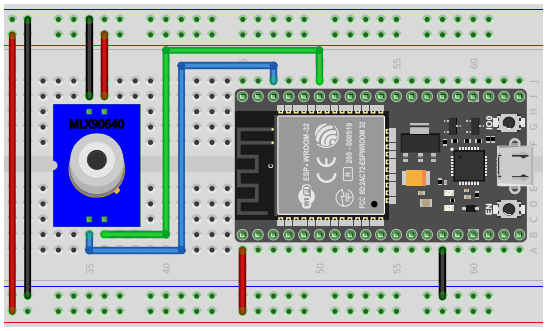


Figure 4.20: Circuit for testing the MLX90640 IR thermal sensor.

Many frames from different distances to the heater were taken, in order to better detect the maximum distance of detection where the heat is still well perceived. In the end, a graph relating the area with the heater's average temperature and the distance to the heater was created, comparing the obtained values with the average ambient temperature.

As seen in the Figure 4.21, the heater is easily detected at distances within 2 and 20 meters. However, at distances of 40 meters, is still possible to detect the heater, since the core has a temperature of around 25 ºC and the average ambient temperature is around 20 ºC. In the Figure 4.22 is possible to see the comparison of the results at distances of 4 meters and 40 meters. The aspect of the image is identical and the core of the heater is

still detectable. This concludes that, at a distance of 40 meters, is possible to detect an object with a core temperature of around 130 ºC and an area of 0.05 m².

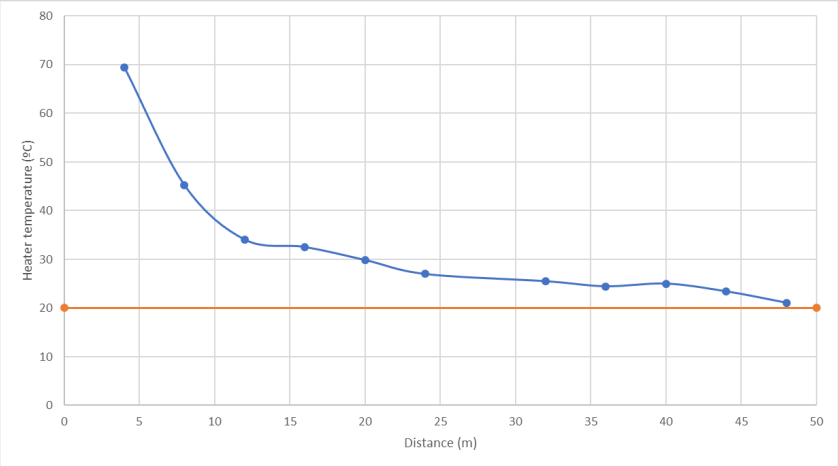
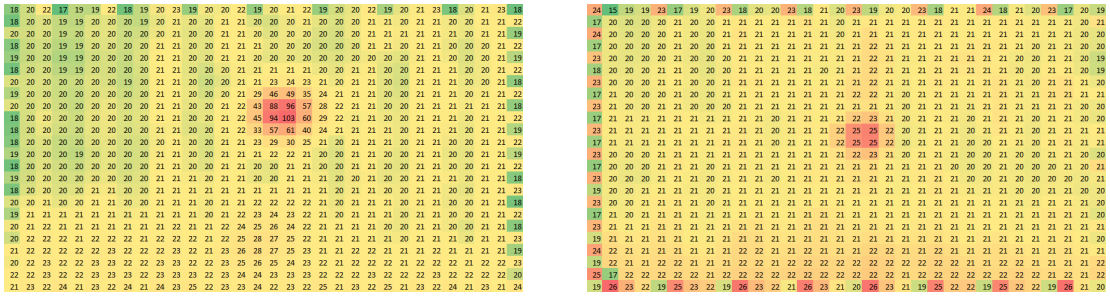


Figure 4.21: Relation between heater average temperature and distance to heater.



(a) (b)

Figure 4.22: Comparison between the thermal image of the heater at a distance of 4 meters (a) and at a distance of 40 meters (b).

4.7 OLED Display

After testing every necessary component, it was time to test the OLED display. Despite not being a crucial component, it's very useful to show data to the user of the Alarm and Thermal Mapping Modules. In this case, a small 0.96" SSD1306 monochrome OLED display will be able to show messages and useful information to the user.

One unexpected problem that emerged was that some libraries for the OLED display were not compatible with the libraries of other components using I2C, like the single-pixel and the multi-pixel IR thermal sensors. After searching and experimenting many different library and existing DIY projects, a solution was found by trying the project in [64]. This project uses two Adafruit libraries, the "Adafruit SSD1306" [65] and the "Adafruit GFX" [66], both dedicated for SSD1306 monochrome OLED displays and compatible with the libraries already in use for the project.

4.8 Building the Thermal Mapping Module

As mentioned in the Section 3.3.2, this module will be composed by an ESP32 DevKitC development board, one MLX90640 IR thermal sensor, one 0.96" OLED display and one u-blox NEO-7M UART GPS receiver.

Regarding the set up of the components, the refresh rate of the IR thermal sensor was set to 2 Hz since, after many tests, it was the one rate that gave out best and more stable results.

The baud rate of the development board didn't make much difference to the speed of processing, yet it was increased from 115200 to 921600 when the ESP-12e board was changed for the ESP32 DevKitC board, since it allows a faster refresh ratio of the COM port.

Relative to the GNSS receiver, and as mentioned at the end of Section 4.4, both baud rate and frequency of data acquisition will stay at default values, due to configuration losses after a power cycle. Therefore, the module won't update to new values in less than 1 second.

Removing unnecessary lines of "Serial.println()" and "Serial.print()" makes the code faster. Sending data to the OLED display also takes a considerable time, so more optimized ways to send messages to the display also make the code faster. The part that takes the most time is when acquiring the values of each pixel of the MLX90640 IR thermal sensor. The library used is close to optimized, so changing the function that collects the data from the sensor is out of question and would take too much time. However, it's possible to obtain all 768 temperature values and GPS coordinates within 1 and 1.5 seconds between readings.

The code used in this module is shown in the Listing I.15, resulting from a mix of the different codes tested for each of the components. The final scheme for the module is show in the Figure 4.23. This module will receive the GPS data, such as date, time and coordinates, and an array of 768 values from the multi-pixel IR thermal sensor. Based on this information, some messages are shown in the OLED display.

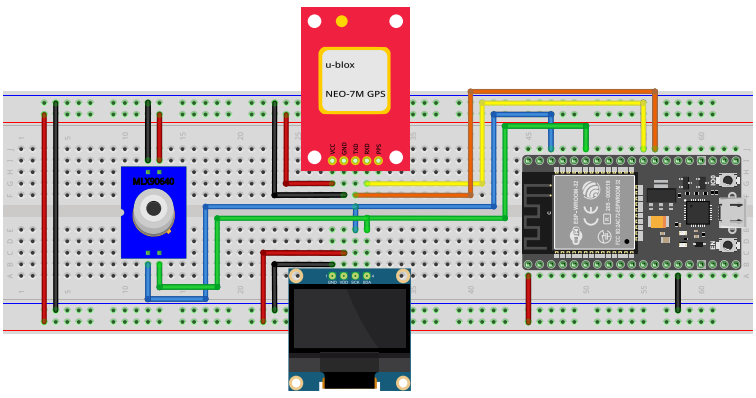


Figure 4.23: Final scheme for the Thermal Mapping Module.

To test the functionalities of the module, a small scale test was performed. This test consisted of creating a small fire in a barbecue grill, since it's the safest way to create something similar to a small fire. The second step consisted of driving alongside the placed fire, collecting all the necessary data (thermal and location data). The objective is to verify the capabilities of the module, test different vehicle speeds while scanning, scan at different distances and prove that is possible to have overlap between consequent readings.

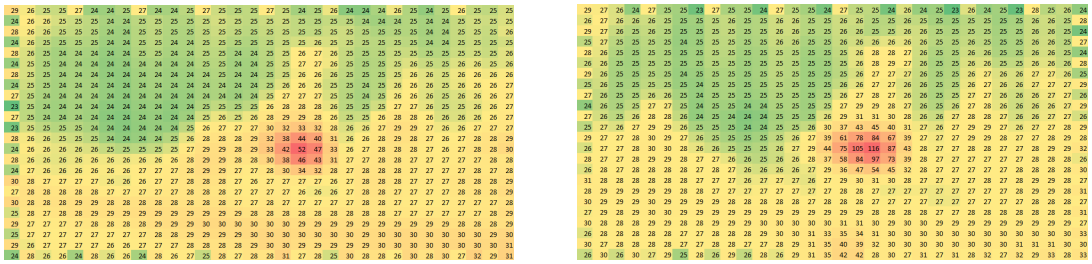
Before starting the test, some measurements were made at distances of 10 meters, 20 meters and 30 meters, in order to confirm that it was possible to measure the fire, since its size was somewhat small. In the Figure 4.24 are compared the 3 different distances where the fire was measured, giving more or less a preview of how distant the module would be from the barbecue fire.

(a) (b) (c)

Figure 4.24: Comparison between the measuring distances at 10 meters (a), 20 meters (b) and 30 meters (c).

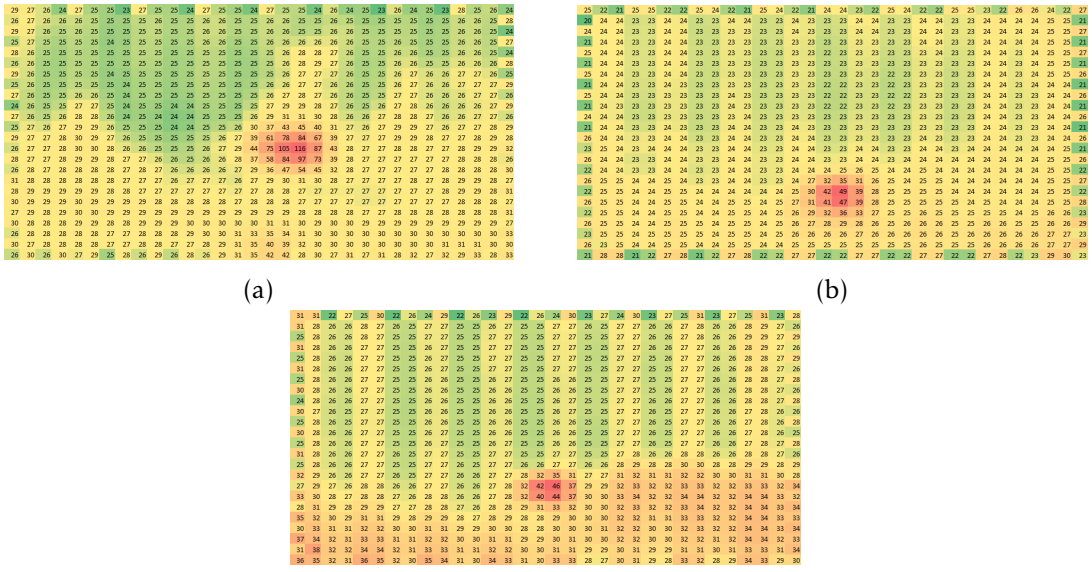
By comparing the fire from the Figure 4.24a with the ones from Figures 4.24b and 4.24c, it's possible to see that the flame in the latter figures is much stronger than in the first one. To verify the impact of the state of the barbecue fire, two different measurements were made at the same distance of 10 meters. In the Figure 4.25 is possible to see that, with the stronger flame, the sensor perceives almost double the core temperature of the weaker flame. This shows that the intensity of the fire might influence some results of the test when the distance to the barbecue grill is increased.

In the Figure 4.26 is shown a comparison between measurements at 3 different distances, with the flame maintaining more or less the same intensity. The variation in the core temperature of the fire is expected, since this question was approached in the Section 4.6.



