Tracking of Hot Objects Using IR-Camera

Ivan Pascual Garrido (s192395)
Master of Science in Engineering
2020

DTU Electrical Engineering
Department of Electrical Engineering

Tracking of Hot Objects Using IR-Camera

Report written by:
Ivan Pascual Garrido (s192395)

Advisor(s):
Nils Axel Andersen, Associate Professor at the Electrical Engineering Department of DTU

DTU Electrical Engineering
Technical University of Denmark
2800 Kgs. Lyngby
Denmark

elektro@elektro.dtu.dk

Project period:	14 September 2020- 4 April 2021
ECTS:	35
Education:	M.Sc.
Field:	Electrical Engineering
Class:	Public
Edition:	1. edition
Remarks:	This report is submitted as partial fulfillment of the requirements for graduation in the above education at the Technical University of Denmark.
Copyrights:	©Ivan Pascual Garrido, 2020

Abstract

Detection of hot objects with thermal infrared cameras has historically been of interest mostly for military purposes. However, decreasing price and size combined with increasing image quality and resolution in recent years have opened up new application fields, such us firefighting missions, which is the focus of this project. In many cases, the detection of hot objects is not enough and their tracking is also necessary. That is why infrared cameras usually work together with mobile robots to provide the system with movement.

The main purpose of this project is to develop and implement a system consisting of an infrared camera and an autonomous mobile robot coordinated with each other to detect and track hot objects in an unknown environment. To achieve this, two thematic blocks corresponding to both devices fundamentally make up the proposed solution. Regarding the equipment selection, Lepton 3.5 from FLIR brand is the IR-camera for the main perception part of the system while the SMR developed by the Automation and Control department of DTU constitutes the mobile robot.

A Python script is responsible for implementing the algorithm to detect hot objects and its location in the images captured by the IR-camera, allowing to distinguish if those objects correspond to fires according to a pre-established temperature threshold. Thanks the continuous reading of this hot object data, a SMR-CL script handles the algorithm that allows the robot to perform the following sequence of tasks: check for any fire in the room, track and reach its location, extinguish it and return to the starting position. The communication between both scripts is implemented using a custom made plugin in C++.

The collected data during the experiments is shown by means of an infrared imaging animation developed in Python, and a simulation of the robot trajectory and direction map developed in Matlab. By analysing the results obtained in the final testing phase, it is worth to conclude that the solution constitutes a robust system capable of tracking both static and moving hot objects within different environments while the design is fully customizable to a real application.

Contents

Abstract **i**

Contents **iii**

1 Introduction **1**
1.1 Objectives . . . 1
1.2 Thesis outline . . . 3

2 Analysis **5**
2.1 Introduction . . . 5
2.2 Camera . . . 5
2.3 Robot . . . 6
2.4 Environment . . . 6
2.5 Software . . . 7
2.6 Modules . . . 8
2.7 Summary . . . 8

3 Hot object detection **11**
3.1 Introduction . . . 11
3.2 Theoretical background . . . 11
3.2.1 Infrared radiation . . . 12
3.2.2 Thermal imaging . . . 13
3.2.3 Features, advantages and limitations . . . 16
3.2.4 Applications . . . 17
3.3 Equipment description . . . 19
3.3.1 Hardware . . . 19
3.3.2 Software . . . 19
3.4 Design and implementation . . . 21
3.4.1 Approach . . . 21
3.4.2 Algorithm . . . 22
3.5 Module test . . . 25
3.6 Summary . . . 26

4 Hot object tracking **29**
4.1 Introduction . . . 29
4.2 Theoretical background . . . 29

4.2.1 Locomotion 29
4.2.2 Kinematics 30
4.2.2.1 Motion control 30
4.2.3 Perception 31
4.2.4 Navigation: localization-based versus behavior-based solutions . . 32
4.3 Equipment description 33
4.3.1 Hardware 33
4.3.2 Software 34
4.4 Design and implementation 35
4.4.1 Approach 35
4.4.2 Algorithm 37
4.5 Module test 40
4.6 Summary 41

5 Data representation **43**
5.1 Introduction 43
5.2 Thermal imaging animation 43
5.3 Robot map simulation 46
5.4 Module test 47
5.5 Summary 49

6 System communication **51**
6.1 Introduction 51
6.2 Detection - Tracking modules 51
6.3 Detection - Representation modules 52
6.4 Tracking - Representation modules 53
6.5 Summary 53

7 System test **55**
7.1 Introduction 55
7.2 Static hot object test 55
7.3 Moving hot object test 57
7.4 Summary 59

8 Conclusion **61**

Appendices **63**

A Source code **65**
A.1 Hot object detection script 65
A.2 Hot object tracking script 73
A.3 Data representation scripts 77
A.4 System communication scripts 81

B Online resources **87**

B.1 Libraries 87

B.2 Videos 87

Bibliography **89**

List of Figures

1.1 Image captured by a thermal infrared camera on a firefighting mission [20]. . 2

2.1 PureThermal Mini - FLIR Lepton Smart I/O Module with Lepton 3.5 infrared camera [8]. 6
2.2 Small Mobile Robot platform developed by the Automation and Control department of DTU. 7
2.3 Overview of the modules that make up the proposed solution. 9

3.1 The electromagnetic spectrum with sub-divided infrared spectrum. 12
3.2 Atmospheric transmittance in part of the infrared region [23]. 13
3.3 Intensity of blackbody radiation versus wavelength at different temperatures [14]. 14
3.4 Reflected temperature can affect the temperature a thermal camera measures [21]. 15
3.5 Examples of thermography applications in different fields. 18
3.6 Flowchart showing the algorithm used to solve the hot object detection module. 23

4.1 The global reference frame and the robot local reference frame [19]. 30
4.2 Typical situation for feedback control of a mobile robot [19]. 31
4.3 Comparison of the information flow in localization-based (left panel) and in behavior-based (right panel) approach. For the latter, any number of behaviors may be involved, and the figure only shows an example with four behaviors. 33
4.4 Scheme showing the behavior-based proposal used to solve the hot object tracking module. 37
4.5 Scheme showing the acceptable limits in the frames captured by the IR-camera for hot object tracking. The robot should move so that the detected hot object remains on the central panel. 39

5.1 Scheme of the array resulting from concatenating every frame captured by the infrared camera during the experiment. 44
5.2 Representation of an image captured by the Lepton 3.5 infrared camera when a hot object is detected. 48
5.3 Representation of an image captured by the Lepton 3.5 infrared camera when no hot object is detected. 49

5.4 Representation of the direction (left panel) and trajectory (right panel) map of the Small Mobile Robot. 50

7.1 Representation of the direction (left panel) and trajectory (right panel) map of the SMR when tracking two static hot objects. 56
7.2 Representation of the direction (left panel) and trajectory (right panel) map of the SMR when tracking a moving hot object. 58

List of Tables

3.1 Specifications of the *Lepton 3.5* infrared camera from FLIR brand [6]. 20

Nomenclature

AGC Automatic Gain Control

API Application Programming Interface

ATRV-Jr All-Terrain Robotic Vehicle Junior

AUT Automation and Control department of DTU

CCI Command and Control Interface

CSV Comma-Separated Values

DLL Dynamic Link Library

DTU Danmarks Tekniske Universitet (Technical University of Denmark)

FFC Flat-Field Correction

FIR Far infrared

FLIR Forward-Looking Infrared

FOV Field Of View

I^2C Inter-Integrated Circuit

IR Infrared

JPEG Joint Photographic Experts Group

LTS Long Term Support

LWIR Longwave Infrared

MPEG Moving Picture Experts Group

MWIR Midwave Infrared

NIR Near Infrared

NUC Non-Uniformity Correction

OS Operation System

RGB Red Green Blue

SMR-CL Small Mobile Robot Control Language

SMR Small Mobile Robot

SWIR Shortwave Infrared

TIR Thermal Infrared

TWI Two-Wire Interface

UAV Unmanned Aerial Vehicle

UBC USB Video Class

USB Universal Serial Bus

VOx Vanadium Oxide

CHAPTER 1

Introduction

Visual detection of hot objects is an engineering area that has been and currently is subject to extensive research and development. Thermal infrared cameras have historically been of interest mainly for military purposes. However, as price and size decrease while image resolution and quality increase, the use of this type of camera in other applications has been rapidly growing.

Compared to cameras operating in the visual spectrum, infrared cameras are advantageous due to their high performance in all weather conditions, ability to work even in complete darkness, low maintenance and protection of privacy, among others. This great potential makes them suitable for a wide variety of important fields such as firefighting, automotive safety (obstacle detection), industry (plant inspection), security (intrusion detection), or search and rescue (localisation of missing people) [2].

In many cases, the detection of hot objects is not enough and their tracking is also necessary. That is why infrared cameras usually work together with mobile robots to provide the system with movement and thus be able to detect and track both static and moving hot objects.

Nowadays, mobile robotics is one of the fastest expanding fields of scientific research. They can be distinguished from other robots by their ability to move autonomously, with enough intelligence to react and make decisions based on the perception they receive from the environment [17].

Due to their abilities, mobile robots can substitute humans in many fields and be a great option to track objects safely, precisely, and autonomously in different places.

This thesis addresses the current necessity of tracking hot objects for several applications and analyses the advantages and limitations of using thermal infrared cameras in conjunction with mobile robots for this purpose.

1.1 Objectives

The main goal of this project is to develop and implement a system consisting of an infrared camera and an autonomous mobile robot coordinated with each other to detect and track hot objects. Specifically, the focus is on firefighting within different environments (Figure 1.1).

To do this, the camera should be able to:

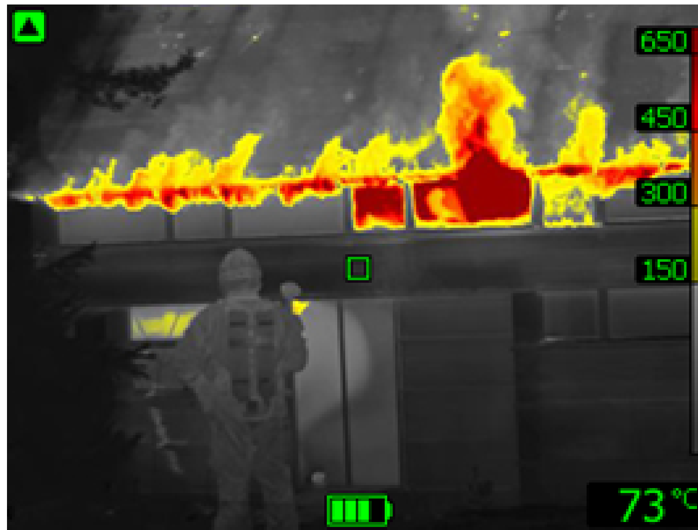


Figure 1.1: Image captured by a thermal infrared camera on a firefighting mission [20].

- Continuously capture images with high frame rate while the robot is running.
- Identify if there is a hot enough object in the room according to a pre-established temperature threshold.
- Find the hottest spot representing the fire and get its temperature and location in the frame.
- Send the hot object data of interest to the robot.
- Produce an animation from the captured images to show the results of the experiment.

On the other hand, the robot should be able to:

- Continuously read the hot object data from the camera.
- Check for any fire in the room.
- Track and reach the fire location.
- Extinguish the fire.
- Return to the starting position.
- Produce a simulation of the robot map to show the results of the experiment.

1.2 Thesis outline

In this section, an overview of the chapters found in this project is presented:

- **Chapter 1** gives a brief introduction to the fields of infrared cameras and mobile robots that contextualize this project and states its main objectives.
- **Chapter 2** analyses a solution that satisfactorily meets the objectives, setting the necessary frame for its subsequent design and implementation.
- **Chapter 3** addresses the module responsible for the infrared camera tasks in order to detect hot objects in the room.
- **Chapter 4** handles the module responsible for the robot tasks in order to track the detected hot objects.
- **Chapter 5** discusses the treatment and representation of data collected by both devices during the experiments.
- **Chapter 6** deals with the communications between all modules for the coordinated operation of the overall solution.
- **Chapter 7** analyses the results of the system test covering two possible scenarios corresponding to a static and a dynamic hot object.
- **Chapter 8** contains the main conclusions of the project regarding the achievement of its objectives as well as other final considerations of interest.

CHAPTER 2
Analysis

2.1 Introduction

In order to achieve the main objectives mentioned above, the realization of this project can be approached in several ways. That is why a prior study of possible solutions regarding each subsystem is required to allow choosing the one that best satisfies the desired operation.

These decisions principally refer to the hardware and software to be used for the proper design and implementation of the overall solution and depends largely on the corresponding availability at the university.

The present analysis further divides the project into thematic packages that will be described and evaluated on its own later on.

2.2 Camera

Unlike visible light images, thermography is a type of imaging that is accomplished with an infrared (IR) camera calibrated to display temperature values across an object or scene [22]. Therefore, IR-camera allows one to make non-contact measurements of an object's temperature and, consequently, is the type of camera used for the main perception part of the system.

Due to the unstoppable growth in the use of thermography in industry, a great rivalry has arisen to dominate the IR-camera market. There are numerous companies focused on the development and commercialization of these devices, offering, in turn, a wide variety of ranges and models to cover the different applications demanded by the industry.

FLIR Systems is the world's largest commercial company specializing in this field and furthermore the Automation and Control department of DTU (AUT) has some of its products available for students. The brand chosen for the infrared camera is therefore FLIR.

With regard to the specific model, the *PureThermal Mini - FLIR Lepton Smart I/O Module* has been chosen for the camera board and the *Lepton 3.5 Uncooled VOx Microbolometer* for the camera itself (Figure 2.1). The main reason for this selection

is that it constitutes an affordable and easy to hack thermal USB camera with a large scene dynamic range of temperature and reduced dimensions and weight.

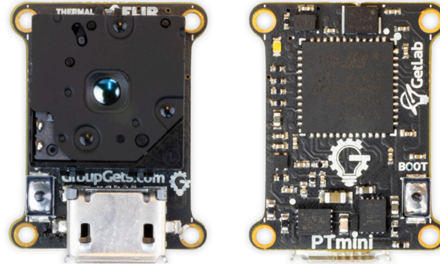


Figure 2.1: PureThermal Mini - FLIR Lepton Smart I/O Module with Lepton 3.5 infrared camera [8].

A more detailed description of the features of the chosen products will be included in section 3.3, within the chapter about hot object detection.

2.3 Robot

Due to the high prices for commercial robot platforms at present, the Automation and Control department of DTU decided to build their own platform. Custom made hardware from the shelf components is the philosophy behind the Small Mobile Robot (SMR) design. Although the department offers other bigger and better equipped options such as the Terrain Hopper, iRobot ATRV-Jr or DTU MaizeRunner, the SMR has been chosen since it covers all this project needs with a very compact design.

Regarding that design, an important parameter to be fulfilled is that the robot should be so small that it can easily be handled by one person without any safety problems and on the other hand, it should be big enough to carry equipment such as cameras. As seen in Figure 2.2, the 28 by 28 cm size of the SMR together with its lightweight chassis guarantee this parameter.

For more information about the chosen equipment, section 4.3 in the hot object tracking chapter can be consulted.

2.4 Environment

Once the camera and the robot have been selected, the type of environment for the subsequent experiments needs to be defined as well. The system should be versatile enough to meet the main objective of tracking hot objects regardless the site. In fact, as the focus of the project is on firefighting, it is reasonable to assume that the environmental conditions may change over time due to damage and other factors caused by the

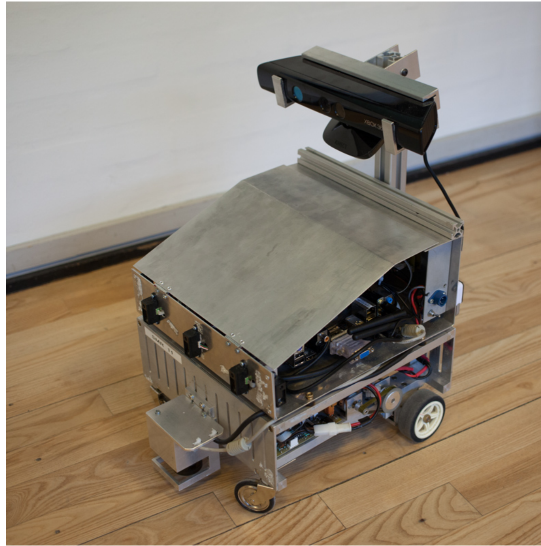


Figure 2.2: Small Mobile Robot platform developed by the Automation and Control department of DTU.

fire itself or its surroundings. Therefore, the environment is considered to be unknown, meaning no map representation is available in advance.

On the other hand, the SMR uses a local network associated with the DTU building 326 for its communications and, consequently, the experiments are only carried out indoors while outdoor scenarios are beyond the scope of this project.

2.5 Software

This section addresses the references to suitable software for the hardware mentioned above. In most cases, the software to be used is the only option corresponding to the chosen hardware and, in other cases, the best available option has been selected in terms of good performance as well as usability.

Regarding the IR-camera, the following main software applies:

- **Libuvc** - A cross-platform library that supports enumeration, control and streaming for USB Video Class (UVC) devices.
- **Python** - The programming language used for the image acquisition and creation of the corresponding simulation.

On the side of the robot, the software list to use is as follows:

- **Linux** - The main Operating System.

- **Mobotware** - A plug-in based platform developed at AUT for the control software of mobile robots.
- **SMR-CL** - Small Mobile Robot Control Language is a simple interpreted language developed at AUT intended to control mobile robots.
- **C++** - The language to program the plugin that communicates the SMR with the IR-camera.
- **Matlab** - The programming platform responsible for simulating the data provided by the SMR.

2.6 Modules

All things considered, it is possible to identify different modules that need to work in coordination to meet every agreed target at the beginning of the project. By doing so, the system should be able to detect hot objects using the IR-camera, communicate with the SMR to track them and finally represent the collected data. Therefore, the proposed solution is split into four thematic packages or modules that are outlined in Figure 2.3 and briefly described below:

- **Hot object detection** - The first module is responsible for capturing images from the IR-camera, identifying if there is a hot enough object in the room and getting its temperature and location in the frame.
- **Hot object tracking** - This module mainly addresses the sequence of movements to be carried out by the SMR to check for any fire in the room, track and reach its location, extinguish it and return.
- **Data representation** - This module covers the animation of the images captured by the camera during the experiment as well as the simulation of the robot trajectory and direction map.
- **System communication** - It guarantees that the rest of modules communicate with each other for the correct functioning of the overall system, that is, it handles the continuous exchange of hot object data between the IR-camera and the robot and allows the scripts in charge of representation to read the data collected by both devices.

2.7 Summary

In this chapter, an analysis of a solution which satisfactorily meets the objectives or requirements stated in chapter 1 has been conducted, setting the necessary frame for its subsequent design and implementation.

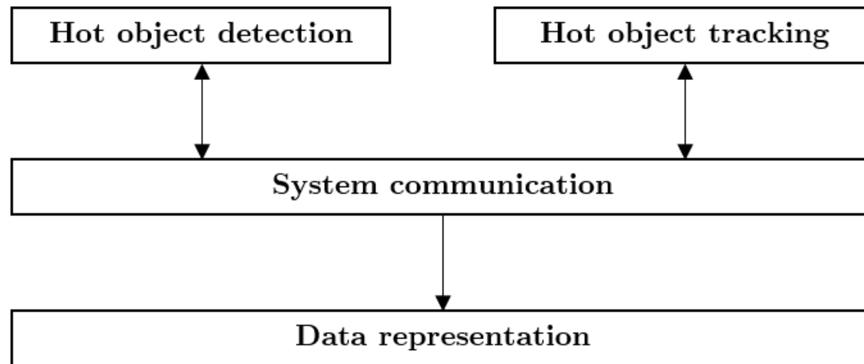


Figure 2.3: Overview of the modules that make up the proposed solution.

Firstly, the best available hardware at DTU for the project purpose has been selected. The *PureThermal Mini - FLIR Lepton Smart I/O Module* along with the *Lepton 3.5* make up the IR-camera for the main perception part of the system while the SMR developed by AUT constitutes the mobile robot. In addition, the experiment scenario has been defined in such a way that the environment is indoor and unknown.

Next, the required software for each sub-task associated with both the camera and the robot has been referenced and briefly commented.

Lastly, the solution under analysis has been divided into four modules to encompass the primary missions of the project: system communication, hot object detection, hot object tracking and data representation.

In the following chapters, the approach and development of each of the proposed thematic packages will be presented at length.

CHAPTER 3

Hot object detection

3.1 Introduction

The present chapter addresses the first module of the proposed solution, which is responsible for the infrared camera tasks in order to detect hot objects in the room. Specifically, the objectives related to this module are capturing images from the IR-camera, identifying if there is a hot enough object around and getting its temperature and location in the frame.

To meet the stated objectives, it is necessary to have some basic theoretical notions of the subject matter. Therefore, the first section sets a theoretical background that introduces the reader to the infrared radiation tenets, thermal imaging along with its main characteristics and some of the most common applications.

Once the foundations are known, it is possible to describe in more detail the choices made in the analysis regarding the equipment to be used. The equipment description section covers both hardware and software tools related to the camera, determining the advantages and limitations of what is available to implement the solution.

Next, an approach to the solution is conducted in a way that the best method to achieve the objectives is identified and presented. The following subsection constitute a gradual process that begins with a conceptual diagram and ends with the detailed implementation of each of the parts that make up the solution for this module. Every step followed by the algorithm that governs the camera is duly justified and described in such subsection.

Finally, a module test is included to verify that the proposal fulfills the initial intention as expected.

3.2 Theoretical background

Thermal infrared imaging forms the basis of this section. All subsections included here constitute a comprehensive introduction to the field and aim to provide a solid understanding of how infrared cameras work. This theoretical background is mainly focused on the knowledge that is useful and can be applied in some way to the subsequent implementation of the solution, rather than just a concatenation of conceptual definitions.

3.2.1 Infrared radiation

Infrared radiation is a part of the electromagnetic spectrum, see Figure 3.1, and its name originates from the Latin word *infra*, which means *below*. That is, the infrared band lies below the visual red light band, since it has longer wavelength.

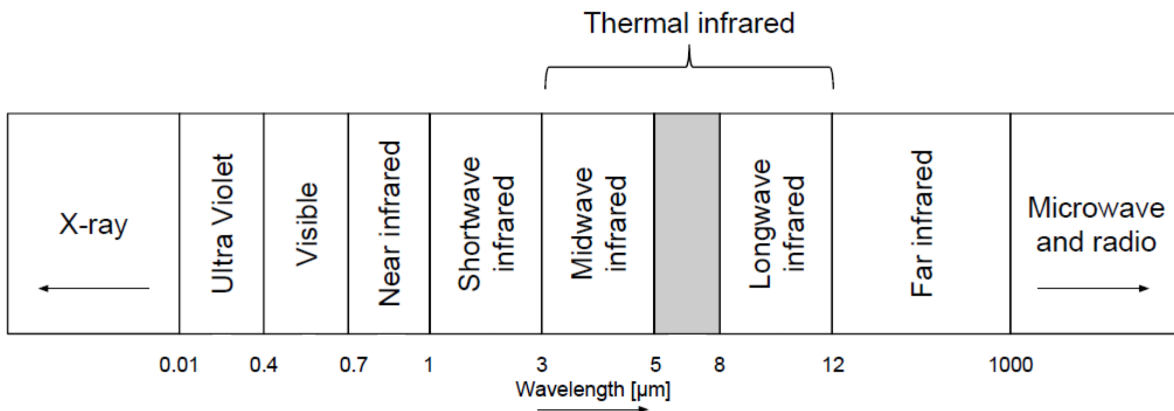


Figure 3.1: The electromagnetic spectrum with sub-divided infrared spectrum.

The infrared spectrum can be divided into several spectral regions based on their properties. There exist different sub-division schemes in different scientific fields, but the most common is as follows: near infrared (NIR, wavelengths 0.7–1 μm), shortwave infrared (SWIR, 1–3 μm), midwave infrared (MWIR, 3–5 μm), longwave infrared (LWIR, 8–12 μm), and far infrared (FIR, 12–1000 μm). LWIR, and sometimes MWIR, is typically referred to as thermal infrared (TIR). TIR cameras are sensitive to *emited* radiation in everyday temperatures and should not be confused with NIR and SWIR cameras that, in contrast, mostly measure *reflected* radiation. These non-thermal cameras are dependent on illumination and behave in general in a similar way as visual cameras [2].

Due to scattering by particles and absorption by gases, the atmosphere attenuates radiation, making the measured apparent temperature decrease with increased distance. CO_2 and H_2O are responsible for most of the absorption of infrared radiation [12]. Figure 3.2 shows the percentage of transmitted radiation depending on the wavelength and indicates the molecule that is responsible for the transmission gaps. As can be seen in the figure, there are two main sections in which the atmosphere transmits a major part of the radiation. These are called the atmospheric windows and can be found between 3–5 μm (mid-wave window) and 8–12 μm (long-wave window) [16]. These windows correspond to the MWIR and LWIR bands mentioned above. Conversely, due to the large transmission gap between 5 and 8 μm, there is no reason for cameras to be sensitive in this band. The same goes for radiation above 14 μm.