(a) (b)

Figure 4.25: Comparison between a weak flame (a) and a strong flame (b) measured at 10 meters of distance.



(a) (b)

(c)

Figure 4.26: Comparison between measurements values at a distance of 10 meters (a), 20 meters (b) and 30 meters (c).

To run the test itself, the module would be inside of a vehicle, collecting data. The vehicle would pass at constant speed in a straight line, next to the barbecue fire. This will allow to verify any discrepancies with the GPS data and the overlap between thermal images.

Firstly, were made 2 passes in a road located at 20 meters of distance from the barbecue fire in its perpendicular and closest point. The first pass would be at 20 km/h and the second at 30 km/h. There was an attempt to run a third pass at 40 km/h, but the specific road used was too bumpy to collect data to the computer and would also compromise the safety of the vehicle, since some damage could be caused to it.

In the second step, two other passes would be made in a road at a distance of 30 meters to the barbecue fire in its closest point. The first pass would be run at a constant speed of 30 km/h and the second at 40 km/h.

The first pass in the first road resulted in collecting 8 values across 34 meters of distance, which translates in about 4.86 meters between consequent readings. The measurement positions in the map are displayed in the Figure 4.27, where the red star represents the first position and the red cross represents the last position. In this pass, the GPS data was very accurate, even given that the surrounding area had very high density of trees, which could affect the precision of the coordinates.

Figure 4.27: Mapping of the first pass, at a distance of 20 meters and speed of 20 km/h.

In the Figure 4.28 is the sequence of thermal images obtained in this first pass. The most positive aspects are that the flame was well detected and there was some overlap between different positions, which was expected. On the other hand, is possible to see that, in the Figures 4.28c to 4.28f the area of heat varies, which is due to both the flame being in a diagonal relative to the sensor and the heat being reflected in the walls of the barbecue grill. In the Figures 4.28a, 4.28g and 4.28h is notable a red blur in the bottom part of the image, which is the interior of the vehicle, therefore, those values are not going to be taken into account as a heat source. Overall, the results of this pass are as expected and very positive, proving that, at a speed of 20 km/h and a distance of 20 meters to the possible fire location, is possible to make a very safe and precise aftermath analysis to discover possible reignition spots.

After the first pass, a second pass was made, at a constant speed of 30 km/h. The path made was similar to the previous one, as shown in the Figure 4.29. However, a small delay happened between the second and the third positions, which increased the total distance to around 47 meters for 6 acquired positions. If the aforementioned distance is discarded, the average distance between two consecutive positions is around 7.5 meters, which is a good spacing between measurements. It is also possible to see in the figure that there is a small deviation between the road and the obtained coordinates. This measuring error might be caused by two factors. The first one is that the experiment was made in an area with high density of vegetation. The second cause is that the GNSS receiver might not had many satellites in view, in order to give more accurate positioning. If the receiver only has 4 satellites in view, its positioning will be more prone to errors than if the receiver has 8 satellites in view, for example.

Regarding the thermal values obtained, the second pass also gave great results, as was

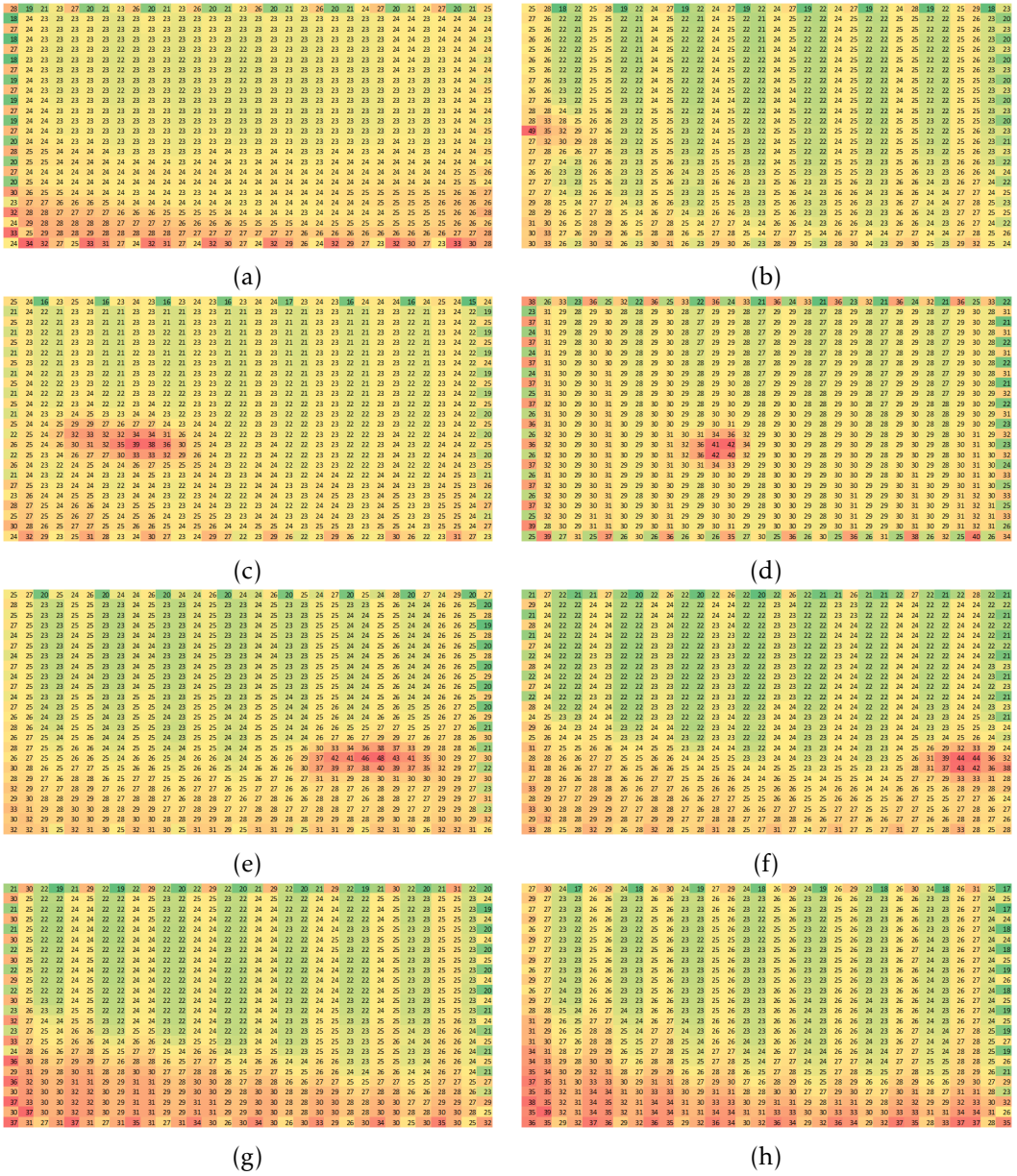


Figure 4.28: Sequence of thermal images obtained at a distance of 20 meters to the barbecue fire and at a constant speed of 20 km/h.

Figure 4.29: Mapping of the second pass, at a distance of 20 meters and speed of 30 km/h.

expected. The Figure 4.30 shows the sequence of thermal images obtained during this pass. Once again, the results obtained were as expected, with the barbecue fire appearing in 3 different consequent moments, shown in the Figures 4.30d, 4.30e and 4.30f, proving that some overlap between consequent readings was created. This concludes that, for a short distance of 20 meters to a small spot of fire, is possible to detected it with precision at a speed of 30 km/h while overlapping consequent readings.

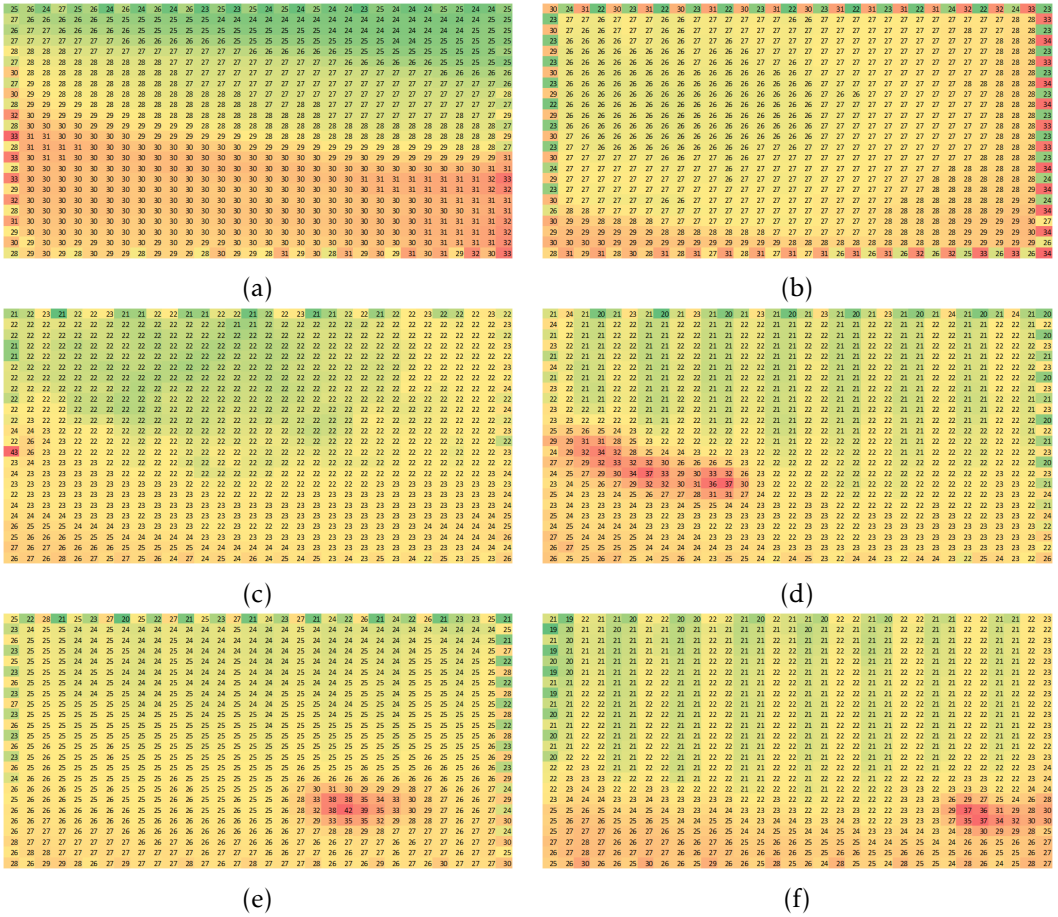


Figure 4.30: Sequence of thermal images obtained at a distance of 20 meters to the barbecue fire and at a constant speed of 30 km/h.

The third pass of the experiment was made in a parallel road to the one of the first and second pass, but this one at a distance of 30 meters to the barbecue fire. The speed was kept at 30 km/h and the test was run in a similar way. However, this test could also be affected by a row of bushes between the road and the area where the fire was located, shown in the Figure 4.31, since the sensor could sometimes not have a clear view of the flame. Another problem was some vehicles that were in the way between the sensor and the flame, which is also possible to see in the aforementioned figure.

Regarding the GPS coordinates received, this time they came with even more error than the previous two passes. As mentioned in the second pass, the two possible causes

Figure 4.31: Row of bushes between the road and the barbecue fire, which could represent a problem for obtaining thermal values.

to this deviation in the coordinates might be the few number of satellites connected to the GNSS receiver and the fact that the experiment was made in a region with high density of vegetation, meaning that, during the cold start, the GNSS receiver didn't have a perfectly clear view of the sky. On the other hand, regarding a test made in the Section 4.4, this greater deviation was also expected. The resulting values are shown in the Figure 4.32 and, despite the deviation, the distance between consequent values is good. The five values obtained were covered by a distance of 36.5 meters and the distance between consequent values was around 9 meters. The average spacing is greater in the third pass than in the second pass using the same constant speed, which is a strange aspect. However, this difference might be justified by poor reading of the speedometer of the vehicle.