Radiation that is not absorbed or scattered in the atmosphere can reach and interact with matter. There are three forms of interaction that can take place when energy strikes or is incident upon an object. These are absorption (whose coefficient is α), transmission (τ), and reflection (ρ). The total incident energy will interact with the surface in one or

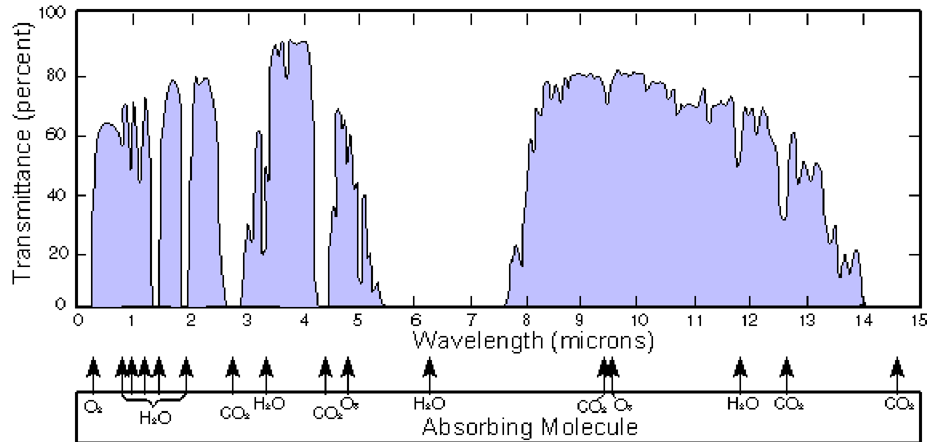


Figure 3.2: Atmospheric transmittance in part of the infrared region [23].

more of these three ways. The proportions of each will depend on the wavelength of the energy as well as the material and conditions.

Conservation of energy requires that $\rho+\tau+\alpha = 1$, where $\rho, \tau, \alpha \in [0, 1]$. On the basis of fundamental physics, every object at any given absolute temperature above 0 K emits thermal radiation, which is defined as energy transfer in the form of electromagnetic waves [4].

A blackbody is an ideal body that allows all incident radiation to pass into it and that absorbs internally all the incident radiation, resulting in $\alpha = 1$. By measuring the blackbody radiation curves at different temperatures (Figure 3.3), it is possible to observe that the intensity peak shifts to shorter wavelengths as the temperature increases while the intensity increases with the temperature. For extremely hot objects, the radiation extends into the visible spectrum, e.g., as seen for a red-hot iron [7].

Most materials studied for practical applications are assumed to be so called grey bodies, which have a constant scale factor of the radiation between 0 and 1. This factor is called emissivity (ϵ) and is defined as the amount of radiation actually emitted by a body compared with that of a blackbody under identical conditions [5]. Further, Kirchhoff's law states that $\alpha = \epsilon$, i.e., $\alpha = 1$ for a black body. Since emissivity depends on the material, it is an important property when measuring temperatures with a thermal camera.

3.2.2 Thermal imaging

Thermal imaging, also often briefly called thermography, is the process of converting infrared radiation (heat) into visible images that depict the spatial distribution of temperature differences in a scene viewed by non-contact thermal imaging devices. In most cases, this measurement technique is able to quantitatively measure surface tempera-

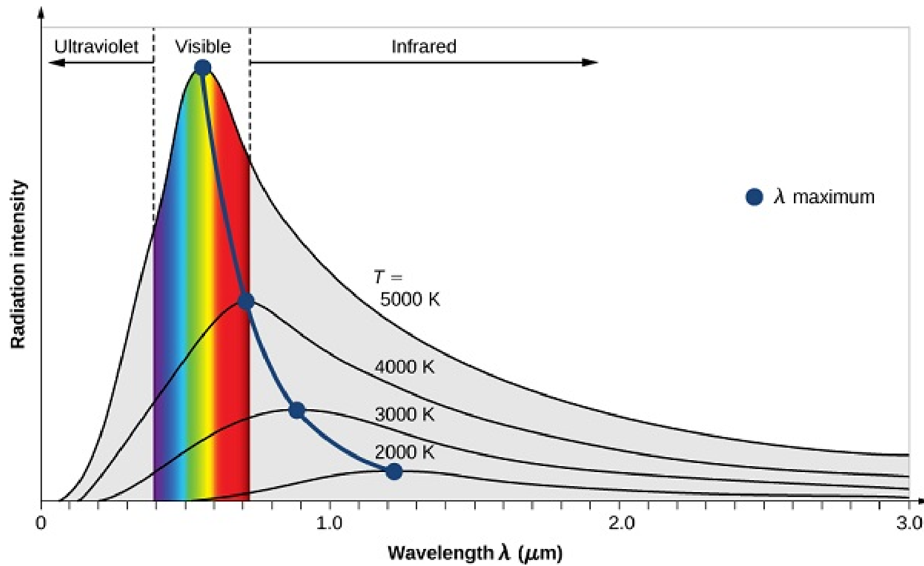


Figure 3.3: Intensity of blackbody radiation versus wavelength at different temperatures [14].

tures of objects. For better understanding by humans of captured images, pseudo colors are usually used to map pixel intensity values in order to visualize details more clearly.

Colors in the visible range express reflection of light. However, colors in an infrared image express both reflection and emission. Emission of heat comes from the material itself while reflection comes from its surroundings. Notice that these properties are complementary, i.e., a good emitter is a poor reflector and vice versa. This is where emissivity comes into play, which represents how efficiently an object radiates heat as explained above.

When viewing a highly polished metal object with a low emissivity, that surface will act like a mirror. Instead of measuring the temperature of the object itself, the camera will also detect the reflected temperature. Reflected temperature (also known as background temperature or $T_{reflected}$) is any thermal radiation originating from other objects that reflect off the target you are measuring. This is explained through Figure 3.4.

To properly obtain an accurate surface temperature reading with thermal imaging, this $T_{reflected}$ value (along with the emissivity) must be quantified and included in the camera's object parameters. This is used (specially for lower emissivity objects, where reflected temperature has a high influence) so that the software can compensate for, and ignore, the effects of the radiation which does not correspond to the actual surface temperature of the target.

By contrast, measuring an exact temperature is not necessary when the application simply aims to locate thermal patterns such as missing insulation or air leakage in a building, which is called qualitative thermography. In those situations, leaving the

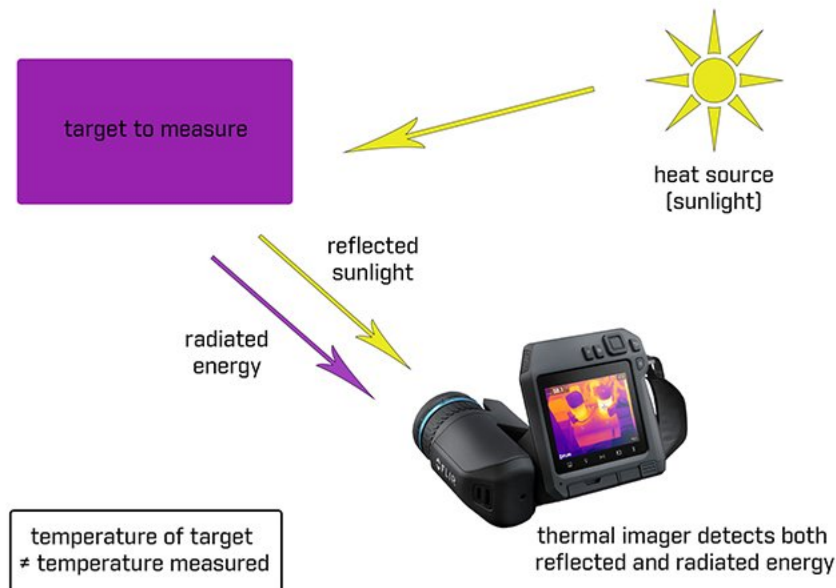


Figure 3.4: Reflected temperature can affect the temperature a thermal camera measures [21].

values at their default settings (typically 0.95 for emissivity, and 20 °C for reflected temperature) will suffice.

There are currently two types of thermal imaging sensors on the market, cooled and uncooled. Cooled cameras are the most sensitive to tiny differences in temperature, offer very high image quality, and are compatible with longer-range lenses. Images are typically stored as 16 bits per pixel to allow a large dynamic range. On the other side, uncooled cameras usually have bolometer-type detectors and operate in LWIR. They yield noisier images at a lower frame rate, but are smaller, silent, and less expensive, being particularly well-suited for mobile applications.

Some uncooled cameras are calibrated in order to measure temperatures and provide access to the raw 16-bit intensity values, so called radiometric cameras, while others convert the images to 8 bits and compress them, e.g., using MPEG. In the latter case (non-radiometric), the dynamic range is adaptively changed to provide an image that looks good to the eye, but the temperature information is lost. For automatic analysis, such as target detection, classification, and tracking, it is suitable to use the original signal, i.e., the raw 16-bit intensity values from a radiometric camera [2].

In many different applications, the most commonly used detector is an uncooled bolometer camera for LWIR with germanium lens. This material is opaque and reflective in the visual spectrum while transparent in the thermal spectrum and is therefore employed for the lens instead of glass, which has the opposite properties.

3.2.3 Features, advantages and limitations

Thermography is an excellent example of an acquisition and analysis technique that can be used in a wide variety of fields. This technique is so useful mainly due to the following **features** [3]:

- *It is non-contact (uses remote sensing)*
  - Keeps the user out of danger.
  - Does not intrude upon or affect the target.
- *It is two-dimensional*
  - Comparison between areas of the target is possible.
  - The image allows for excellent overview of the target.
  - Thermal patterns significantly enhance problem diagnosis.
- *It is real time (or close to real time)*
  - Enables efficient scanning of stationary targets.
  - High end cameras can capture fast moving targets.
  - High end cameras can capture rapidly changing thermal patterns.

Certainly, thermal detection technologies are advantageous in many ways, but they also present some drawbacks or limitations compared to cameras operating in the visual spectrum that should be taken into consideration.

Listed here are relevant **advantages** when using thermal imaging [11]:

- *High performance during darkness and/or difficult weather conditions.* A thermal camera is sensitive to emitted radiation, even from relatively cold objects, in contrast to a visual camera that measures reflected radiation and thus depends on illumination.
- *Low maintenance for lower total cost of ownership.* Unlike visible-light cameras, thermal cameras generate virtually no maintenance costs and, over time, are a more cost-effective solution.
- *Protection of privacy.* Infrared cameras can detect an individual without revealing their identity and therefore protect people's privacy. This also can be a disadvantage as listed further down.
- *Fewer false alarms.* This type of camera can produce sharp, accurate images, which means fewer false alarms. They can clearly differentiate an animal, for example, from an intruder.

Regarding the **limitations**, the following are worth mentioning [11]:

- *High initial investment cost.* Thermal cameras' optics and detection systems are not as widely used as those of visible-light cameras, and they are built using complicated components, which increases their purchase prices.
- *Inability to 'see' through certain materials like water and glass.* Unlike visible light, infrared radiation cannot go through water or glass, since it is reflected in those materials.
- *More training is necessary.* In comparison to visual cameras, thermal detectors typically require more training for correct usage. To provide accurate measurements, the operator needs to be aware of the physical principles and phenomena commonly viewed in thermal imagery (emissivity, reflected temperature...).
- *Person identification is not possible.* If person identification is requested, it has to be combined with a visual camera.

3.2.4 Applications

The different characteristics analysed above define the type of applications in which it is appropriate to use infrared cameras. A few categories are identified and briefly described below along with Figure 3.5, which displays example images from each category [2] [18]:

a) **Military**. Thermal cameras are popularly used by the army and navy for border surveillance and law enforcement. They are also widely used in military aviation to identify, locate and target the enemy forces (gunfire, mines, snipers, etc.).

b) **Agriculture**. Monitoring of wild and domestic animals can be used, e.g., to detect inflammations, perform behaviour analysis, or to estimate population sizes

c) **Automotive safety**. Detection and tracking of pedestrians or other vehicles, using a small thermal camera mounted in the front of the car or train. It also provides detection of defects of many products for the automotive industry.

d) **Building inspection**. When looking to improve upon energy efficiency and lead the world forward in the fight against climate change, improving building structure to combat energy loss and resource wasting is greatly aided with the use of infrared cameras.

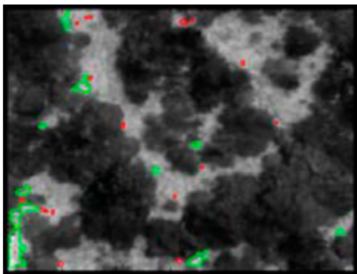
e) **Industry**. This is a broad area that includes applications such as detection of defects and deficiencies of multiple materials, positioning, non-destructive testing etc.

f) **Medicine**. Thermal cameras are being used to help detect cancer earlier, locate the source of arthritis, and even catch circulation issues before they become too problematic.

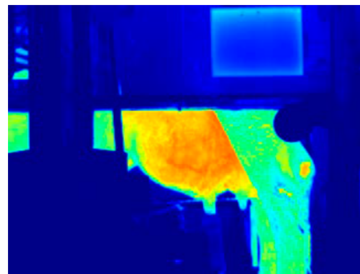
g) **Search and rescue**. Localisation of missing people regardless of daylight and weather conditions using cameras carried by UAVs, helicopters, or rescue robots.

h) **Security**. Thermography is an ideal tool for finding and tracing of suspects, identification of persons and vehicles, and search for crime victims, among others.

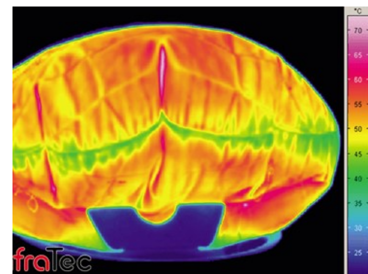
i) **Firefighting**. Firefighters use thermal imaging cameras every day to see through smoke, locate and rescue victims, identify hot spots, navigate safely, and stay better oriented during response missions.



(a) **Military**.
Mine detection.



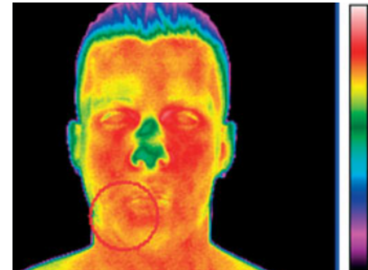
(b) **Agriculture**.
Mastitis detection.



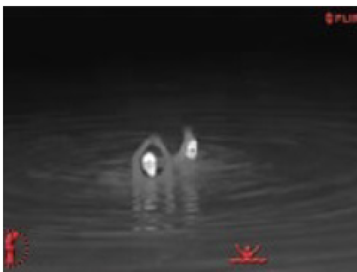
(c) **Automotive safety**.
Looking for defects in an airbag.

(d) **Building inspection**.
Detection of heat loss in a building [7].

(e) **Industry**.
Pump leak detection [12].

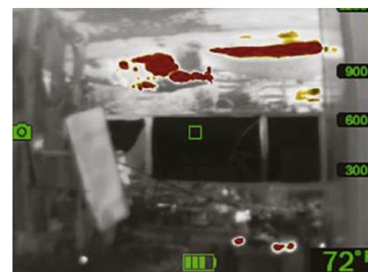


(f) **Medicine**.
Localisation of infections [12].



(g) **Search and rescue**.
Localisation of missing people.

(h) **Security**.
Intrusion detection.



(i) **Firefighting**.
Detection of hot spots [13].

Figure 3.5: Examples of thermography applications in different fields.

3.3 Equipment description

Once the basic tenets of thermography are understood, it is possible to take a closer look into the chosen equipment. This chapter describes both the camera and the corresponding software to analyse what is available for the implementation of the solution.

3.3.1 Hardware

As already mentioned, the sensor responsible for detecting hot objects consists of two parts: the *PureThermal Mini - FLIR Lepton Smart I/O Module* for the camera board and the *Lepton 3.5 Uncooled VOx Microbolometer* for the camera itself. This device needs to be of reduced dimensions and weight, since it is mounted on the Small Mobile Robot, so it is an uncooled-type camera. Furthermore, it provides radiometric images, meaning each pixel can represent the temperature measured at that point, which is also important when locating the hottest spots in the environment. Easy to integrate and operate, it is intended for mobile devices as well as any other application requiring very small footprint, very low power, and instant-on operation.

This micro thermal camera is assembled on the *Lepton Smart I/O Module*, which is pre-configured to operate as a plug-and-play UVC 1.0 USB thermal webcam that works with standard webcams and video apps on all major platforms (Windows, Linux, Mac, and Android). It supports open source reference firmware and viewer software, and it presents a compact form-factor ready to be embedded into production systems. Specifically, the dimensions of the board are 19.15 mm in height and 15.32 mm in width [9].

Regarding the *Lepton 3.5* camera, Table 3.1 contains some specifications that should be highlighted.

3.3.2 Software

The Lepton camera module supports a Command and Control Interface (CCI) hosted on a Two-Wire Interface (TWI) similar to I^2C (a standard serial communication protocol). The interface consists of a small number of registers through which a host issues commands to, and retrieves responses from the Lepton camera module. It can be operated in its default mode or configured into other modes through the CCI.

The chosen module supports the USB video class (UVC), and this makes it very easy to capture thermal imaging data from a host PC using standard tools and libraries and includes the ability to update firmware over USB, which lengthens the shelf life of the device. *Libuvc* is one of these standard tools and has been chosen for the implementation, since it is a way to easily access camera commands and avoid directly hacking into the firmware. It constitutes a comprehensive library and can be defined mainly through the following features [10]:

Table 3.1: Specifications of the *Lepton 3.5* infrared camera from FLIR brand [6].

Sensor technology:	Uncooled VOx microbolometer
Array format:	160 x 120 (19,200 active pixels)
Frame rate:	8.7 fps
Thermal sensitivity:	<50 mK (0.05 °C)
Temperature compensation:	Automatic (output image independent of camera temperature)
Scene dynamic range:	-10 to +400 °C
Horizontal FOV:	57 °
Lens type:	f/1.1
Output format:	User-selectable 14-bit, 8-bit (AGC applied), or 24-bit RGB (AGC and colorization applied)
Size (w x l x h):	10.50 x 12.70 x 7.14 mm
Weight:	0.9 grams
Power consumption:	150 mW typical, 650 mW during shutter event, 5mW standby
Operating temperature range:	-10 to +80 °C

- Application Programming Interface (API) for UVC device discovery and management (finding, inspecting and opening).
- Support for creating, managing and consuming video streams (device to host).
- Read/write access to standard device settings (functions for manipulating main settings and stream parameters)
- Tools for managing frame buffers and converting between various image formats.: RGB, YUV, JPEG, etc.
- Cross-platform user library.

As already mentioned in the analysis, a Python script is responsible for the camera tasks, since it is a very versatile and productive language, as well as easy to use and fast to develop. In particular, the version used is Python 3.6.9. Although Libuvc is a library based on C language, it is seamlessly adaptable to Python using the so called *ctypes* library. This is a foreign function library for Python that provides C compatible data types and allows calling functions in DLLs or shared libraries. Therefore, it can be used to wrap Libuvc in pure Python.

Furthermore, two other important Python libraries are needed for the correct operation of the source code: *Numpy*, used for working with arrays with faster execution time; and *Matplotlib*, used for creating static, animated, and interactive visualizations.

3.4 Design and implementation

This section addresses the design and implementation of the proposal that provides a solution to the problems for which this module is responsible. Firstly, the approach specifies the main characteristics and operating conditions of the camera applied to this case study and sets the best way for the programming code to handle the device accordingly. The second subsection goes into the details of the proposed solution and explains each step followed by the hot object detection algorithm that governs the camera along with some important considerations.

3.4.1 Approach

One of the first steps is to decide in which mode the camera should operate. As stated in Table 3.1, the output format is user-selectable and can be of 14-bit, 8-bit or 24-bit RGB. The former provides raw data, which implies that the AGC mode is disabled and the TLinear mode is enabled. AGC stands for Automatic Gain Control and is a process whereby the raw image is converted into a contrast-enhanced image suitable for display, that is, a 14-bit to 8-bit conversion, with the corresponding loss of thermal information. Enabling the TLinear mode changes the pixel output from representing incident radiation in 14-bit digital counts to representing scene temperature values in centikelvin, which is possible because the Lepton 3.5 is a radiometric camera and is therefore calibrated for that purpose. As this module's core priority is to get the thermal data of every pixel rather than obtain images that look good to the eye, the raw 14-bit option is selected, which is in fact the default output format.

Nevertheless, the TWI interface is a communication protocol that only allows 16-bit transfers. Consequently, each pixel value is represented by 16 bits, covering all 14 bits of thermal information from the Lepton camera, and the default data type is `uint16` (unsigned integer with a data width of 16-bits). Since the Lepton 3.5 supports a 160 x 120 format, the resulting image is represented by an `uint16` array of 19200 pixels. As discussed in the next subsection, sometimes the proposed algorithm converts temperature values from centikelvin to degrees Celsius for easier handling and reading.

Regarding the internal parameters of the camera, both the emissivity and the reflected temperature should be adjusted to get a highly accurate reading of the temperature of the target object, which is known from the theoretical section. For this project application, however, only thermal patterns need to be read, since the objective is to identify the hottest areas of the environment, instead of getting their exact temperatures. Apart from that, two factors make it impossible to quantify the emissivity and the reflected temperature with sufficient accuracy: there is no specific target object, since the environment is unknown; and the robot is in motion, so the camera captures scenes with different conditions each time. As a result, these values are left at their default settings.

Lepton is factory calibrated to produce a highly uniform output image, although drift effects over long periods of time degrade uniformity, resulting in imagery which appears

grainy and/or blotchy. For scenarios in which there is ample scene movement, Lepton is capable of automatically compensating for drift effects using an internal algorithm called scene-based non-uniformity correction (scene-based NUC). On the other hand, for applications in which the scene is essentially stationary, it is recommended to habitually perform a flat-field correction (FFC) whereby the NUC terms are recalibrated to produce the most optimal image quality. Considering that this project clearly deals with dynamic scenes, the default automatic mode for the FFC is selected and no additional manual FFC is required. In this automatic mode, Lepton performs FFC at start-up, every 3 minutes and in case the camera temperature has changed more than 1.5 °C, which will suffice. It should be noted that the entire FFC process takes less than a second, so it is not significant enough to affect the correct performance of the camera-robot system.

At this point, it has been argued that the default settings of the Lepton 3.5 camera are suitable for the operating conditions of this application and, by extension, there is no need to modify any video mode, output format, data type, internal parameter or image correction. By not having this need, there is no reason to use the specific API provided by Lepton to access the internal CCI commands. Instead, this implementation remains on a more superficial layer in this regard, and the already mentioned Libuvc library is employed, since it is a generic API that enables fine-grained control over any UVC device.

3.4.2 Algorithm

Once the camera is properly configured and ready to be programmed, it is time to develop the algorithm in charge of this hot object detection module. As mentioned above, the implementation of such algorithm is carried out by means of a Python script.

As a reminder, this module aims to fulfill three main objectives: to capture images with a short time span from the IR-camera, identify if there is a hot enough object around and get its temperature and location in the frame. To do this, the algorithm represented in Figure 3.6 has been proposed, which outlines the set of instructions to be followed.

The following lines describe the whole process in a progressive way, making continuous reference to Appendix A.1, which contains the source code of the `CAM_SCRIPT.py` file responsible for the camera's mission. It should be noted that this file as well as the diagram from Figure 3.6 also includes some tasks that, although they interact with this module, correspond to the data representation and system communication modules (written in italics on the diagram). That said, here the focus is only on the detection of hot objects.

As can be seen in the flowchart, the Python script begins when the SMR starts running. It is not known how exactly it is initialized, since the implementation of this task will be addressed in the system communication module (Section 6.2), but it is assumed that in some manner the camera is able to determine that the robot has started

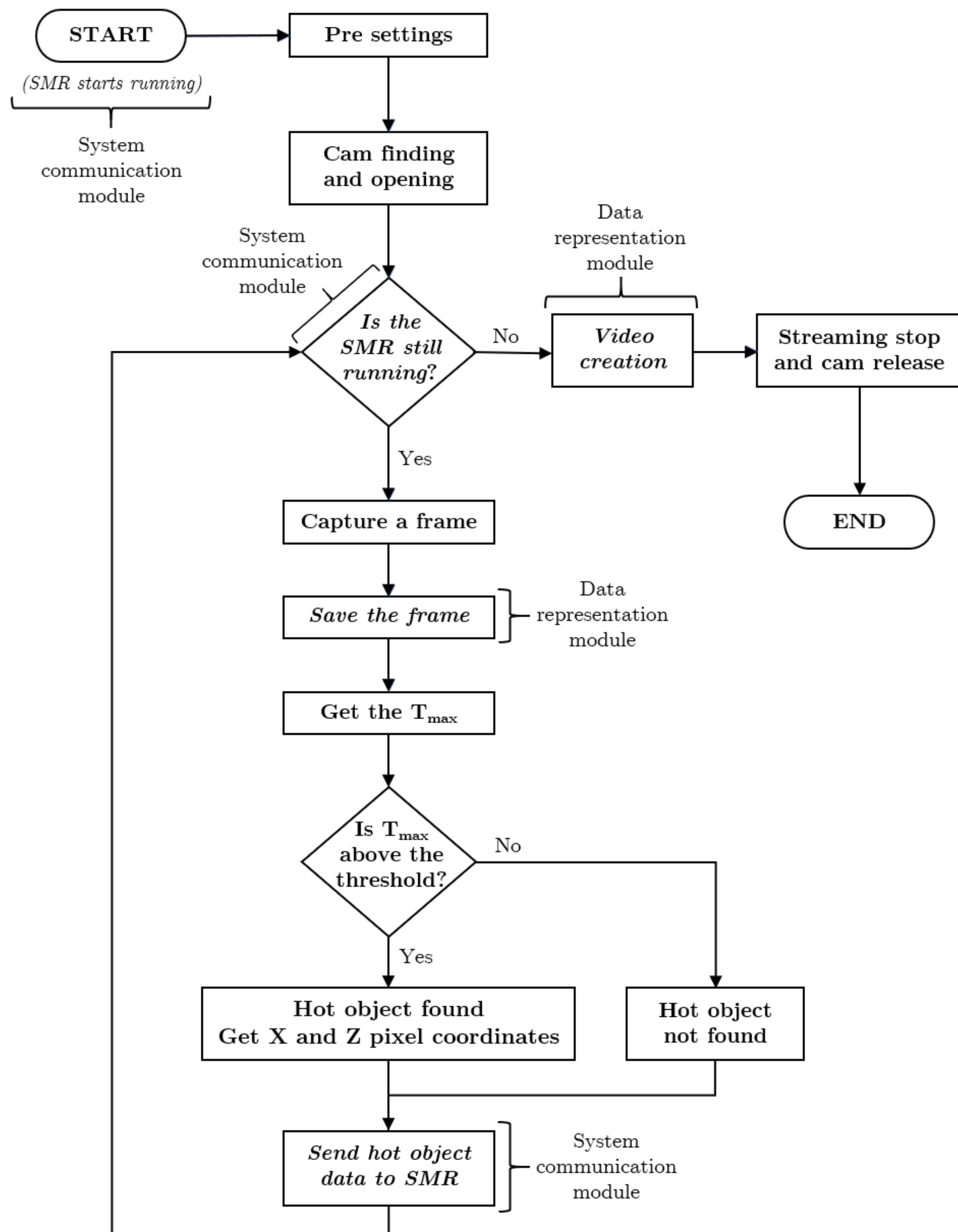


Figure 3.6: Flowchart showing the algorithm used to solve the hot object detection module.