Figure 4.32: Mapping of the third pass, at a distance of 30 meters and speed of 30 km/h.

As was expected, the thermal values were quite affected by the row of bushes alongside the road. It's possible to verify in the Figure 4.33 that the bushes block the view to the flame, since in the Figure 4.33d is possible to see clearly the flame but in the Figures 4.33c and 4.33e only a small and weak blur is visible. It's also worth noting that the red blurs on the bottom and left part of the images are part of the reflection of the vehicle, therefore

these values are not taken into account. If the impact of the bush barrier is discarded, the results are close to what was expected, with a small flame being detected clearly at least once, despite its overlapping not being as clear as pretended.

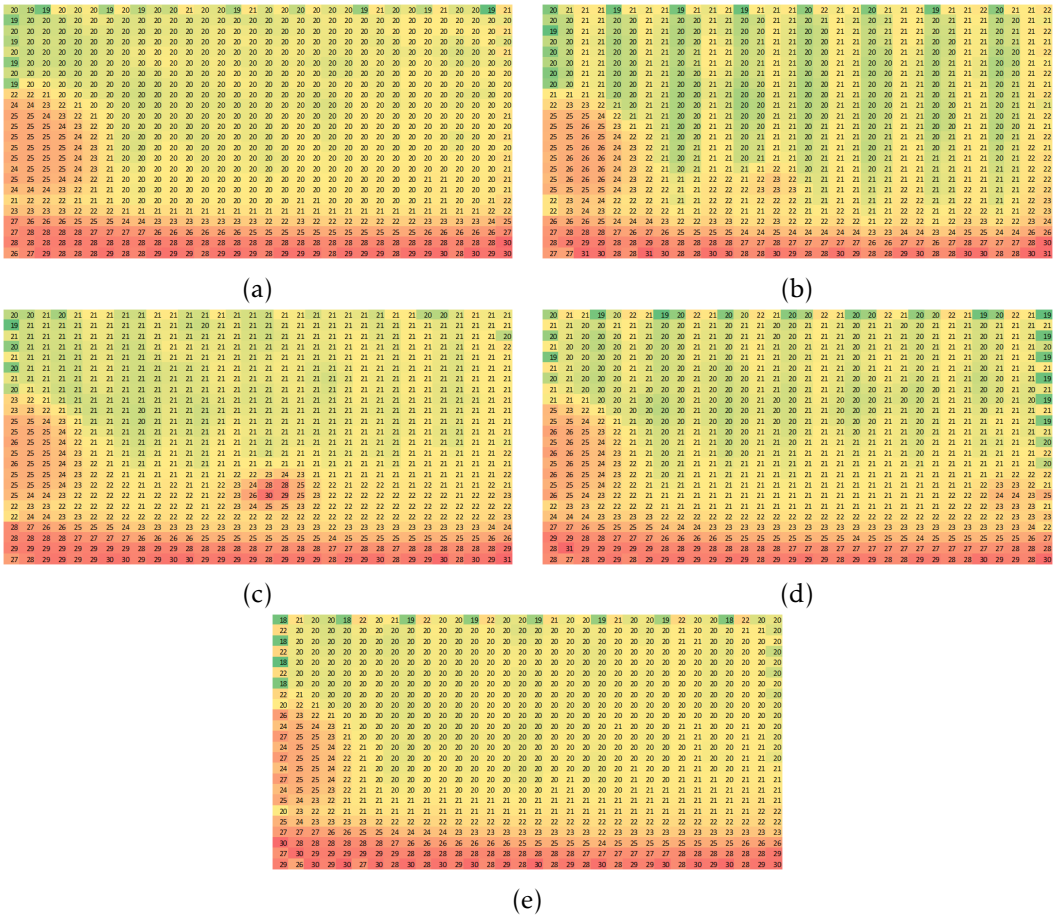


Figure 4.33: Sequence of thermal images obtained at a distance of 30 meters to the barbecue fire and at a constant speed of 30 km/h.

After the third pass, a final one was attempted at a speed of 40 km/h. Unfortunately, due to the density of the barrier of bushes and the flame starting to get weaker, it wasn't possible to get any relevant thermal data. Nevertheless, the experiment was a success, since it proved the capabilities of the Thermal Mapping Module.

4.9 Building the Alarm Module

As planned in the Section 3.2, the last prototype is the final module, composed of four MLX90614 single-pixel IR thermal sensors, one ESP-12e development board, one u-blox NEO-7M UART GPS receiver, one ME2-O2 oxygen sensor and one 0.96" OLED display.

In the scope of the configuration of the components, is very similar to the one used in the Thermal Mapping Module, but for different reasons. The baud rate and refresh frequency of the GNSS receiver stays at 9600 and 1 Hz, respectively, since the system

doesn't have to check the conditions inside and outside the vehicle in periods smaller than 1 second. As for the baud rate of the development board, it stays at 115200, since the ESP-12e isn't capable of greater baud rates.

The code used to run the Alarm Module is found in the Listing I.16, and is based in the various testing made to the components. The final scheme for the module is show in the Figure 4.34

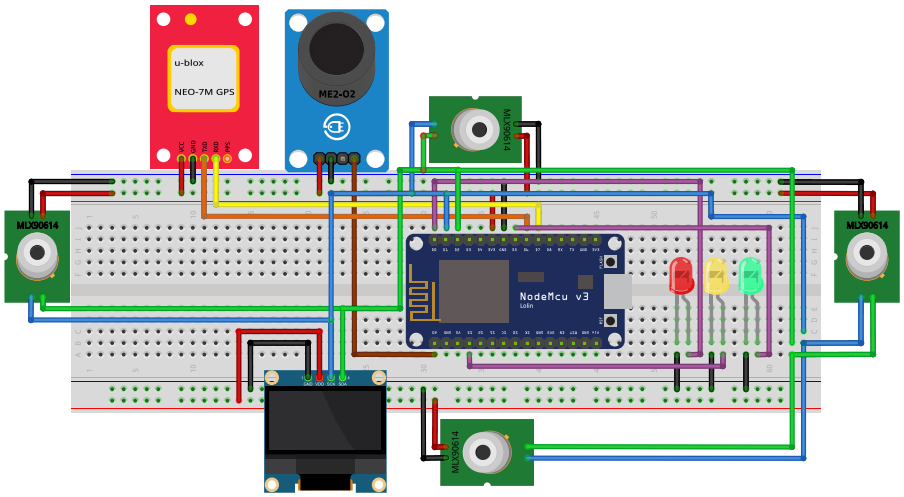


Figure 4.34: Final scheme for the Alarm Module.

The module running the aforementioned code is capable of showing in the OLED display the temperatures read from the four IR thermal sensors, the average ambient temperature, the GPS coordinates and the O_2 concentration. This module is also capable of using 3 different color LEDs to show different danger states. The green LED is active when there is no danger to the user of the module. The yellow LED is active when there is danger regarding the levels of O_2 in the air. The red LED is active when there is danger regarding nearby high temperatures surrounding the vehicle equipped with the module. Official testing is yet to be done to this module, however, it's fully functional and ready to collect data.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

In conclusion to this thesis, the achieved result was inside the expectancy from the beginning of the project. In general, the development was good, with the majority of the components being quite easy to work with, since all of them had some sort of example coding available in the internet. This compatibility of the components with the Arduino IDE also made it easier to connect them to the development board. Using the Arduino IDE also made it easier to understand the way each example code worked, allowing to merge different codes and customizing them in order to fill the need for the module in use.

However, some components, like the oxygen sensor and the multi-pixel IR thermal sensor were harder to implement in the respective modules, due to the small amount of coding available. The one that proved to be harder to implement was the MLX90640 multi-pixel IR thermal sensor, since this code came with 2 critical libraries. Those files came with many lines of coding and with few to none comments, which made harder to understand or optimizing the code to be more efficient.

Regarding the efficiency matter, another problem during the project was related to the configuration of the GNSS receiver, since this component is not able to keep the changes made to the configuration. This affected possible configuration changes to the baud rate and the frequency of signal acquisition, which could make the GNSS receiver much faster.

It's worth mentioning that some tests were delayed due to some material not being available at the time, which resulted in periods where the project was somewhat idled.

In general, and despite some of the drawbacks during the period of development, the project went as expected and the initial goals were met, resulting in an exciting and interesting work as well as a great learning curve.

5.2 Bill of Materials (BoM)

In order to define the costs of the project, the Bill of Materials (BoM) was divided into two tables, being one for the Thermal Mapping Module and the other for the Alarm Module. The total cost to build the Alarm Module is 240.36 €, without considering material such as jumper wires or breadboards, and the cost per component is shown in the Table 5.1. As for the Thermal Mapping Module, the total cost is 96 € without considering the cost of jumper wires and breadboards, and the cost of the components is shown in the Table 5.2. It is also worth mentioning that the majority of the prices were round up, in order to make the calculations easier. Another question to take into account is that, when it becomes necessary to build multiple modules, some components will become cheaper, since some companies reduce the cost per unit when a large quantity of the same component is purchased.

Table 5.1: BoM for the Alarm Module.

	Component	**Model Name**	**Quantity**	**Price per Unit**
	Development Board	NodeMCU ESP-12e	1	6 €
	Single Pixel IR Thermal Sensor	Melexis MLX90614	4	34 €
	I2C 0.96"OLED Display	Velleman VMA438	1	15 €
	O_2 Sensor	Grove ME2-O2	1	50 €
	GNSS Receiver	uBlox UART GPS NEO-7M	1	33 €
	Coloured LED	Red (1), Yellow (1) and Green (1) LED	3	0.12 €
Total	240.36 €			

Table 5.2: BoM for the Thermal Mapping Module.

	Component	**Model Name**	**Quantity**	**Price per Unit**
	Development Board	Espressif ESP32 DevKitC	1	9 €
	Multi Pixel IR Thermal Sensor	Melexis MLX90640	1	42 €
	I2C 0.96"OLED Display	Velleman VMA438	1	15 €
	GNSS Receiver	uBlox UART GPS NEO-7M	1	33 €
Total	96 €			

5.3 Future Work

In order to conclude the whole functional module, some features need to be included or tweaked.

The first feature to include will be the connection to a web server. A strong possibility is to connect the development board to the Google Sheets, using the WiFi capabilities of both ESP-12e and ESP32 DevKitC. This will allow to send data and save it in the form of a table of a Comma-Separated Values (CSV) file, which is easier to visualize and analyse data from.

Other feature to be implemented is the capability of the Thermal Mapping Module to scan from either sides of the vehicle, instead of just one fixed side. The solution will

be the inclusion of a second multi-pixel IR thermal sensor and a switch or button that allows to choose which of the sides to read from, making the module more versatile.

Given the problem with the speed and accuracy of the GNSS receiver, those questions will be addressed too. A solution to keep the customized settings of the receiver will be found and applied. This solution will lie in something like a battery that keeps feeding the GNSS receiver enough power to keep the configuration of the user. To try and get better accuracy, some test using an external antenna will be made, to verify if the signal is received with more accurate information regarding the GPS coordinates of the module.

When both modules are correctly tweaked and improved, the objective is to merge them together, creating a multi-function module that can work either as an alarm or to analyse the aftermath of wildfires. When doing so, only one development board will control the whole system, in this case, the ESP32 DevKitC. Then, the circuit will be moved from the breadboards to a printed circuit board, turning it into a more compact circuit. To finish it, a packaging for the circuit is going to be made, giving it extra protection from the heat, wind and rain. It will also make the module much easier to transport, use and apply in a vehicle.

Bibliography

[1] Lusa. "Incêndios: Quase 53 mil hectares arderam na região centro - dados provisórios." In: *Diário de Notícias* (June 2017). URL: https://www.dn.pt/lusa/interior/incendios-quase-53-mil-hectares-arderam-na-regiao-centro---dados-provisorios-8586699.html. accessed: 10/2018.

[2] "Números negros de Pedrógão: 66 mortos e mais de 250 feridos." In: *Jornal de Notícias* (Dec. 2017). URL: https://www.jn.pt/nacional/interior/sintese-pedrogao-grande-sessenta-e-seis-mortos-mais-de-250-feridos-e-500-casas-destruidas-8991727.html. accessed: 10/2018.

[3] M. Oliveira. "Bombeiros criticam estratégia de combate ao incêndio de Monchique." In: *Público* (Aug. 2018). URL: https://www.publico.pt/2018/08/08/sociedade/noticia/bombeiros-criticam-estrategia-de-combate-no-incendio-de-monchique-1840416. accessed: 01/2019.

[4] M. Béu. "Portugal pede ajuda a Bruxelas para mapear fogos de Monchique. Área ardida chega a 20 mil hectares." In: *Observador* (Aug. 2018). URL: https://observador.pt/2018/08/07/portugal-pede-ajuda-a-bruxelas-para-mapear-fogos-de-monchique-area-ardida-chega-a-20-mil-hectares/. accessed: 01/2019.

[5] J. R. Gyorki. *Understanding the Infrared Temperature Sensor*. Sept. 2009. URL: https://www.sensortips.com/temperature/infrared-temperature-sensor/. accessed: 10/2018.

[6] *Stefan-Boltzmann Law*. URL: http://hyperphysics.phy-astr.gsu.edu/hbase/thermo/stefan.html. accessed: 11/2018.

[7] Fluke. *What is Emissivity?* URL: https://www.flukeprocessinstruments.com/en-us/service-and-support/knowledge-center/infrared-technology/what-is-emissivity. accessed: 11/2018.

[8] *Blackbody Radiation*. URL: http://hyperphysics.phy-astr.gsu.edu/hbase/mod6.html. accessed: 11/2018.

[9] P. Jacobs. *Thermal infrared characterization of ground targets and backgrounds*. 2nd ed. Jan. 2006, pp. 1–184. DOI: 10.1117/3.651915.

[10] N. Paulino, A. S. Garção, and J. Goes. "UWB RADAR Receiver Architecture." In: *Low Power UWB CMOS Radar Sensors*. Springer, Jan. 2008, pp. 49–113. ISBN: 978-1-4020-8409-6. DOI: 10.1007/978-1-4020-8410-2.

[11] Y. Rauste, E. Herland, H. Frelander, K. Soini, T. Kuoremaki, and A. Ruokari. "Satellite-based forest fire detection for fire control in boreal forests." In: *International Journal of Remote Sensing* 18.12 (1997), pp. 2641–2656. DOI: 10.1080/014311697217512.

[12] A. Chamankar, S. Mohammadi, and M. Jamshidian. "Fire detection with image processing and PIR sensor." In: *International journal of Science and Engineering* 1.4 (2012).

[13] OMEGA. *Introduction to Non contact Infrared Thermometers*. URL: https://sea.omega.com/my/prodinfo/infraredthermometer.html. accessed: 10/2018.

[14] E. TollBox. *Radiation Heat Transfer*. 2003. URL: https://www.engineeringtoolbox.com/radiation-heat-transfer-d_431.html. accessed: 10/2018.

[15] Melexis. *MLX90614 family. Single and Dual Zone Infra Red Thermometer in TO-39*. Rev 009. Melexis. July 2015.

[16] *Photodiode Working Principle, Characteristics and Applications*. URL: https://www.elprocus.com/photodiode-working-principle-applications/. accessed: 11/2018.

[17] *Collins English Dictionary – Complete and Unabridged*. 12th ed. *thermopile*. HarperCollins, 2014. URL: https://www.thefreedictionary.com/thermopile. accessed: 11/2018.

[18] *Collins English Dictionary – Complete and Unabridged*. 12th ed. *bolometer*. HarperCollins, 2014. URL: https://www.thefreedictionary.com/bolometer. accessed: 11/2018.

[19] *Dictionary of the English Language*. 5th ed. *pyroelectricity*. Houghton Mifflin Harcourt Publishing Company, 2016. URL: https://www.thefreedictionary.com/pyroelectricity. accessed: 11/2018.

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

[21] TechTarget. *Satellite Constellation*. Mar. 2017. URL: https://whatis.techtarget.com/definition/satellite-constellation. accessed: 12/2018.

[22] GPS.gov. *Space Segment*. URL: https://www.gps.gov/systems/gps/space/. accessed: 12/2018.

[23] 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: 12/2018.

[24] Information and Analysis Center for Positioning, Navigation and Timing. *GLONASS History*. URL: https://www.glonass-iac.ru/en/guide/index.php. accessed: 12/2018.

[25] European GSA. *Galileo is the European global satellite-based navigation system*. URL: https://www.gsa.europa.eu/european-gnss/galileo/galileo-european-global-satellite-based-navigation-system. accessed: 12/2018.

[26] European GSA. *FAQ - How many satellites will Galileo have?* URL: https://www.gsa.europa.eu/european-gnss/galileo/faq#satellites. accessed: 12/2018.

[27] RACELOGIC. *How does DGPS (Differential GPS) work?* URL: https://racelogic.support/@api/deki/pages/488/pdf/How\%2bdoes\%2bDGPS\%2b(Differential\%2bGPS)\%2bwork\%253F.pdf?stylesheet=default. accessed: 12/2018.

[28] ESA. *System Design Details*. 2014. URL: https://gssc.esa.int/navipedia/index.php/Receiver_Block_Diagram. accessed: 12/2018.

[29] D. Plaušinaitis. *GNSS Receiver Front-ends II: Components*. 2008. URL: http://kom.aau.dk/~dpl/courses/mm08_slides.pdf. accessed: 12/2018.

[30] R. Keim. "Understanding Automatic Gain Control." In: (Nov. 2016). URL: https://www.allaboutcircuits.com/technical-articles/understanding-automatic-gain-control/. accessed: 12/2018.

[31] ESA. *Digital Signal Processing*. 2014. URL: https://gssc.esa.int/navipedia/index.php/Digital_Signal_Processing. accessed: 12/2018.

[32] ESA. *Baseband Processing*. 2014. URL: https://gssc.esa.int/navipedia/index.php/Baseband_Processing. accessed: 12/2018.

[33] ESA. *Tracking Loops*. 2014. URL: https://gssc.esa.int/navipedia/index.php/Tracking_Loops. accessed: 12/2018.

[34] OpenSreetMap Wiki. *GPS Chipset*. URL: https://wiki.openstreetmap.org/wiki/GPS_Chipset. accessed: 11/2018.

[35] M. Brain. *How Microcontrollers Work*. Apr. 2000. URL: https://electronics.howstuffworks.com/microcontroller1.htm. accessed: 12/2018.

[36] PC MAG. *Definition of: microcontroller*. URL: https://www.pcmag.com/encyclopedia/term/46924/microcontroller. accessed: 12/2018.

[37] *Collins English Dictionary – Complete and Unabridged*. 12th ed. *nonvolatile memory*. HarperCollins, 2014. URL: https://www.thefreedictionary.com/nonvolatile+memory. accessed: 12/2018.

[38] "12 Best Development Boards for DIY Projects." In: *Mepits* (Jan. 2015). URL: https://www.mepits.com/project/236/diy-projects/12-best-development-boards-for-diy-projects. accessed: 01/2019.

[39] *Arduino Website*. URL: https://www.arduino.cc. accessed: 01/2019.

[40] N. Heath. "How the Raspberry Pi was created: A visual history of the $35 board." In: *TechRepublic* (Dec. 2018). URL: https://www.techrepublic.com/pictures/how-the-raspberry-pi-was-created-a-visual-history-of-the-35-board/. accessed: 01/2019.

[41] *Raspberry Pi Products*. URL: https://www.raspberrypi.org/products/. accessed: 01/2019.

[42] *Microcomputador Raspberry Pi 3 Model B+ 1.4GHz 1GB - com WiFi 2.4/5GHz + Bluetooth 4.2 + PoE*. URL: https://mauser.pt/catalog/product_info.php?products_id=81510&src=raspberrypi. accessed: 01/2019.

[43] K. Chester. "The Internet of Things: a Look at Embedded WiFi Development Boards." In: *The Fusion Digital Journal* (Aug. 2015). URL: https://blog.fusiondigital.io/the-internet-of-things-a-look-at-embedded-wifi-development-boards-7abee1311711. accessed: 01/2019.

[44] *NodeMCU ESP8266 Placa Lua Internet de Desenvolvimento com WiFi - PRETO*. URL: https://pt.gearbest.com/transmitters-receivers-module/pp_366523.html. accessed: 01/2019.

[45] A. Moreira, P. Prats-Iraola, M. Younis, G. Krieger, I. Hajnsek, and K. P. Papathanassiou. "A tutorial on synthetic aperture radar." In: *IEEE Geoscience and Remote Sensing Magazine* 1.1 (Mar. 2013), pp. 6–43. ISSN: 2168-6831. DOI: 10.1109/MGRS.2013.2248301.

[46] M. Parker. "Radar basics – Part 5: synthetic aperture radar." In: *EE Times* (July 2011). URL: https://www.eetimes.com/document.asp?doc_id=1278931. accessed: 01/2019.

[47] C. Wolff. *Synthetic Aperture Radar*. URL: http://www.radartutorial.eu/20.airborne/ab07.en.html. accessed: 01/2019.

[48] M. Inggs and R. Lord. "Applications of satellite imaging radar." In: *South African Institute of Electrical Engineers, SAIEE, South Africa* (2000).

[49] E. Saraiva, R. Soares, A. Batista, H. Tertuliano, and A. Gomes Held. "Weather Radar: An Efficient Tool For Forest Fire Detection." In: *Proceedings of the 15th International Conference on Automatic Fire Detection (Aube'14), Duisburg, Germany*. Oct. 2014, pp. 14–16. URL: https://www.researchgate.net/profile/Antonio_Batista3/publication/276954869_Weather_Radar_An_Efficient_Tool_For_Forest_Fire_Detection/links/55c11b0f08ae092e96683f14.pdf.

[50] A. Maier, A. Sharp, and Y. Vagapov. "Comparative analysis and practical implementation of the ESP32 microcontroller module for the internet of things." In: *2017 Internet Technologies and Applications (ITA)*. Sept. 2017, pp. 143–148. DOI: 10.1109/ITECHA.2017.8101926.

[51] Espressif Systems. *ESP32-WROOM-32 Datasheet*. V2.8. Espressif Systems. Jan. 2019.

[52] Espressif Systems. *ESP8266EX Datasheet*. V6.0. Espressif Systems. Nov. 2018.

[53] Melexis. *MLX90640 32x24 IR Array Datasheet*. Rev 011. Melexis. Aug. 2018.

[54] S. Quinn. *Using ESP8266 SPIFFS*. URL: https://www.instructables.com/id/Using-ESP8266-SPIFFS/. accessed: 01/2019.

[55] ESP8266 Community Forum. *How to get a file from SPIFFS via upread to the computer*. Aug. 2016. URL: https://www.esp8266.com/viewtopic.php?f=32&t=4570. accessed: 01/2019.

[56] J. Valdez and J. Becker. "Understanding the I2C bus." In: *Texas Instruments SLVA704* (2015).

[57] C. Ramsay. *Arduino and Multiple MLX90614 Sensors Take Two*. Sept. 2017. URL: https://chrisramsay.co.uk/posts/2017/09/arduino-and-multiple-mlx90614-sensors-take-two/. accessed: 03/2019.

[58] *GitHub ESP8266 GPS code*. May 2016. URL: https://github.com/mkconer/ESP8266_GPS. accessed: 04/2019.

[59] HamsterMap. URL: http://www.hamstermap.com/quickmap.php. accessed: 04/2019.

[60] *Manual - NEO-6M Ublox GPS V2*. V1.0. DROTEK. Jan. 2014. URL: https://drotek.com/ftp/pdf/ublox_EN.pdf.