to work. Something similar happens with the rest of the tasks that depend on other modules. In this regard, a module is understood as a delimited box that has its own characteristics and objectives, but that is coordinated in a certain way with the rest of

modules, receiving inputs and also providing outputs.

The preliminary settings are the first stage and fundamentally consist of package and library imports, definition of functions and initialization of parameters. The most important libraries have already been discussed in the software description, while the functions will be addressed as they appear in the flowchart. Regarding the parameters, the temperature threshold (`hot_threshold_celsius` in the code from Appendix A.1) should be underlined due to its important influence on the algorithm. Thanks to this threshold, the program is able to know if an object is considered hot (corresponding pixel value above it) or not (pixel value below it). This differentiation is essential because the particular goal of the project is not to track any hot object, but to track an object only if it is hot enough to represent a fire. Although it is assumed that a fire is at a very high temperature, the camera reading decreases with the distance, so initially a value of 50°C is established as the threshold, which will be validated in the system test chapter.

Before Lepton starts capturing images, the device should be found and opened for use. Given that such operations are standardised for any UVC device, Libuvc fully covers the camera access process, which includes basic functions (`uvc_init`, `uvc_find_device` and `uvc_open`) along with functions that allows for the camera streaming (`uvc_get_frame_formats_by_guid`, `uvc_get_stream_ctrl_format_size` and `uvc_start_streaming`). For the latter, it is also checked that the data type and the height and width of the frame are correct. If any of the functions fail, that is, the expected result is not obtained, the program quits and a message specifying the type of error is printed. As a brief comment, this is the moment when the camera automatically performs the aforementioned FFC (at start-up).

Some of the UVC functions used cannot be found directly in the Libuvc library, but they are gathered in an open source file called `uvctypes.py`, whose hyperlink is attached in Appendix B.1. It is a sort of intermediate file, since it imports the aforementioned ctypes library to adapt Libuvc to Python language and adds several useful parameters, functions and object classes.

Initially, the program tries to capture the first frame to make sure that, apart from having accessed the camera, it also works properly. The capturing process is done through a callback function that continuously stores streaming data in a buffer or queue. Fortunately, this function has a size parameter to control the number of items allowed in the queue, which is set to 2 with the aim of improving resource efficiency. Accordingly, the program simply takes an item from the queue (`q.get`) every time a frame needs to be captured. As already mentioned, the obtained item is a Numpy array with 19200 `uint16` data entries representing the approximate temperature values associated with each pixel.

Once the first capture is successful, the camera is ready to enter the capturing loop, which takes place as long as the robot is running. Here again, the implementation of the robot status reading is unknown for the time being.

The first step within the loop is to capture a frame following the procedure explained

above. Then it is saved along with the previously captured frames in a way that is explained in the corresponding data representation module (Section 5.2).

To be able to compare with the preset temperature threshold (50°C), the highest value of the array is taken using the `max` function and converted from centikelvin to degrees Celsius. If this value (T_{max} in the diagram) is above the threshold, fire has been detected and its location in the frame should be obtained thanks to the `unravel_index` function. Conversely, if T_{max} is below, there is no need to know which pixel this temperature corresponds to, since it is an object that is not considered hot enough to represent a potential fire.

Last, this hot object data is sent to the SMR by means of a function, whose implementation is again discussed in the system communication module. Such data include the following main parameters:

- `found` is set to 0 (if there is no fire), 1 (if a fire has been detected) or 3 (if the capturing loop has ended).
- `x_coordinate` is the pixel index on the horizontal axis.
- `z_coordinate` is the pixel index on the vertical axis.

The process described is repeated until the SMR completes its tasks, which triggers the video creation from the frames saved so far.

It is worth noting that the video could be made during the capturing loop or even displayed at the same time. Nevertheless, it should not be forgotten that a core goal of this module is to capture images as fast as possible so that the robot has access to hot object data almost in real time and that its movements are fluid and without delay (specially when tracking moving objects). Given the high computation time involved in the processing and visualization of images, this part needs to be placed after the loop, i.e., once the experiment has finished. Thus, the data collected by the camera can be represented and analysed a posteriori without affecting the algorithm performance during the experiment.

Finally, the use of Libuvc is again required, in this case for the end of the streaming and device release thanks to the following standard functions: `uvc_stop_streaming`, `uvc_unref_device` and `uvc_exit`.

3.5 Module test

The IR-camera functions have been tested by means of several experiments covering all possible situations considered by the algorithm. A small adjustment has been applied to the algorithm to avoid the tasks that will be implemented later in the system communication and data representation modules. Regarding the former, the hot object data is not sent in each iteration and the algorithm starts and ends manually rather than based

on the robot status. For the latter, the frames are simply not saved in each iteration and the video creation is skipped. These modifications do not affect the camera operation and allows for testing the detection module independently.

In this module test, the function called `append_row_to_HOC` has been used for data logging over time, whose implementation can be found in the functions definition part of the `CAM_SCRIPT.py` file attached in Appendix A.1. Although it is not used in the final solution, thanks to this function, it is possible to keep track of the results obtained by the camera without representing them. It basically creates a CSV file and appends a new row in each iteration of the capturing loop containing the following data: date, time, pixel index on the horizontal and vertical axis of the hot object (`x_coordinate` and `z_coordinate`), and the corresponding temperature.

By using a thermal resistor, it has been possible to verify the camera's response to a potential fire, which correctly updates the coordinates in the frame based on the position of the resistor. With respect to the temperature, it is also updated in such a way that it increases or decreases depending on the voltage applied to the resistor. Equally, if the resistor is hidden with the hand or simply placed out of the camera range, the maximum recorded temperature drops suddenly and no fire is detected.

Furthermore, the time column in the CSV file should be highlighted, since it shows that around 9 frames per second are captured, which matches the frame rate specification of the camera indicated in Table 3.1. This involves that the algorithm makes the most of the camera in this regard and allows the maximum data update rate so that the robot movements are fluid and without delay, which is one of the fundamental goals of the solution.

Lastly, some longer tests have been conducted to check the camera stability, resulting in satisfactory operation regardless of the test duration.

3.6 Summary

The chapter presented here has covered all necessary stages for the successful detection of hot objects by the selected infrared camera.

The first chapter has given a comprehensive introduction to the field of infrared thermal imaging. It should be underlined that most infrared cameras operate in LWIR (8–12 μm) avoiding the so-called atmospheric transmission gaps, and that the sensor is usually uncooled due to its portability and low cost. In addition, the advantages over a visual camera have been analysed, for example the ability to work in darkness, but also the disadvantages, such as the higher initial cost. A wide variety of applications have also been mentioned and displayed, ranging from industry field, namely pump leak detection, to firefighting missions, such as the case of this project.

The equipment description has made it possible to understand the importance of having an infrared camera that is radiometric to be able to read approximate tempera-

tures. Emissivity and reflected temperature are camera parameters that can be adjusted to improve the measurement accuracy, but this is not necessary for detecting hot objects as this application focuses on thermal patterns rather than exact temperatures.

As analysed, the default camera settings properly respond to the operating conditions of this application and no specific library is needed. Accordingly, the Libuvc standard library can be used, which allows for handling UVC devices like the selected Lepton 3.5 camera.

The explained Python script is responsible for implementing the algorithm to detect hot objects, allowing to fluently capture images and distinguish if those objects correspond to fires according to a pre-established temperature threshold.

Finally, the test section proves that the implemented algorithm meets all its requirements specified above at this stage of development.

CHAPTER 4
Hot object tracking

4.1 Introduction

This chapter is the second core thematic unit and follows the same structure that the previous one. At this point in the development of the solution, the system is capable of detecting a fire but cannot reach it. The hot object tracking module provides a solution to this and has the Small Mobile Robot as its core element.

The content begins with a theoretical background on autonomous mobile robots and the description of the selected SMR, follows with the design and implementation of the proposed solution, and ends with some testing as a verification of the module.

4.2 Theoretical background

Mobile robots can move autonomously (in an industrial plant, laboratory, planetary surface, etc.), that is, without assistance from external human operators. A robot is autonomous when the robot itself has the ability to determine the actions to be taken to perform a task, using a perception system that helps it [17].

Autonomous mobile robots forms the basis of this section. The following subsections briefly synthesizes the most relevant theoretical considerations in connection with this field. Although it is a very broad field, the content presented here focuses only on the theory applicable to the specific case of this project, in which the ultimate goal is that the robot serves as a means to track fires detected by the infrared camera.

4.2.1 Locomotion

A mobile robot needs locomotion mechanisms that enable it to move unbounded throughout its environment. While there are a large variety of possible ways to move, virtually all industrial applications locomote using wheeled mechanisms, a well-known human technology for vehicles.

There are four major wheel classes: standard (rotation around the wheel axle and the contact point), castor (two degrees of freedom), Swedish (three degrees of freedom) and ball (difficult realization). These wheels can be combined in different ways creating

numerous configurations depending on the needs of the application in terms of stability, maneuverability and controllability.

4.2.2 Kinematics

Kinematics is the most basic description of how mechanical systems behave. Within the field of mobile robots, dynamics constraints, i.e., consideration of the action of forces, are also included specially in the case of high-speed mobile robots.

The robot is modeled as a rigid body on wheels, operating on a horizontal plane. The total dimensionality of this robot chassis on the plane is three: two for position in the plane and one for orientation along the vertical axis, which is orthogonal to the plane.

In order to specify the position of the robot on the plane, a relationship between the global reference frame of the plane and the local reference frame of the robot can be established, as in figure 4.1. The axes X_I and Y_I define an arbitrary inertial basis on the plane as the global reference frame from some origin O: $\{X_I, Y_I\}$. To specify the position of the robot, a point P on the robot chassis is chosen as its position reference point. The basis $\{X_R, Y_R\}$ defines two axes relative to P on the robot chassis and is thus the robot's local reference frame. The position of P in the global reference frame is specified by coordinates x and y, and the angular difference between the global and local reference frames is given by θ. The pose of the robot can be described as a vector with these three elements [19].

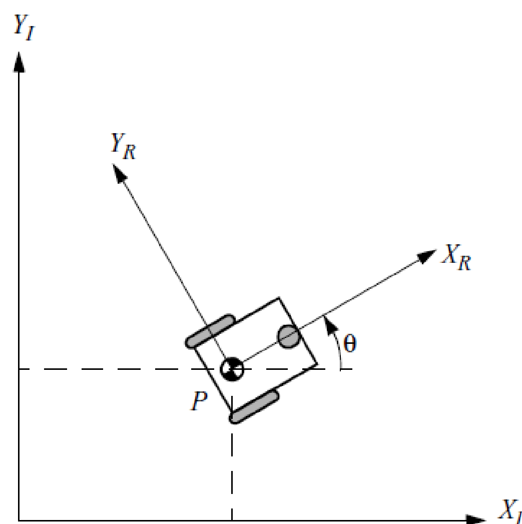


Figure 4.1: The global reference frame and the robot local reference frame [19].