[61] Seeed - The IoT Hardware Enabler. *Grove - Gas Sensor(O2)*. Sept. 2016. URL: https://github.com/SeeedDocument/Grove_Gas_Sensor_O2. accessed: 05/2019.

[62] *Safetygram 17, Dangers of oxygen-deficient atmospheres*. Air Products and Chemicals, Inc. 2014. URL: https://www.airproducts.com/~/media/Files/PDF/company/safetygram-17.pdf. accessed: 08/2019.

[63] SparkFun. *SparkFun MLX90650 Arduino Exmple*. June 2018. URL: https://github.com/sparkfun/SparkFun_MLX90640_Arduino_Example. accessed: 05/2019.

[64] Surtr Technology. *Arduino Contactless Thermometer with MLX90614 + OLED/LCD*. May 2019. URL: https://surtrtech.com/2019/05/04/arduino-contactless-thermometer-with-mlx90614-oled-lcd/. acessed: 06/2019.

[65] Adafruit Industries. *Adafruit_SSD1306*. Mar. 2015. URL: https://github.com/adafruit/Adafruit_SSD1306. accessed: 06/2019.

[66] Adafruit Industries. *Adafruit-GFX-Library*. Apr. 2015. URL: https://github.com/adafruit/Adafruit-GFX-Library. accessed: 06/2019.

ANNEX I

CODING

Listing I.1: MLX90614 testing code from Adafruit.

```
#include <Wire.h>
#include <Adafruit_MLX90614.h>

Adafruit_MLX90614 mlx = Adafruit_MLX90614();

void setup() {
  Serial.begin(9600);
  Serial.println("Adafruit MLX90614 test");
  mlx.begin();
}

void loop() {
  Serial.print("Ambient = ");
  Serial.print(mlx.readAmbientTempC());
  Serial.print("*C\tObject = ");
  Serial.print(mlx.readObjectTempC()); Serial.println("*C");
  Serial.print("Ambient = ");
  Serial.print(mlx.readAmbientTempF());
  Serial.print("*F\tObject = ");
  Serial.print(mlx.readObjectTempF()); Serial.println("*F");
  Serial.println();

  delay(500);
}
```

Listing I.2: SPIFFS example.

```
#include <string.h>
#include "FS.h"

bool spiffsActive = false;
#define TESTFILE "/testfile.txt"

void setup()
{
  Serial.begin(115200);
  delay(1000);

  // Start filing subsystem
  if (SPIFFS.begin()) {
      Serial.println("SPIFFS Active");
      Serial.println();
      spiffsActive = true;
  } else {
      Serial.println("Unable to activate SPIFFS");
  }
  delay(2000);
}

void loop()
{
  if (spiffsActive) {
    if (SPIFFS.exists(TESTFILE)) {
      File f = SPIFFS.open(TESTFILE, "r");
      if (!f) {
        Serial.print("Unable To Open '");
        Serial.print(TESTFILE);
        Serial.println("' for Reading");
        Serial.println();
      } else {
        String s;
        Serial.print("Contents of file '");
        Serial.print(TESTFILE);
        Serial.println("'");
        Serial.println();
        while (f.position()<f.size())
        {
          s=f.readStringUntil('\n');
          s.trim();
          Serial.println(s);
        }
        f.close();
      }
      Serial.println();
```

```
48
49      f = SPIFFS.open(TESTFILE, "a");
50      if (!f) {
51        Serial.print("Unable To Open '");
52        Serial.print(TESTFILE);
53        Serial.println("' for Appending");
54        Serial.println();
55      } else {
56        Serial.print("Appending line to file '");
57        Serial.print(TESTFILE);
58        Serial.println("'");
59        Serial.println();
60        f.println("This line has been appended");
61        f.close();
62      }
63
64      f = SPIFFS.open(TESTFILE, "r");
65      if (!f) {
66        Serial.print("Unable To Open '");
67        Serial.print(TESTFILE);
68        Serial.println("' for Reading");
69        Serial.println();
70      } else {
71        String s;
72        Serial.print("Contents of file '");
73        Serial.print(TESTFILE);
74        Serial.println("' after append");
75        Serial.println();
76        while (f.position()<f.size())
77        {
78          s=f.readStringUntil('\n');
79          s.trim();
80          Serial.println(s);
81        }
82        f.close();
83      }
84
85    } else {
86      Serial.print("Unable To Find ");
87      Serial.println(TESTFILE);
88      Serial.println();
89    }
90  }
91
92  while (true){
93    yield();
94  }
95 }
```

Listing I.3: Saving MLX90614 data using SPIFFS.

```
#include <TimeLib.h>
#include <Wire.h>
#include "FS.h"
#include <Adafruit_MLX90614.h>
#include <string.h>
Adafruit_MLX90614 mlx = Adafruit_MLX90614();
bool spiffsActive = false;
#define TESTFILE "/long_range.txt"
/*PrintWriter output;*/
int time1;
int time2;
const int button = 15;
int temp = 0;
int prev = 0;
int ligado = 0;
int waiting = 1;
int prev_state = 1;//waiting
void setup()
{

  Serial.begin(115200);
  mlx.begin();
  pinMode(14, OUTPUT); // Initialize the LED_BUILTIN pin as an output 14=D5
  pinMode(button, INPUT); // 15 = D8
  // Start filing subsystem
  if (SPIFFS.begin()) {
      Serial.println("SPIFFS Active");
      Serial.println();
      spiffsActive = true;
  } else {
      Serial.println("Unable to activate SPIFFS");
  }
  Serial.println(hour(now()));
  delay(2000);
}
void loop(){
  temp = digitalRead(button);
  int state;
  if (temp == HIGH && prev == LOW) {
    state = 1;
    Serial.println("Start recording values");
    if (spiffsActive) {
      if (SPIFFS.exists(TESTFILE)) {
          String s = "";
          int i = 0;
          digitalWrite(14, HIGH);
          time1 = millis();
          while(i<200){
              s += mlx.readObjectTempC();
```

```
50            s += "\n\r";
51            //f.print(mlx.readObjectTempC());
52            //f.println("*C");
53            //f.println();
54            i++;
55            delay(100);
56          }
57        time2=millis();
58        int tottime = time2-time1;
59        s += "\n\r";
60        s += "Reading period: ";
61        s += tottime;
62        s += " milliseconds";
63        s += "\n\r";
64        digitalWrite(14, LOW);
65        File f = SPIFFS.open(TESTFILE, "w");
66        if (!f) {
67          Serial.print("Unable To Open '");
68          Serial.print(TESTFILE);
69          Serial.println("' for Appending");
70          Serial.println();
71
72        } else {
73          Serial.print("Appending values to file '");
74          Serial.print(TESTFILE);
75          Serial.println("'");
76          Serial.println();
77          f.println(s);
78          f.close();
79
80       }
81
82     } else {
83       Serial.print("Unable To Find ");
84       Serial.println(TESTFILE);
85       Serial.println();
86     }
87     Serial.println("Done!");
88    }
89  }
90  else{
91    state = 0; //waiting
92  }
93  if(prev_state != state && state == 0){
94    Serial.println("Waiting...");
95  }
96  prev = temp;
97  prev_state = state;
98  delay(50);
99 }
```

Listing I.4: ESP8266 Web Server example.

```
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>
#include <ESP8266mDNS.h>

const char* ssid = ".......";
const char* password = ".......";

ESP8266WebServer server(80);

const int led = 13;

void handleRoot() {
  digitalWrite(led, 1);
  server.send(200, "text/plain", "hello from esp8266!");
  digitalWrite(led, 0);
}

void handleNotFound(){
  digitalWrite(led, 1);
  String message = "File Not Found\n\n";
  message += "URI: ";
  message += server.uri();
  message += "\nMethod: ";
  message += (server.method() == HTTP_GET)?"GET":"POST";
  message += "\nArguments: ";
  message += server.args();
  message += "\n";
  for (uint8_t i=0; i<server.args(); i++){
    message += " " + server.argName(i) + ": " + server.arg(i) + "\n";
  }
  server.send(404, "text/plain", message);
  digitalWrite(led, 0);
}

void setup(void){
  pinMode(led, OUTPUT);
  digitalWrite(led, 0);
  Serial.begin(115200);
  WiFi.begin(ssid, password);
  Serial.println("");

  // Wait for connection
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
```

```
48   Serial.println("");
49   Serial.print("Connected to ");
50   Serial.println(ssid);
51   Serial.print("IP address: ");
52   Serial.println(WiFi.localIP());
53
54   if (MDNS.begin("esp8266")) {
55     Serial.println("MDNS responder started");
56   }
57
58   server.on("/", handleRoot);
59
60   server.on("/inline", [](){
61     server.send(200, "text/plain", "this works as well");
62   });
63
64   server.onNotFound(handleNotFound);
65
66   server.begin();
67   Serial.println("HTTP server started");
68 }
69
70 void loop(void){
71   server.handleClient();
72 }
```

Listing I.5: Sending MLX90614 data saved using SPIFFS to Web page.

```
1  #include <ESP8266WiFi.h>
2  #include <WiFiClient.h>
3  #include <ESP8266WebServer.h>
4  #include <ESP8266mDNS.h>
5  #include "FS.h"
6  #include <string.h>
7  Adafruit_MLX90614 mlx = Adafruit_MLX90614();
8  const char* ssid = "ELEC_Mestrado";
9  const char* password = "nmos130nm";
10 bool spiffsActive = false;
11 #define TESTFILE "/long_range.txt"
12 ESP8266WebServer server(80);
13
14 const int led = 13;
15
16 void handleRoot() {
17   digitalWrite(led, 1);
18   server.send(200, "text/plain", "hello from esp8266!");
19   digitalWrite(led, 0);
20 }
21
```

```
22 void handleNotFound(){
23   digitalWrite(led, 1);
24   String message = "File Not Found\n\n";
25   message += "URI: ";
26   message += server.uri();
27   message += "\nMethod: ";
28   message += (server.method() == HTTP_GET)?"GET":"POST";
29   message += "\nArguments: ";
30   message += server.args();
31   message += "\n";
32   for (uint8_t i=0; i<server.args(); i++){
33     message += " " + server.argName(i) + ": " + server.arg(i) + "\n";
34   }
35   server.send(404, "text/plain", message);
36   digitalWrite(led, 0);
37 }
38 
39 void handleDownload(){
40   if (!SPIFFS.begin()) {
41     Serial.println("SPIFFS failed to mount !\r\n");
42   }
43   else {
44     String str = "";
45     File f = SPIFFS.open(server.arg(0), "r");
46     if (!f) {
47       Serial.println("Can't open SPIFFS file !\r\n");
48     }
49     else {
50       char buf[1024];
51       int siz = f.size();
52       while(siz > 0) {
53         size_t len = std::min((int)(sizeof(buf) - 1), siz);
54         f.read((uint8_t *)buf, len);
55         buf[len] = 0;
56         str += buf;
57         siz -= sizeof(buf) - 1;
58       }
59       f.close();
60       server.send(200, "text/plain", str);
61     }
62   }
63 }
64 
65 void setup(void){
66   pinMode(led, OUTPUT);
67   digitalWrite(led, 0);
68   Serial.begin(115200);
69   WiFi.begin(ssid, password);
70   Serial.println("");
71   mlx.begin();
```

```
72   pinMode(14, OUTPUT);
73
74   // Start filing subsystem
75   if (SPIFFS.begin()) {
76       Serial.println("SPIFFS Active");
77       Serial.println();
78       spiffsActive = true;
79   } else {
80       Serial.println("Unable to activate SPIFFS");
81   }
82   // Wait for connection
83   while (WiFi.status() != WL_CONNECTED) {
84     delay(500);
85     Serial.print(".");
86   }
87   Serial.println("");
88   Serial.print("Connected to ");
89   Serial.println(ssid);
90   Serial.print("IP address: ");
91   Serial.println(WiFi.localIP());
92
93   if (MDNS.begin("esp8266")) {
94     Serial.println("MDNS responder started");
95   }
96
97   server.on("/", handleRoot);
98   server.on("/download", handleDownload);
99   server.on("/inline", [](){
100    server.send(200, "text/plain", "this works as well");
101  });
102
103  server.onNotFound(handleNotFound);
104
105  server.begin();
106  Serial.println("HTTP server started");
107 }
108
109
110 void loop(void){
111   digitalWrite(14, HIGH);
112   server.handleClient();
113   delay(100);
114   digitalWrite(14, LOW);
115 }
```