4.2.2.1 Motion control

Many mobile robots use a drive mechanism known as differential drive. It consists of two drive wheels mounted on a common axis and usually placed on each side of the vehicle.

The wheels can independently being driven either forward or backward and hence does not require an additional steering motion. To balance the robot, additional wheels may be added.

Motion or kinematic control of a differential wheeled vehicle can be approached by two methods [19]:

- **Open loop control**. In this first approach, the trajectory to be followed is divided into motion segments of defined shapes, such as straight lines and parts of a circle. The control problem is thus to precompute a smooth trajectory based on line and circle segments which drives the robot from the initial position to the target position. This approach can be regarded as open-loop motion control, since the measured robot position is not fed back for velocity or position control, meaning it will not adapt to dynamic changes of the environment.

- **Feedback control**. A more appropriate approach is to use a real-state feedback controller. With this kind of controller the robot's path-planning task is reduced to setting intermediate positions (subgoals) lying on the requested path. A typical problem is showed in Figure 4.2 and consists of finding a feedback control law such that the position error with respect to the target tend to zero through the control of linear $\nu(t)$ and angular $\omega(t)$ velocity.

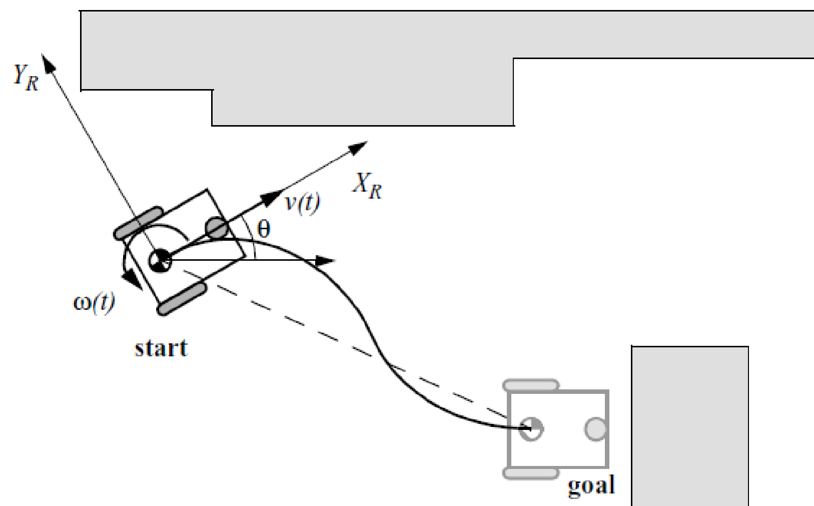


Figure 4.2: Typical situation for feedback control of a mobile robot [19].

4.2.3 Perception

It is vital for an autonomous mobile robot to acquire knowledge about its work environment and itself. This is achieved by means of sensors and subsequently extracting relevant information from those sensor measurements [17].

Mobile robot sensors are grouped into two important functional axes: proprioceptive or exteroceptive, and passive or active. They are briefly described below:

- *Proprioceptive* sensors read values internal to the robot (motor speed, wheel load, etc.)
- *Exteroceptive* sensors acquire information from the robot's environment (distances, light intensity, etc.)
- *Passive* sensors measure ambient environmental energy (microphones, cameras, etc.)
- *Active* sensors radiate energy into the surroundings and then measure the reaction (ultrasonic sensors, laser rangefinders, etc.).

When talking about perception, it is worth emphasizing odometry. The idea behind odometry is to use data from the robot motion sensors to estimate change in position relative to a starting location. In mobile robots, the most common form of odometry is wheel-odometry, in which the position and velocity estimations are obtained from a rotary encoder attached to each wheel (proprioceptive sensor). Because the sensor measurement errors are integrated, the position error accumulates over time and the position needs to be updated from time to time.

4.2.4 Navigation: localization-based versus behavior-based solutions

In creating a navigation system, it is clear that the mobile robot will need sensors and a motion control system. Sensors are absolutely required to avoid hitting moving obstacles, among others, and some motion control system is required so that the robot can deliberately move. It is less evident, however, whether or not the mobile robot will require a localization system. Localization-based and behavior-based are the two possible approaches in this regard, whose differences are outlined in Figure 4.3.

Regarding the former, first, the robot interpret its sensors to extract meaningful data. Next, a (usually complex) world model is built, and the robot must decide how to act to achieve its goals within this world model, before finally deciding upon an action, which is executed in the real world. Therefore, this approach strongly relies on localization and cognition modules.

On the other hand, the behavior-based alternative designs sets of simple behaviors or actions (running simultaneously or not) that together result in the desired robot motion. Fundamentally, this approach avoids explicit reasoning about localization and position, and thus generally avoids explicit path planning as well [19].

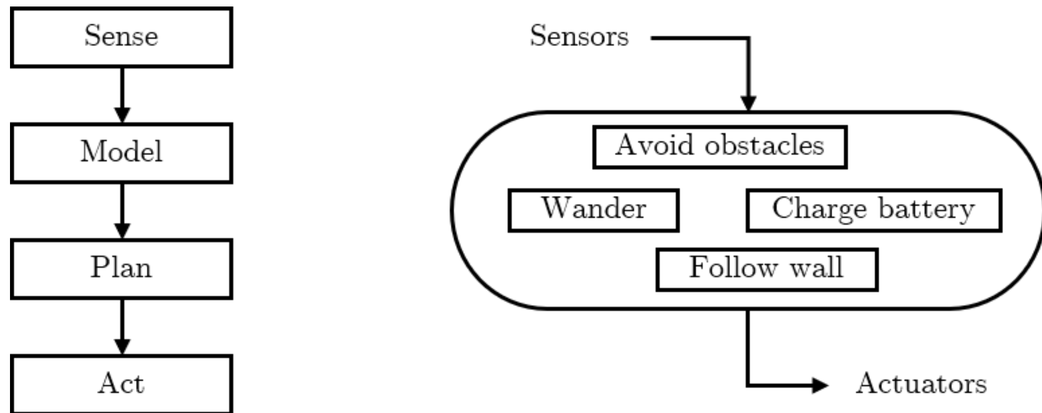


Figure 4.3: Comparison of the information flow in localization-based (left panel) and in behavior-based (right panel) approach. For the latter, any number of behaviors may be involved, and the figure only shows an example with four behaviors.

The best solution depends on the application, being necessary to study the operating conditions, especially the type of environment, the complexity of tasks to be carried out and the final objective of the overall system.

4.3 Equipment description

Once the theoretical basis is established, the equipment for this part of the solution needs to be addressed. In this section, the choices in terms of robot hardware and software resulting from the analysis in Chapter 2 are described in more depth. Besides the purely technical specifications, some considerations are included in connection with the fundamentals learned above.

The first subsection explains the most relevant characteristics of the selected Small Mobile Robot of DTU, including the sensors and actuators it uses. Next, the software subsection covers its Operation System (OS), the main modules and the corresponding programming language.

4.3.1 Hardware

As is known, the SMR platform is a custom made hardware developed by Automation and Control department of DTU. It is a differential-drive vehicle with two motorized standard wheels at the rear and two castor wheels at the front. This versatile robot consists of the basic elements presented below [1]:

- The Computer is a standard μATX. The chipset used on the motherboard has sound and USB capabilities. A plug-in PCI card carries a PC-Card WaveLAN wireless LAN card connecting to the Internet.

- Motor and power amplifiers are connected to the computer through the RS-485 serial communication bus on the SMR. The velocity servo in the power module uses the encoders for measuring wheel speeds.

- Power supply is ATX compliant and enables the SMR to run on battery or external power while recharging. When fully charged, it can run for approximately 2 hours. The battery is a 12 V sealed lead-acid battery and has a capacity of 7 Ah.

- 7 IR-distance sensors are connected to the RS-485 bus to enable obstacle avoidance and sensing the environment. They measure at distances up to 60 cm.

- Encoders on each wheel give 2000 tics per wheel revolution equal to approximately 0.1 mm resolution. The encoder values are read from the power amplifier module on the RS-485 bus.

- Also connected to the RS-485 bus, a reflectance sensor measures surface reflectance (used for line following) and a laser scanner handles further localisation and obstacle detection.

- Wireless LAN connects the SMR to the section WLAN and the Internet.

The Lepton infrared camera should be considered an additional sensor, which can be easily connected to the robot thanks to the USB capabilities. It is mounted on top of the vehicle and oriented in the same direction to get a satisfactory field of view.

4.3.2 Software

The software is based on the standard Slackware distribution of Linux. The control software is based on Mobotware developed at AUT, which is a hierarchical distributed system based of plugins and communication through TCP-IP sockets. The system has three core blocks:

- *Robot Hardware Deamon (RHD).* Flexible hardware abstraction layer for real-time critical sensors.

- *Mobile Robot Controller (MRC).* Real-time closed–loop controller of robot motion and mission execution.

- *Automation Robot Servers (AURS).* Advanced framework for processing of complex sensors and soft real-time mission planning and management.

The hierarchical structure makes allows for programming the system at all levels and the use of plugins makes it easy to add new functionality depending on the needs of each application.

To make programming tasks as easy as possible, an interpreted robot control language has also been developed at AUT and included in the architecture. The so-called SMR-CL language is targeted specifically for robot applications giving an efficient program being able to run on even small computers. SMR-CL is inspired by Colbert, but while Colbert relies on C syntax, SMR-CL requires very little programming skills. Specific robot issues such as setting velocity and acceleration can be done as options to commands giving a compact and readable code, while its simple structure gives transparency to the algorithm at hand. Furthermore, this language provides built-in features like multiple stop conditions and handles real-time issues at the appropriate level.

As for the robot operation, the motion controller is provided with pose information based on odometry, free space information based on IR-distance sensors and line information based on the reflectance sensor, and outputs speed commands to the two motors. The primitive motion commands are the following: `stop`, `fwd`, `turn`, `turnr`, `drive`, `follow_line` and `follow_wall` [15]. The basic format in SMR-CL is:

$$command\ \ parameters\ \ [@v\ velocity]\ \ [@a\ acceleration]\ \ [:\ controlconditions] \tag{4.1}$$

where [ ] means optional.

Thanks to the WLAN connection, all user files are located on the section server and mounted automatically using NFS. In the case of this project, the implementation of the solution is carried out through a Virtual Machine, whose OS is Ubuntu 18.04.5 LTS.

4.4 Design and implementation

The development of the solution regarding the module responsible for tracking hot objects is discussed here. The section begins with a definition of the approach to be followed, specifying the characteristics of the proposed solution with emphasis on the sensors and actuators to use, the type of navigation and the sequence of tasks of the robot. Next, the proposed algorithm that makes this sequence possible is described.

4.4.1 Approach

Before implementing the algorithm in charge of this module, it is important to define which sensors and actuators are necessary for the system to carry out its tasks successfully. Regarding the sensors, the first and most obvious is the Lepton 3.5 IR-camera, since it provides the needed hot object data. Its characteristics and functions have already been described in detail in the previous module.

Among the default sensors of the SMR platform, only the wheel encoders and IR distance sensors are employed. The encoders are useful for position and velocity estimations, i.e., odometry of the system, which the controller needs when executing motion

commands. The IR-sensors are used in order to avoid possible obstacles in the path and, in the case of this project, to know when the hot object location is reached. Specifically, the front three sensors are used because the robot approaches the hot object head-on.

Regarding the actuators, as the SMR is a differential-drive vehicle, the use of both motors for the drive wheels is mandatory for any movement to be carried out. In addition, taking into account that the focus of this project is on firefighting, the system should be able, in a real application case, to extinguish the detected fire by means of a kind of water hose. However, neither this actuator nor the required operating and environmental conditions for testing are available. Accordingly, its use is considered beyond the scope of this project, although the proposed solution is adapted and prepared for its inclusion when necessary.

Another important aspect to define is the type of navigation of the robot. As it is known from the theoretical framework, the choice between a localisation-based and behavior-based solution strongly depends on the need of localization and cognition modules. In the case of this project, the infrared camera gives information about where in the room the hot object is, but it is not able to calculate the distance from the robot to the object. This implies that the exact target position is unknown and the robot should follow the direction towards the object indicated by the camera, relying on the IR-distance sensors (which are short-range) to know when to stop because the object has been reached. This strategy obviates the need for direct localization as well as the decision-making at the robot's cognitive level (e.g., path planning), which would require a given environment map and goal location. By not relying on a predefined map, this navigation strategy is more robust in the sense that it can operate effectively regardless the surroundings, which is consistent with the analysis in Section 2.4 regarding the consideration of the environment as unknown. That being said, the proposed type of navigation is closer to a behavior-based approach.

Furthermore, it is worth emphasizing that following the direction towards the hot object implies that the robot should turn left if the camera detects that the object is on the left, turn right if it is on the right, and go straight if it is roughly in the middle. This kind of robot movements have no complexity and they are reactive behaviors, i.e., there is a direct relation between sensors and actuators, which is another central condition that characterizes behavior-based solutions.

However, internal states, which provide the robot with memory, are also included in a behavior-based approach. These internal states are actually needed for this project application, for example to store the initial position.

To determine the whole list of different tasks or behaviors to be carried out by the SMR, it is necessary to recall the objectives of the project in relation to this module. Given the fact that they involve clearly differentiated actions, each objective has been assigned to one or more tasks as follows: the robot should be able to check for any fire in the room (checking task), track and reach the fire location (going), extinguish the fire (firefighting), and return to the starting position (returning plus initial position). This

set of behaviors is outlined in Figure 4.4, which also summarizes the present section, since exposes the type of navigation and highlights the sensors and actuators employed for the solution of this module.

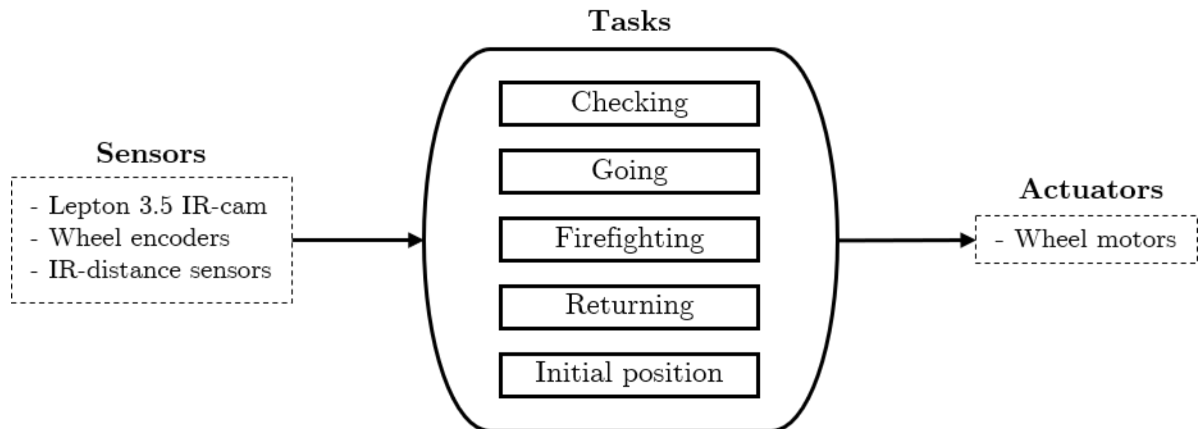


Figure 4.4: Scheme showing the behavior-based proposal used to solve the hot object tracking module.

4.4.2 Algorithm

This subsection addresses the algorithm that allows the robot to perform the set of tasks in Figure 4.4. The following lines describe the whole process, making some references to Appendix A.2, which contains the SMR-CL source code of the `SMR_SCRIPT` file responsible for the robot's objectives of this module.

In order to make it possible, a sequence of states needs to be applied through different labels and the corresponding conditions to obtain the desired general behaviour of the robot. In SMR-CL, the commands `label "labelname"` and `goto "labelname"` allows for jumps in the code flow and are therefore used to switch from one task to another when needed. This is important, since the order in the task sequence may vary depending on the hot objects found. As a comment, to talk about the five possible tasks of the robot, the words task, state, behavior or action are used interchangeably. The variable `State` is a numeric value or flag that is updated at the beginning of each code block corresponding to a task to keep track of which task the robot is performing at all times. It is worth mentioning that in this approach, no task occurs simultaneously with another.

The movements of the robot strongly depends on the hot object data received from the infrared camera, which is continuously updated in three internal system variables: `$l8` indicates if a hot object is found or not (`found` variable in the detection algorithm from Section), `$l9` indicates the pixel index on the horizontal axis (`x_coordinate`), and `$l3` indicates the pixel index on the vertical axis (`z_coordinate`). The implementation of this communication between both modules is addressed in Section 6.3.

The list below covers a description of the different robot tasks that make up the sequence of states of the algorithm. Further details at the code level can be consulted in Appendix A.2.

- **Checking**. The robot performs a rotation (`turn` command in the code) until a fire is detected during the specified checking time (`CheckingTime` parameter, which is set to 10 seconds). There are three possible situations:
  - *Initial checking.* Rotation performed at the beginning of the program to check for any fire in the room.
  - *Intermediate checking.* If the robot is going towards the fire and for some reason the camera suddenly stops detecting it, a rotation is performed trying to detect it again in the surroundings.
  - *Final checking.* Once the robot has returned to the home or starting position, it makes a rechecking to ensure there is no more fire in the room.
- **Going**. Once the fire is detected (`found = 1`), the robot tracks it thanks to the feedback from the infrared camera until its location is reached. Figure 4.5 shows the established acceptable limits in the captured frames for fire tracking, resulting in three possible movements:
  - *Turn left.* The robot should turn left if the fire is detected in the left panel (`x_coordinate < 60`). This turn is performed with a certain radius so that the movement is smooth, which is implemented with the `turnr` command and a radius parameter of 1 meter.
  - *Turn right.* The robot should turn right if the fire is in the right panel (`x_coordinate > 100`). The command used is the same as before but with an opposite sign angle.
  - *Move forward.* The robot should go straight if the fire is detected in the central panel (`60 < x_coordinate < 100`). The SMR-CL command used here is `drive`.
- **Firefighting**. This task begins when the fire is reached, which is known when any of the three front IR-sensors detect a shorter distance than a specified safe distance (`SafeDistance`, which is set to 0.2 m). The IR-distances are stored in the following system variables: `$irdistfrontleft`, `$irdistfrontmiddle` and `$irdistfrontright`. The firefighting task consists of an animation pretending to extinguish the fire, since, as already mentioned, the robot would be equipped with a water hose in a real application. Some turns from side to side covering a 90 degrees angle make up the animation until the fire is no longer detected (`found = 0`), meaning the firefighting work has been completed successfully.
- **Returning**. First, the robot faces home by means of a turn, whose angle is calculated as the difference between the current orientation of the robot and the

one if it was facing home (whose coordinates were stored at the beginning of the program). Next, the robot moves forward until home location is reached. The mentioned movements relies on estimations from odometry, whose position coordinates and orientation of the robot at any time are stored in the system variables `$odox`, `$odoy` and `$odoth`. If a new fire is detected during the return, the program will jump directly to the going task in order to track it.

- **Initial position**. Once the robot is at home and has completed the final checking, it goes back to the initial angle by means of a turn, whose angle is calculated as the difference between the current orientation and the one at the beginning of the experiment, which was also stored. Again, this movement relies on odometry. Consequently, the start and end position will not match exactly, but it is not the goal of this project. Although the fact of returning to the initial position is useful because it is where the robot is supposed to have better visibility of the room to perform the checking tasks, the application mainly aims to track any fire detected by the infrared camera.

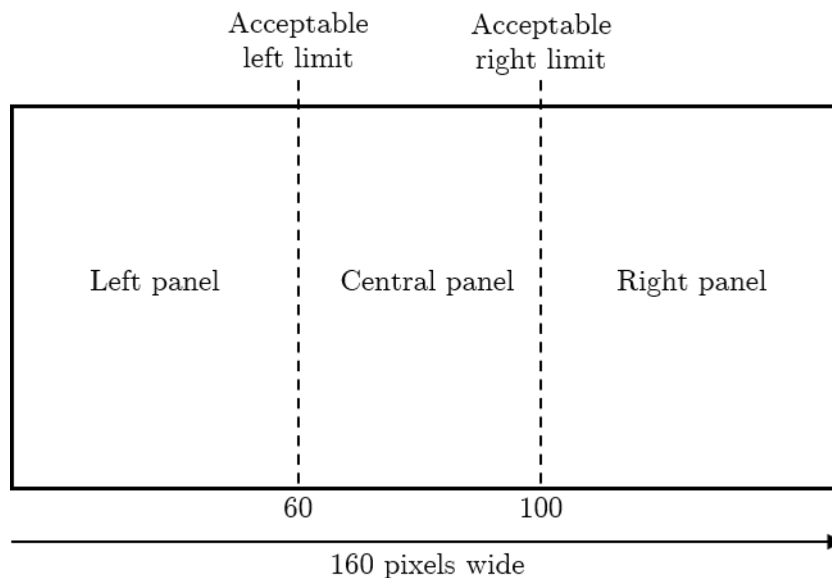


Figure 4.5: Scheme showing the acceptable limits in the frames captured by the IR-camera for hot object tracking. The robot should move so that the detected hot object remains on the central panel.

The sequence of tasks explained above will continue as long as any new fire is detected before the initial position task is performed.

4.5 Module test

The implemented solution has been tested to verify the fulfillment of the objectives regarding the hot object tracking module. As this test is carried out independently of the rest of modules, some small adjustments needs to be applied to the algorithm, namely to the hot object data that will be obtained from the infrared camera in the final system test.

With respect to the `found` variable, it is temporarily replaced by the system variable `$irdistright`, which corresponds to the distance reading from the IR-sensor that is on one side of the robot and is not used in the final solution. By doing this, covering the sensor (using tape for example) represents that a fire has been detected and leaving it uncovered represents the opposite. This simple adjustment allows for controlling the detection of the supposed fire and therefore testing the algorithm at this point of the development. On the other hand, `x_coordinate` and `z_coordinate` indicating the location of the fire in the frame are defined as constants for now.

That being said, several tests are conducted covering three possible situations: the fire remains on the left panel of the frame, on the central panel and on the right panel (see Figure 4.5). In this sense, it has been possible to verify that the robot turns left, continues straight or turns right, respectively, until reaching the target location, that is, the front IR-sensors detect a distance shorter than the safe distance while the right IR-sensor is covered with tape. By covering and uncovering this sensor, it has been guaranteed that in a normal situation where only one fire is detected, the robot performs the sequence of tasks programmed by the algorithm as expected: initial checking, going, firefighting, returning, final checking and initial position. It is worth mentioning that the robot keep extinguishing the fire as long as the it is active, covering an adequate range of action.

Additionally, more complex cases have also been successfully tested. First, the detection of a second fire has been simulated during the final checking, to which the robot has responded by repeating the cycle of tasks. Second, the detection of another fire has been simulated during the return and the robot has interrupted it to switch directly to the going task. Last, a situation has been simulated in which the robot loses sight of the fire during the going task (for example because an obstacle is interposed) and it performs an intermediate checking in order to find it again as expected. If so, it continues with the normal sequence of movements, and if not, it switches to the returning task.

To improve accuracy in terms of odometry and distance reading from the IR-sensors, built-in calibration methods have been applied using the default option of MRC (the program implementing the SMR-CL scripts), that is, by means of the `MRC -c` command in the terminal window logged on to the robot. Odometry calibration consists of following several closed square trajectories clockwise and counterclockwise, whereas IR-sensors calibration consists of placing an object at different known distances from the robot. By doing this, the accuracy achieved in the tests is increased in such a way that the initial and final position of the robot hardly differ and the distances read by the IR-sensors are

much more reliable, which allows a better approach to the fire and detection of possible obstacles.

Lastly, it has also been possible to check that the two hours of battery life offered by the robot are sufficient for the experiments this project deals with. As a result, it is not necessary to include any battery recharging task.

4.6 Summary

In the chapter presented here, the Small Mobile Robot has been addressed as the main tool that allows for the tracking of hot objects detected by the Lepton infrared camera. As a first step, a theoretical framework has been set to contextualize the fast expanding field of autonomous mobile robots and review the primary concepts regarding locomotion, kinematics, perception and navigation. For the latter, it is worth emphasizing the differentiation between localization-based and in behavior-based approaches depending on the need for the application of explicit reasoning about localization and position, and path planning.

A description of the SMR has been included to show the features of this hardware custom made by AUT, together with the corresponding considerations regarding the architecture of the software used. The simplicity and versatility of the SMR-CL language, which is also custom made, has been highlighted in this section.

Regarding the design and implementation of the solution, it has required the use of the Lepton 3.5 IR-camera, IR-distance devices and wheel encoders as sensors, and the wheel motors as actuators. Considering the operating conditions, it has been argued the need to design a sequence of different actions or behaviors to be followed by the robot that correspond to the goals of this module. Specifically, the application framework has focused on a firefighting mission, which has been finally tested considering different situations in terms of fire location in the frame and number of fires detected. The SMR has responded satisfactorily to all possible situations and has shown improved accuracy when applying calibration methods to odometry and IR-sensors.

CHAPTER 5
Data representation

5.1 Introduction

The present chapter discusses the treatment of data collected by both the camera and the robot during the experiment and the way it is represented. The data representation module does not influence the results of the experiment in any way , since it takes place a posteriori, but it constitutes a key phase of the project because it allows the study and analysis of those results.

Given that all the knowledge applied here is directly related to the already studied fields of infrared cameras and mobile robots, no additional theoretical background or equipment description is required. Accordingly, the following sections assume that the reader has sufficient context of the project at this point and focus on the development of this part of the solution.