Listing I.6: Changing MLX90614 address.

```
#include "i2cmaster.h"

// New slave address (default is 0x5A)
byte NewMLXAddr = 0x50;

void setup(){
  Serial.begin(9600);
  Serial.println("Setup...");
  // Initialise some stuff
  i2c_init();
  PORTC = (1 << PORTC4) | (1 << PORTC5);
  delay(5000);
  // Read on universal address (0)
  ReadAddr(0);
  // Change to new address NewMLXAddr
  ChangeAddr(NewMLXAddr, 0x00);
  // Read on universal address (0)
  ReadAddr(0);
  Serial.println("**---READY---**");
  // Signal user to cycle power
  Serial.println("> [setup] Cycle power NOW to store new address - you have 10
      ↪ seconds");
  delay(10000);
  // Read on universal address (0)
  ReadAddr(0);
  Serial.println(" Warning, next ReadTemp() may fail if address has not been set.");
  ReadTemp(NewMLXAddr);
  Serial.println("**---DONE---**");
}
/**
* Read temperature from MLX at new address once setup() is done.
*
*/
void loop(){
    delay(5000);
    ReadTemp(NewMLXAddr);
    while(1){

    }
}
/**
* Changes the address of the MLX to NewAddr1.
*
*/
word ChangeAddr(byte NewAddr1, byte NewAddr2) {
  Serial.print("> [ChangeAddr] Will change address to: ");
  Serial.println(NewAddr1, HEX);
```

```
47    i2c_start_wait(0 + I2C_WRITE);
48    i2c_write(0x2E);
49    i2c_write(0x00);
50    i2c_write(0x00);
51
52    if (i2c_write(0x6F) == 0) {
53      i2c_stop();
54      Serial.println("> [ChangeAddr] Data erased.");
55    }
56    else {
57      i2c_stop();
58      Serial.println("> [ChangeAddr] Failed to erase data");
59      return -1;
60    }
61
62    Serial.print("> [ChangeAddr] Writing data: ");
63    Serial.print(NewAddr1, HEX);
64    Serial.print(", ");
65    Serial.println(NewAddr2, HEX);
66
67    for (int a = 0; a != 256; a++) {
68      i2c_start_wait(0 + I2C_WRITE);
69      i2c_write(0x2E);
70      i2c_write(NewAddr1);
71      i2c_write(NewAddr2);
72
73      if (i2c_write(a) == 0) {
74        i2c_stop();
75        delay(100);
76        Serial.print("> [ChangeAddr] Found correct CRC: 0x");
77        Serial.println(a, HEX);
78        return a;
79      }
80    }
81    i2c_stop();
82    Serial.println("> [ChangeAddr] Correct CRC not found");
83    return -1;
84  }
85  /**
86  * Reads the MLX address.
87  *
88  */
89  byte ReadAddr(byte MLXAddress) {
90    byte NewMLXAddress;
91    Serial.println("> [ReadAddr] Reading address");
92    if (MLXAddress == 0) {
93      Serial.print(" Using MLX univeral address");
94    } else {
95      Serial.print(" Using MLX address: ");
96      Serial.print(MLXAddress, HEX);
```

```
97    }
98    Serial.print(", Data: ");
99    i2c_start_wait(MLXAddress + I2C_WRITE);
100   i2c_write(0x2E);
101   i2c_rep_start(MLXAddress + I2C_READ);
102
103   NewMLXAddress = i2c_readAck();
104   Serial.print(NewMLXAddress, HEX);
105   Serial.print(", ");
106   Serial.print(i2c_readAck(), HEX);
107   Serial.print(", ");
108   Serial.println(i2c_readNak(), HEX);
109   i2c_stop();
110   return NewMLXAddress;
111 }
112 /**
113 * Utility function to read temperature from MLX at address
114 *
115 */
116 float ReadTemp(byte address) {
117   int data_low = 0;
118   int data_high = 0;
119   int pec = 0;
120   byte MLXAddress = address<<1;
121
122   Serial.print("> [ReadTemp] Read temperature ");
123   if (MLXAddress == 0) {
124     Serial.print("using MLX univeral address");
125   } else {
126     Serial.print("using MLX address ");
127     Serial.print(address, HEX);
128   };
129   Serial.print(": ");
130
131   i2c_start_wait(MLXAddress + I2C_WRITE);
132   i2c_write(0x07);
133   i2c_rep_start(MLXAddress + I2C_READ);
134   data_low = i2c_readAck();
135   data_high = i2c_readAck();
136   pec = i2c_readNak();
137   i2c_stop();
138
139   float temp = 0x0000;
140   temp = (float)(((data_high & 0x007F) << 8) + data_low);
141   temp = (temp * 0.02) - 273.16;
142   Serial.print(temp);
143   Serial.println(" C");
144   return temp;
145 }
```

Listing I.7: TinyGPS++ Basic Example.

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

static const int RXPin = 13, TXPin = 12;
static const uint32_t GPSBaud = 9600;
TinyGPSPlus gps;
SoftwareSerial ss(RXPin, TXPin);
char c ;

void setup()
{
  Serial.begin(115200);
  ss.begin(GPSBaud);

  Serial.println(F("DeviceExample.ino"));
  Serial.println(F("A simple demonstration of TinyGPS++ with an attached GPS module"));
  Serial.print(F("Testing TinyGPS++ library v. "));
      ↪ Serial.println(TinyGPSPlus::libraryVersion());
  Serial.println(F("by Mikal Hart"));
  Serial.println();
  delay(100);
}

void loop()
{
    displayInfo();
    delay(200);
    smartDelay(300);
  if (millis() > 5000 && gps.charsProcessed() < 10)
  {
    Serial.println(F("No GPS detected: check wiring."));
  }
}

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

```
47
48    Serial.print(F(" Date/Time: "));
49    if (gps.date.isValid())
50    {
51      Serial.print(gps.date.month());
52      Serial.print(F("/"));
53      Serial.print(gps.date.day());
54      Serial.print(F("/"));
55      Serial.print(gps.date.year());
56    }
57    else
58    {
59      Serial.print(F("INVALID"));
60    }
61
62    Serial.print(F(" "));
63    if (gps.time.isValid())
64    {
65      if (gps.time.hour() < 10) Serial.print(F("0"));
66      Serial.print(gps.time.hour());
67      Serial.print(F(":"));
68      if (gps.time.minute() < 10) Serial.print(F("0"));
69      Serial.print(gps.time.minute());
70      Serial.print(F(":"));
71      if (gps.time.second() < 10) Serial.print(F("0"));
72      Serial.print(gps.time.second());
73      Serial.print(F("."));
74      if (gps.time.centisecond() < 10) Serial.print(F("0"));
75      Serial.print(gps.time.centisecond());
76    }
77    else
78    {
79      Serial.print(F("INVALID"));
80    }
81
82    Serial.println();
83  }
84
85  static void smartDelay(unsigned long ms)
86  {
87    unsigned long start = millis();
88    do
89    {
90      while (ss.available())
91        gps.encode(ss.read());
92    } while (millis() - start < ms);
93  }
```

Listing I.8: Code to obtain GPS and temperature data.

```
#include <Wire.h>
#include "FS.h"
#include <Adafruit_MLX90614.h>
#include <string.h>
#include <TinyGPS++.h>
#include <SoftwareSerial.h>

bool spiffsActive = false;
const int button = 15;
const int led = 14;
int temp = 0;
int prev = 0;
int state;
int prev_state = 1;//waiting
static const int RXPin = 13, TXPin = 12;
static const uint32_t GPSBaud = 9600;
int inicio = 0;
String hor;
String minu;
String sec;

Adafruit_MLX90614 mlx = Adafruit_MLX90614(0x50);
TinyGPSPlus gps;
SoftwareSerial ss(RXPin, TXPin);
#define TESTFILE "/long_range.txt"

void setup() {
  Serial.begin(115200);
  ss.begin(GPSBaud);
  mlx.begin();
  pinMode(led, OUTPUT);
  pinMode(button, INPUT);
  if (SPIFFS.begin()) {
    Serial.println();
      Serial.println("SPIFFS Active");
      Serial.println();
      spiffsActive = true;
  } else {
      Serial.println("Unable to activate SPIFFS");
  }
  delay(1000);
}

void loop() {

     if(!(gps.location.isValid())){
       Serial.println("Waiting for signal...");
       delay(1000);
     }
```

```
50      if(gps.location.isValid() && inicio != 1){
51        Serial.println();
52        Serial.println("Signal acquired. Press button to start collecting data.");
53        Serial.println();
54        inicio=1;
55      }
56
57      temp = digitalRead(button);
58      delay(100);
59      if(temp == HIGH && prev == LOW){
60        Serial.println("Collecting...");
61        state=1;
62        String s = "Temp,Lat,Long,Time";
63        s+="\n\r";
64        Serial.println(s);
65        int i = 1;
66        digitalWrite(led,HIGH);
67        while(i<201){
68            float temp = mlx.readObjectTempC();
69            if(temp < 1037){
70              if (gps.time.hour() < 10){
71                hor = "0"+String(gps.time.hour());
72              } else{
73                hor = String(gps.time.hour());
74              }
75              if (gps.time.minute() < 10){
76                minu = "0"+String(gps.time.minute());
77              } else{
78                minu = String(gps.time.minute());
79              }
80              if (gps.time.second() < 10){
81                sec = "0"+String(gps.time.second());
82              } else{
83                sec = String(gps.time.second());
84              }
85              s+=String(temp)+",";
86              s+=String(gps.location.lat(),6)+",";
87              s+=String(gps.location.lng(),6)+",";
88              s+=hor+":"+minu+":"+sec+"."+String(gps.time.centisecond());
89              s+="\n\r";
90              Serial.println("Collected "+String(i)+" out of 200");
91              i++;
92              delay(100);
93              smartDelay(100);
94            }
95        }
96        Serial.println(s);
97      } else {
98        state = 0;
99      }
```

```
100       if(prev_state != state && state == 0){
101         Serial.println("Waiting...");
102       }
103       prev = temp;
104       prev_state = state;
105       delay(100);
106       smartDelay(300);
107
108       if (millis() > 5000 && gps.charsProcessed() < 10){
109         Serial.println(F("No GPS detected: check wiring."));
110         while(true);
111       }
112       delay(50);
113  }
114
115  static void smartDelay(unsigned long ms)
116  {
117    unsigned long start = millis();
118    do
119    {
120      while (ss.available())
121        gps.encode(ss.read());
122    } while (millis() - start < ms);
123  }
```

Listing I.9: Code to test the GNSS receiver through a long distance trip.

```
1
2   #include <TinyGPS++.h>
3   #include <SoftwareSerial.h>
4   #include <HardwareSerial.h>
5   /*
6     This sample sketch demonstrates the normal use of a TinyGPS++ (TinyGPSPlus) object.
7     It requires the use of SoftwareSerial, and assumes that you have a
8     4800-baud serial GPS device hooked up on pins 4(rx) and 3(tx).
9   */
10
11  static const int RXPin = 12, TXPin = 13; //nodemcu
12  static const uint32_t GPSBaud = 115200;
13  char c ;
14  TinyGPSPlus gps;
15  SoftwareSerial ss(RXPin, TXPin);
16
17  void setup()
18  {
19    Serial.begin(115200);
20    ss.begin(GPSBaud);
21
22    delay(100);
```

```
23   Serial.print(F("Location: "));
24   Serial.print(F(","));
25   Serial.println(F(" Date/Time: "));
26 }
27
28 void loop()
29 {
30     displayInfo();
31     smartDelay(150);
32
33   if (millis() > 5000 && gps.charsProcessed() < 10)
34   {
35     Serial.println(F("No GPS detected: check wiring."));
36   }
37   else{
38   }
39
40 }
41
42 void displayInfo()
43 {
44
45   if (gps.location.isValid())
46   {
47     Serial.print(gps.location.lat(), 6);
48     Serial.print(F(","));
49     Serial.print(gps.location.lng(), 6);
50   }
51   else
52   {
53     Serial.print(F("INVALID"));
54   }
55
56   if (gps.date.isValid())
57   {
58     Serial.print(F("; "));
59     Serial.print(gps.date.month());
60     Serial.print(F("/"));
61     Serial.print(gps.date.day());
62     Serial.print(F("/"));
63     Serial.print(gps.date.year());
64   }
65   else
66   {
67     Serial.print(F("INVALID"));
68   }
69
70   Serial.print(F("; "));
71   if (gps.time.isValid())
72   {
```

```
    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"));
  }

  Serial.println();
}

static void smartDelay(unsigned long ms)
{
  unsigned long start = millis();
  do
  {
    while (ss.available())
      gps.encode(ss.read());
  } while (millis() - start < ms);
}
```

Listing I.10: Main code for the function of reading oxygen levels in air.

```
#include "Oxygen.h"
#include "arduino.h"
#define O2_preheat_time 60000
bool O2_init_flag = 0;
float calibration_voltage;
#define O2_percentage 20.80

float O2_value()
{
  unsigned int sum = 0;
  if (O2_init_flag == 0)
  {
    O2_init_flag = 1;
    pinMode(O2_pin, INPUT);
    for (unsigned char i = 64;i > 0;i--)
    {
      sum = sum + analogRead(O2_pin);
```

```
18      delay(100);
19    }
20    sum = sum >> 6;
21    calibration_voltage = sum / O2_percentage;
22    return 20.80;
23  }
24  else
25  {
26    for (unsigned char i = 32;i > 0;i--)
27    {
28      sum = sum + analogRead(O2_pin);
29      delay(50);
30    }
31    sum = sum >> 5;
32    float output = sum / calibration_voltage;
33    return output;
34  }
35 }
```

Listing I.11: Header file for the function of reading oxygen levels in air.

```
1  // Oxygen.h
2
3  #ifndef _OXYGEN_h
4  #define _OXYGEN_h
5
6
7  #define O2_pin A0
8  extern bool O2_init_flag;
9  float O2_value();
10
11
12 #endif
```

Listing I.12: Full code for testing O2 sensor.

```
1  #include "Oxygen.h"
2  #include <Wire.h>
3  #include <Adafruit_MLX90614.h>
4
5  Adafruit_MLX90614 mlx = Adafruit_MLX90614(0x50);
6
7  void setup() {
8    Serial.begin(115200);
9    Serial.println("Calibration sensor...This needs one minute");
10   O2_value();
11   Serial.println("Finish Calibration");
12   mlx.begin();
```

```
  Serial.println("O2_concentration,Average_temp,Bottle_temp");
}

void loop() {
  float O2 = 0;
  O2 = O2_value();
  Serial.print(O2);
  Serial.print("%,");
  Serial.print(","); Serial.print(mlx.readAmbientTempC());
  Serial.print(","); Serial.println(mlx.readObjectTempC());
}
```

Listing I.13: Example number 2 for using MLX90640 IR thermal sensor in Arduino IDE.

```

/*
  Output the temperature readings to all pixels to be read by a Processing visualizer
  By: Nathan Seidle
  SparkFun Electronics
  Date: May 22nd, 2018
  License: MIT. See license file for more information but you can
  basically do whatever you want with this code.
  Feel like supporting open source hardware?
  Buy a board from SparkFun! https://www.sparkfun.com/products/14769
  This example outputs 768 temperature values as fast as possible. Use this example
  in conjunction with our Processing visualizer.
  This example will work with a Teensy 3.1 and above. The MLX90640 requires some
  hefty calculations and larger arrays. You will need a microcontroller with 20,000
  bytes or more of RAM.
  This relies on the driver written by Melexis and can be found at:
  https://github.com/melexis/mlx90640-library
  Hardware Connections:
  Connect the SparkFun Qwiic Breadboard Jumper (https://www.sparkfun.com/products/14425)
  to the Qwiic board
  Connect the male pins to the Teensy. The pinouts can be found here:
      ↪ https://www.pjrc.com/teensy/pinout.html
  Open the serial monitor at 115200 baud to see the output
*/

#include <Wire.h>

#include "MLX90640_API.h"
#include "MLX90640_I2C_Driver.h"

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

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

float mlx90640To[768];
```

```
35 paramsMLX90640 mlx90640;
36
37 void setup()
38 {
39   Wire.begin();
40   Wire.setClock(400000); //Increase I2C clock speed to 400kHz
41
42   Serial.begin(115200); //Fast serial as possible
43
44   while (!Serial); //Wait for user to open terminal
45   //Serial.println("MLX90640 IR Array Example");
46
47   if (isConnected() == false)
48   {
49     Serial.println("MLX90640 not detected at default I2C address. Please check wiring.
         ↪ Freezing.");
50     while (1);
51   }
52
53   //Get device parameters - We only have to do this once
54   int status;
55   uint16_t eeMLX90640[832];
56   status = MLX90640_DumpEE(MLX90640_address, eeMLX90640);
57   if (status != 0)
58     Serial.println("Failed to load system parameters");
59
60   status = MLX90640_ExtractParameters(eeMLX90640, &mlx90640);
61   if (status != 0)
62     Serial.println("Parameter extraction failed");
63
64   //Once params are extracted, we can release eeMLX90640 array
65
66   //MLX90640_SetRefreshRate(MLX90640_address, 0x02); //Set rate to 2Hz
67   MLX90640_SetRefreshRate(MLX90640_address, 0x03); //Set rate to 4Hz
68   //MLX90640_SetRefreshRate(MLX90640_address, 0x07); //Set rate to 64Hz
69 }
70
71 void loop()
72 {
73   long startTime = millis();
74   for (byte x = 0 ; x < 2 ; x++)
75   {
76     uint16_t mlx90640Frame[834];
77     int status = MLX90640_GetFrameData(MLX90640_address, mlx90640Frame);
78
79     float vdd = MLX90640_GetVdd(mlx90640Frame, &mlx90640);
80     float Ta = MLX90640_GetTa(mlx90640Frame, &mlx90640);
81
82     float tr = Ta - TA_SHIFT; //Reflected temperature based on the sensor ambient
         ↪ temperature
```

```
    float emissivity = 0.95;

    MLX90640_CalculateTo(mlx90640Frame, &mlx90640, emissivity, tr, mlx90640To);
  }
  long stopTime = millis();

  for (int x = 0 ; x < 768 ; x++)
  {
    //if(x % 8 == 0) Serial.println();
    Serial.print(mlx90640To[x], 2);
    Serial.print(",");
  }
  Serial.println("");
}
//Returns true if the MLX90640 is detected on the I2C bus
boolean isConnected()
{
  Wire.beginTransmission((uint8_t)MLX90640_address);
  if (Wire.endTransmission() != 0)
    return (false); //Sensor did not ACK
  return (true);
}
```

Listing I.14: Custom code for MLX90640 readings with corrections , mirrored image and non-uniform mean filter.

```
#include <Wire.h>
#include "MLX90640_API.h"
#include "MLX90640_I2C_Driver.h"

const byte MLX90640_address = 0x33; //Default 7-bit unshifted address of the MLX90640
#define TA_SHIFT 8 //Default shift for MLX90640 in open air

float mlx90640To[768];
paramsMLX90640 mlx90640;
uint16_t mlx90640Frame[834];
float vdd, Ta, tr, emissivity = 0.95;

void setup()
{
  Wire.begin();
  Wire.setClock(400000); //Increase I2C clock speed to 400kHz
  Serial.begin(921600);
  while (!Serial); //Wait for user to open terminal
  if (isConnected() == false){
    Serial.println("MLX90640 not detected at default I2C address. Please check wiring.
        ↪ Freezing.");
    while (1);
  }
```

```
23   int status;
24   uint16_t eeMLX90640[832];
25   status = MLX90640_DumpEE(MLX90640_address, eeMLX90640);
26   if (status != 0)
27     Serial.println("Failed to load system parameters");
28
29   status = MLX90640_ExtractParameters(eeMLX90640, &mlx90640);
30   if (status != 0)
31     Serial.println("Parameter extraction failed");
32
33   MLX90640_SetRefreshRate(MLX90640_address, 0x02); //2Hz refresh freq
34
35   Wire.setClock(1000000);
36 }
37
38 void loop()
39 {
40      String s="";
41      int k = 0;
42      for (byte x = 0 ; x < 2 ; x++){
43        status = MLX90640_GetFrameData(MLX90640_address, mlx90640Frame);
44        vdd = MLX90640_GetVdd(mlx90640Frame, &mlx90640);
45        Ta = MLX90640_GetTa(mlx90640Frame, &mlx90640);
46        tr = Ta - TA_SHIFT; //Reflected temperature based on the sensor ambient
              ↪ temperature
47        MLX90640_CalculateTo(mlx90640Frame, &mlx90640, emissivity, tr, mlx90640To);
48      }
49      int y = 0;
50      int breakline = 0;
51      for (y = 0; y < 24; y++){
52      k = y*32;
53      for (int x = 31+k ; x >= 0+k ; x--){
54        int div1 = (x+1)%32;
55        int div2= x%32;
56        float valor;
57
58        if (isnan(mlx90640To[x])) {
59          mlx90640To[x] = (mlx90640To[x+1] + mlx90640To[x-1])/2;
60        }
61        if(x < 734 && x > 32 && div1!=0 && div2 !=0){
62          valor = ((mlx90640To[x]*4) + (mlx90640To[x+1]*2) + (mlx90640To[x-1]*2) +
                ↪ (mlx90640To[x+32]*2) + (mlx90640To[x-32]*2) + mlx90640To[x-31] +
                ↪ mlx90640To[x-33] + mlx90640To[x+31] + mlx90640To[x+33]) / 16;
63        }else{
64           valor = mlx90640To[x];
65        }
66        breakline++;
67        if(breakline == 32){
68           breakline = 0;
69           s=s+String(valor,2)+"\n";
```

```
      }
      else{
          s=s+String(valor,2)+",";
      }
     }
  }
  s=s+"\n\r";
  Serial.println(s);
  delay(100);
}

boolean isConnected()
{
  Wire.beginTransmission((uint8_t)MLX90640_address);
  if (Wire.endTransmission() != 0)
    return (false); //Sensor did not ACK
  return (true);
}
```

Listing I.15: Code for the Aftermath Analysis module.

```
#include <TinyGPS++.h>
#include <HardwareSerial.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include "MLX90640_API.h"
#include "MLX90640_I2C_Driver.h"
TinyGPSPlus gps;
HardwareSerial SerialGPS(1);

#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define OLED_RESET -1
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);
const byte MLX90640_address = 0x33;
static const int RXPin = 13, TXPin = 12;
static const uint32_t GPSBaud = 9600;
String hor;
String minu;
String sec;
const int button = 15;
const int led = 14;
#define TA_SHIFT 8
float mlx90640To[768];
paramsMLX90640 mlx90640;
int fire_level = 0;

uint16_t mlx90640Frame[834];
```

```
29 int status;
30 float vdd, Ta, tr, emissivity = 0.95;
31
32 void setup()
33 {
34   Wire.begin();
35   Wire.setClock(400000);
36   Serial.begin(921600);
37   SerialGPS.begin(9600, SERIAL_8N1, 16, 17);
38   while (!Serial);
39
40   if (isConnected() == false)
41   {
42     Serial.println("MLX90640 not detected at default I2C address. Please check wiring.
         ↪ Freezing.");
43     while (1);
44   }
45
46   int status;
47   uint16_t eeMLX90640[832];
48   status = MLX90640_DumpEE(MLX90640_address, eeMLX90640);
49   if (status != 0)
50     Serial.println("Failed to load system parameters");
51
52   status = MLX90640_ExtractParameters(eeMLX90640, &mlx90640);
53   if (status != 0)
54     Serial.println("Parameter extraction failed");
55
56   MLX90640_SetRefreshRate(MLX90640_address, 0x02);
57
58   display.begin(SSD1306_SWITCHCAPVCC, 0x3C);
59   display.clearDisplay();
60   display.setTextColor(BLACK, WHITE);
61   display.setTextSize(2);
62   display.setCursor(0,20);
63   display.println(" WELCOME ");
64   display.display();
65   delay(2000);
66   display.clearDisplay();
67   display.setTextSize(1);
68   display.setTextColor(WHITE);
69   display.setCursor(0,0);
70   display.println("Calibration sensor...This needs one minute");
71   display.display();
72   O2_value();
73   delay(3000);
74   display.setTextColor(WHITE);
75   display.setCursor(0,17);
76   display.println("Finish Calibration");
77   display.display();
```

```
78   Wire.setClock(1000000);
79   Serial.println("pre ciclo do sensor, pos ciclo do sensor");
80   Serial.println();
81 }
82
83 void loop()
84 {
85     float highest_temp = 0;
86      String s="";
87      int k = 0;
88
89      for (byte x = 0 ; x < 2 ; x++)
90      {
91        status = MLX90640_GetFrameData(MLX90640_address, mlx90640Frame);
92        vdd = MLX90640_GetVdd(mlx90640Frame, &mlx90640);
93        Ta = MLX90640_GetTa(mlx90640Frame, &mlx90640);
94        tr = Ta - TA_SHIFT;
95        MLX90640_CalculateTo(mlx90640Frame, &mlx90640, emissivity, tr, mlx90640To);
96      }
97      int y = 0;
98      int breakline = 0;
99      for (y = 0; y < 24; y++){
100
101       k = y*32;
102         for (int x = 31+k ; x >= 0+k ; x--)
103         {
104           int div1 = (x+1)%32;
105           int div2= x%32;
106           float valor;
107           if (isnan(mlx90640To[x])) {
108             mlx90640To[x] = (mlx90640To[x+1] + mlx90640To[x-1])/2;
109
110           }
111           if(x < 734 && x > 32 && div1!=0 && div2 !=0){
112             valor = ((mlx90640To[x]*4) + (mlx90640To[x+1]*2) + (mlx90640To[x-1]*2) +
                  ↪ (mlx90640To[x+32]*2) + (mlx90640To[x-32]*2) + mlx90640To[x-31] +
                  ↪ mlx90640To[x-33] + mlx90640To[x+31] + mlx90640To[x+33]) / 16;
113           }else{
114             valor = mlx90640To[x];
115           }
116
117           breakline++;
118           if(highest_temp < valor){
119             highest_temp = valor;
120           }
121
122           if(breakline == 32){
123             breakline = 0;
124             s=s+String(valor,2)+"\n";
125           }else{
```

```
126             s=s+String(valor,2)+",";
127           }
128         }
129
130     }
131     s=s+"\n\r";
132     Serial.println(s);
133
134     state_of_fire(highest_temp);
135     int sdelay = 1030-(millis()-init_time);
136     if(sdelay < 150){
137       sdelay = 150;
138     }
139     smartDelay(sdelay);
140     Serial.print(init_time);
141     Serial.print(",");
142     Serial.print(millis());
143     Serial.print(",");
144     Serial.print(gps.satellites.value());
145     Serial.print(",");
146     Serial.println(sdelay);
147 }
148
149
150 boolean isConnected()
151 {
152   Wire.beginTransmission((uint8_t)MLX90640_address);
153   if (Wire.endTransmission() != 0)
154     return (false);
155   return (true);
156 }
157
158 static void smartDelay(unsigned long ms)
159 {
160   unsigned long start = millis();
161   do
162   {
163     while (SerialGPS.available())
164       gps.encode(SerialGPS.read());
165   } while (millis() - start < ms);
166 }
167
168 void state_of_fire(float highest_temp){
169   gps_time();
170   display.clearDisplay();
171   display.setTextColor(WHITE);
172   display.setCursor(10,1);
173   display.setTextSize(1);
174   display.println(hor+":"+minu+":"+sec);
175   display.setTextSize(2);
```

```
  if(highest_temp > 250){
        display.setCursor(10,9);
        display.println("DANGER");
        display.setCursor(5,35);
        display.println("FIRE VERY CLOSE");
     }else{
       if(highest_temp > 100){
        display.setCursor(5,20);
        display.println("Fire focus close");
       }else{
        if(highest_temp > 40){
          display.setCursor(10,5);
          display.println("Possible fire at distance");
        }else{
          display.setCursor(10,20);
          display.println("No danger of fire");
        }
       }
     }
  display.display();
}

void gps_time(){
  if (gps.time.hour() < 10){
    hor = "0"+String(gps.time.hour());
  } else{
    hor = String(gps.time.hour());
  }
  if (gps.time.minute() < 10){
    minu = "0"+String(gps.time.minute());
  } else{
    minu = String(gps.time.minute());
  }
  if (gps.time.second() < 10){
    sec = "0"+String(gps.time.second());
  } else{
    sec = String(gps.time.second());
  }
}
```

Listing I.16: Code for the Alarm module.

```
#include "Oxygen.h"
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <Wire.h>
#include <Adafruit_MLX90614.h>
#include <TinyGPS++.h>
#include <SoftwareSerial.h>
```

```
static const int RXPin = 12, TXPin = 13, LED_VERDE = 14, LED_AMARELO = 10, LED_VERMELHO
    ↪ = 16;
int luzverde=0, o2_low=0, body_present = 0, fire = 0;
static const uint32_t GPSBaud = 9600;
float front = 25.00, back = 25.00, left = 25.00, right = 25.00;
float amb_f = 25.00, amb_b = 25.00, amb_l = 25.00, amb_r = 25.00;
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define OLED_RESET -1

Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);
Adafruit_MLX90614 mlx = Adafruit_MLX90614(0x50);
Adafruit_MLX90614 mlx2 = Adafruit_MLX90614(0x5A);
Adafruit_MLX90614 mlx3 = Adafruit_MLX90614(0x5B);
Adafruit_MLX90614 mlx4 = Adafruit_MLX90614(0x5C);
TinyGPSPlus gps;
SoftwareSerial ss(RXPin, TXPin);
void setup() {
  Serial.begin(115200);
  ss.begin(GPSBaud);
  pinMode(LED_VERDE, OUTPUT);
  pinMode(LED_AMARELO, OUTPUT);
  pinMode(LED_VERMELHO, OUTPUT);
  digitalWrite(LED_VERDE, HIGH);
  digitalWrite(LED_VERMELHO, LOW);
  digitalWrite(LED_AMARELO, LOW);
  display.begin(SSD1306_SWITCHCAPVCC, 0x3C); //Start the OLED display
  display.clearDisplay();

  display.setTextColor(WHITE);
  display.setCursor(0,0);
  display.println("Calibration sensor...This needs one minute");
  display.display();
  O2_value();
  delay(3000);
  display.setTextColor(WHITE);
  display.setCursor(0,17);
  display.println("Finish Calibration");
  display.display();
  mlx.begin();
  mlx2.begin();
  mlx3.begin();
  mlx4.begin();

}

void loop() {

  float O2 = O2_value();
  float k1 = mlx3.readObjectTempC();
```

```
57  if(k1 < 1037.00){
58    front = k1;
59  }
60  delay(10);
61  float k2 = mlx4.readObjectTempC();
62  if(k2 < 1037.00){
63    back = k2;
64  }
65  delay(10);
66  float k3 = mlx.readObjectTempC();
67  if(k3 < 1037){
68    left = k3;
69  }
70  delay(10);
71  float k4 = mlx2.readObjectTempC();
72  if(k4 < 1037){
73    right = k4;
74  }
75  delay(10);
76  k1 = mlx3.readAmbientTempC();
77  if(k1 < 1037){
78    amb_f = k1;
79  }
80  delay(10);
81  k2 = mlx4.readAmbientTempC();
82  if(k2 < 1037){
83    amb_b = k2;
84  }
85  delay(10);
86  k3 = mlx.readAmbientTempC();
87  if(k3 < 1037){
88    amb_l = k3;
89  }
90  delay(10);
91  k4 = mlx2.readAmbientTempC();
92  if(k4 < 1037){
93    amb_r = k4;
94  }
95  delay(10);
96
97  float avg_temp = (amb_f+amb_b+amb_l+amb_r)/4;
98  display.clearDisplay();
99  display.setTextSize(1);
100 display.setTextColor(WHITE);
101 display.setCursor(0,0);
102 display.println(" |" + String(front,2)+(char)247+"C|");
103 display.setCursor(0,10);
104 display.println("|"+String(left,2)+(char)247+"C| + |"+ String(right,2)+(char)247 +
        ↪ "C|");
105 display.setCursor(0,20);
```

```
106   display.println(" |"+ String(back,2)+(char)247+"C|");
107   display.setCursor(0,30);
108   display.println("O2:" + String(O2,2)+"% Amb:" + String(avg_temp,2) + (char)247+"C");
109   display.println("Lat: " + String(gps.location.lat(), 6) + " \n\rLong: " +
          ↪ String(gps.location.lng(), 6));
110   display.println(String(gps.date.day()) + "/" + String(gps.date.month()) + "/" +
          ↪ String(gps.date.year()));
111   display.display();
112
113
114   if(O2 < 19 && o2_low==0){
115     digitalWrite(LED_AMARELO, HIGH);
116     digitalWrite(LED_VERDE, LOW);
117     display.clearDisplay();
118     display.setTextColor(WHITE);
119     display.setCursor(0,30);
120     display.println("OXYGEN LEVELS ARE LOW");
121     display.display();
122     delay(2000);
123     o2_low=1;
124   }
125   if(O2>19.5 && o2_low==1){
126     digitalWrite(LED_AMARELO, LOW);
127     digitalWrite(LED_VERDE, HIGH);
128     display.clearDisplay();
129     display.setTextColor(WHITE);
130     display.setCursor(20,30);
131     display.println("OXYGEN LEVELS");
132     display.setCursor(30,40);
133     display.println("NORMALIZED");
134     display.display();
135     delay(2000);
136     o2_low=0;
137   }
138   if((front > 100 || back > 100 || left > 100 || right > 100) && fire == 0){
139     digitalWrite(LED_VERMELHO, HIGH);
140     digitalWrite(LED_VERDE, LOW);
141     display.clearDisplay();
142     display.setTextColor(WHITE);
143     display.setCursor(30,30);
144     display.println("FIRE DETECTED");
145     display.display();
146     fire=1;
147     delay(2000);
148   }
149    if(front < 100 && back < 100 && left < 100 && right < 100 && fire == 1){
150     digitalWrite(LED_VERMELHO, LOW);
151     digitalWrite(LED_VERDE, HIGH);
152     fire = 0;
153    }
```

```
154
155   Serial.println(String(gps.date.day()) + "/" + String(gps.date.month()) + "/" +
          ↪ String(gps.date.year()));
156   Serial.println(String(gps.satellites.value()));
157   smartDelay(400);
158 }
159 static void smartDelay(unsigned long ms)
160 {
161   unsigned long start = millis();
162   do
163   {
164     while (ss.available())
165       gps.encode(ss.read());
166   } while (millis() - start < ms);
167 }
```