The content is divided into two main sections. Firstly, the creation of the animation from all the frames captured by the camera is addressed along with some improvements to facilitate its interpretation. Next, another visualization is described, but in this case to simulate the robot trajectory and direction throughout the experiment.

The chapter ends with some testing and checks that the two implemented partial solutions satisfy their objectives.

5.2 Thermal imaging animation

This thermal imaging animation aims to cover the approach and every decision made with regard to the proposal of data representation for the infrared camera. The corresponding implementation is attached in Appendix A.1 and it is a Python function called `plot_data` that belongs to the `CAM_SCRIPT.py` file already addressed in Section 4.4.2. Although its details at the code level can be consulted in the functions definition part of that script in the appendices, the most relevant considerations are also described in this section.

As a reminder, the need for a high frame rate is the reason why the hot object detection and data representation have been separated into two different modules. By placing the code responsible for creating the animation after the capturing loop of the camera, it has been possible to verify that the computational time greatly decreases and

the robot response is much more precise accordingly.

To follow this approach, it is worth bearing in mind that the entire experiment should be reconstructed once it has finished, meaning that various data needs to be stored during the capturing loop. The first and most important piece of information is the image, that is, the captured frame. Saving the image itself in each iteration would be an option, but the problem is that once it is saved with a standard format (JPEG for example), pseudo colors are automatically applied to the pixels and the thermal information is lost, so the exact location of the hot object can no longer be indicated nor its temperature. This would also imply the use of an array to store the information about pixel coordinates and temperature of the hot object corresponding to each frame. Additionally, the process of saving and afterwards opening and loading an image is very slow. Instead, saving only the raw image array is much more efficient, since it is a faster command and the thermal data is not lost. Therefore, the proposed solution initializes an array and concatenates a new array corresponding to the new captured frame in each iteration. After the experiment, the resulting array contains all captured frames stacked vertically in a way that the width of 160 columns remains and the height is 120 rows multiplied by the total number of frames (see scheme in Figure 5.1).

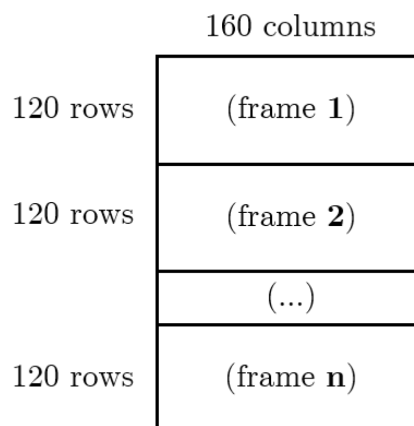


Figure 5.1: Scheme of the array resulting from concatenating every frame captured by the infrared camera during the experiment.

Within the source code, the function that makes this possible is `np.concatenate`, and the array receives the name of `q2` in the capturing loop and `arr_full` inside the `plot_data` function. As one might expect, the format does not change, i.e., it is still a Numpy array of `uint16` data type.

Apart from the image, a timer can be useful to get a rough idea of how long the experiment is taking. That is why the queue called `toc` is used to store a new time value in every iteration of the capturing loop. The `time.perf_counter` function is used because it provides the highest available resolution to measure a short duration. The initial time value (before the loop) is stored in the `tic` variable and is also needed, since the displayed time is calculated as the difference between it and the `toc` corresponding

to each moment.

To sum up, the `plot_data` function is called after the end of the capturing loop and uses the following arguments:

- `arr_full` is the array containing the sequence of captured frames.
- `image_filter` is the chosen colormap to apply to the images.
- `hot_threshold_celsius` is the preset temperature threshold.
- `tic` is the initial time.
- `toc` is the sequence of time values corresponding to each frame.

Once inside the function, the frames array is read in the same order it was constructed. To do this, two indexes are updated in a loop to indicate where each frame begins and ends so that the array of Figure 5.1 is read from top to bottom.

The differentiation between hot and not hot objects is made by comparison with the threshold in the same way as in the hot object detection module (Section 4.4.2). In case a hot object is found, its X and Z pixel coordinates in the frame are also obtained as in such section.

Matplotlib is employed for visualization, which constitutes a comprehensive and powerful tool that provides a Matlab-like interface. Thanks to this library, the following tasks are performed in order to show the results of the experiment:

- The array of each frame (`arr` in the function) is displayed as an image and a colormap is applied to translate each temperature value into pixel intensity. This colormap or filter is defined in the pre settings part of the code. While many options are available, the so-called `plasma` filter has been selected due to the fact that it offers a nice range of colors that adapts specially good to infrared images, showing clear contrast between hot and cold objects.
- The hottest spot is marked with a red square to indicate its position in the frame. This square only appears if the spot is considered a potential fire, i.e., its temperature exceeds the threshold. A grid is added to facilitate the reading of the fire position.
- Some text is added at the bottom left of the image, which includes the timer and the maximum temperature of the scene (in °C) . This temperature is always visible but when it corresponds to a detected fire, the text becomes bold and red.
- The colorbar plays a key role as it allows to represent which temperature corresponds to each color intensity. It is added to the right and consists of eight marks that adaptively change their values depending on the temperature range measured for each scene.

- Last, the final video is created from all previously processed images and is saved in MP4 format. The frame rate needs to be close the actual one during the experiment to get a realistic animation (in this case 10 fps).

As a brief comment, the Python script attached in Appendix A.1 also contains two visualization functions that have not been included in the final solution but have been useful for some tests during the experimental phase. They can be found in the part of functions definition and are the following: `draw_trajectory` (to check that the camera detects moving hot objects correctly) and `draw_histogram` (for a better understanding of the temperature distribution in a scene)

5.3 Robot map simulation

The software package of the SMR developed at AUT includes a simple robot simulator, which replaces the robot hardware while the control software is exactly the same as when using the real robot (MRC). In a real scenario, the MRC communicates with the hardware via a hardware-server called RHD to which it is connected through a socket (the software arquitecture can be consulted in Section 4.3.2). When simulating, the control software simply connects to the simulator instead of the hardware-server.

Nevertheless, this simulator is intended to be used when employing the default sensors of the robot, such as the IR distance or reflectance sensors. In the case of this project, the whole robot algorithm depends on a external sensor, that is, the infrared camera. As the simulator is not able to receive any feedback from the camera, it cannot be used either for testing or for representing the results.

Instead, a new simulator has been designed specially for this project application. This custom-made simulator has been developed using the Matlab platform and the idea behind it is to represent the movements of the robot in a map once the experiment is done. The Matlab script responsible for the robot map simulation can be consulted in Appendix A.3 for more details.

The input for this script is a log file containing primarily the position and direction data generated by the SMR during the experiment. Therefore, the coordinates and orientation of the robot (x, y, θ) is what the script uses to run the simulator. It is worth mentioning that these variables are based on odometry, which implies that the data is approximate since a little error accumulates over time. For short-duration experiments like the ones in this project and for visualization purposes, this level of accuracy is sufficient.

Another important parameter included in the log file is the state, which is a numeric value representing one of the five possible actions or behaviors of the robot (checking, going, firefighting, returning and initial position).

In the log file, the variables are sampled every 0.01 seconds and stored as a line for

each sample time, which makes it easy for matlab to read the data of interest. This data is extracted from the file and stored in Matlab arrays so that the program can use it. In this way, the position and direction of the robot and its corresponding state is known for each moment.

The figure generated by the program is split into two panels: the robot direction map in the left and the robot trajectory map in the right. The former shows the direction of the robot by means of arrows whereas the latter shows its position throughout the experiment. The robot direction map provides comprehensive information, since different colors are used for the arrows according to the possible behaviors of the robot. This differentiation is made thanks to the state parameter mentioned above.

Apart from the final image that displays the entire experiment, a video is generated to show the progress of the experiment, from beginning to end, in such a way that it is possible to see how the robot is moving and changing its direction in the map when tracking the fire. The implementation behind this is a concatenation of frames within an array, similarly to the previous section. Once all the frames are gathered, a function is used to generate a video file from that array. To get an easy to read result, the frequency of the data plotting is reduced so that not all rows from the log file are used, which would lead to a too dense representation, specially for the arrows of the trajectory map. The frame rate of the video is also adjusted so that it progresses at the same time as the thermal imaging animation and can be displayed together simultaneously.

Although it is not visible from a static image, the simulation in the video file shows a dynamic legend in the trajectory map containing the color arrow and state corresponding to each moment. The states are added to the legend as they occur, and become bold when active. To achieve this and to avoid adding the same state twice (for example in cases where multiple fires are detected), the function called `remove_repeated_elements` is used, whose source code is also attached in Appendix A.3.

5.4 Module test

Once the implementation for both representations is discussed, this section aims to show that the thermal imaging animation and the robot map simulation provides successful results. As at this point in the development of the project the system communication module has not yet been addressed, the tests included here are carried out independently. In the case of the representation for the camera, the captured images are displayed without considering the robot movements, while in the case of the representation for the robot, the states and the path to follow are predefined without taking into account the feedback from the camera. In this way, only what concerns the part of the data representation is tested here.

Regarding the thermal imaging animation, two different situations are considered. Figure 5.2 shows a frame when a hot object is detected, whereas Figure 5.3 shows a frame when no hot object is detected around. In the first case, a thermal resistor is

placed in front of the camera, whereas in the second it is hidden with the hand.

By observing both figures, it is possible to see how the temperature range together with the colorbar are adaptively changed depending on the scene captured by the infrared camera. The selected image filter also accomplish its purpose, since it clearly shows temperature contrasts throughout the image. Furthermore, the temperature of the hot object and its position in the frame are properly marked. Therefore, all the goals of the thermal imaging animation are met as expected and it is ready to play its important role in the ultimate system test by showing the results of an entire experiment through a video.

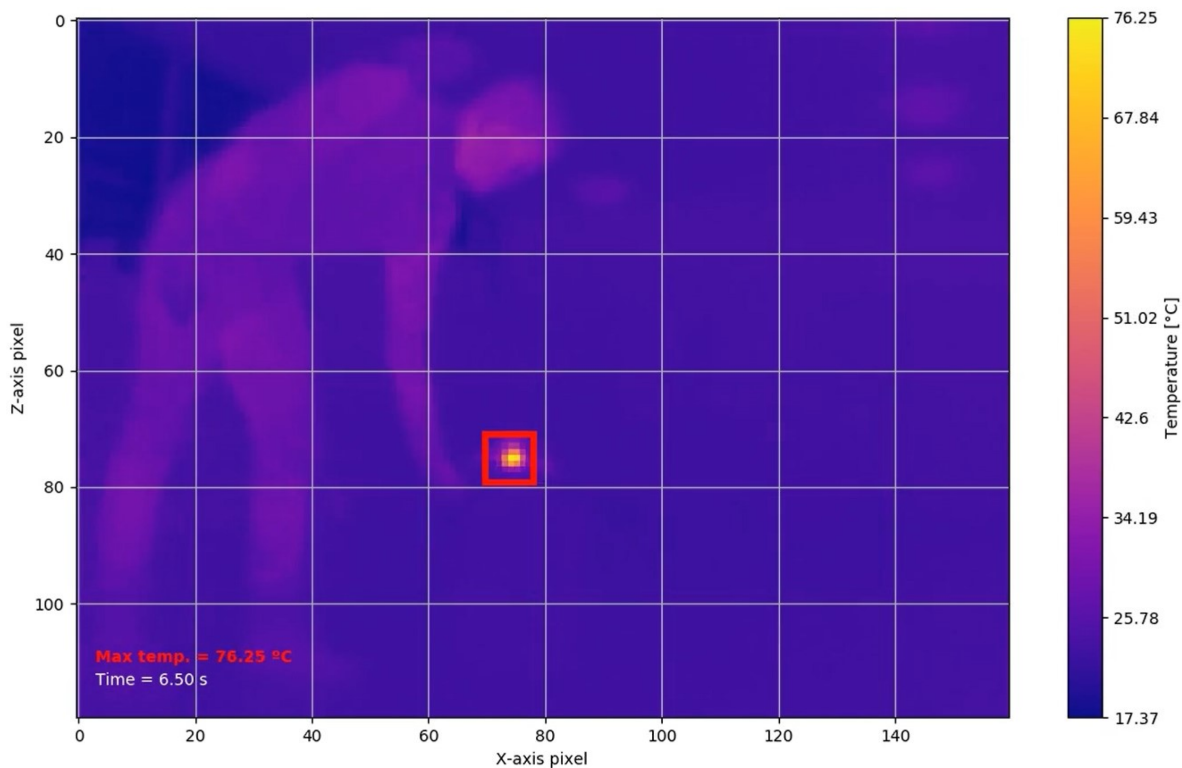


Figure 5.2: Representation of an image captured by the Lepton 3.5 infrared camera when a hot object is detected.

On the other side, the robot map simulation has also been tested. Figure 5.4 shows the image resulting from the simulation of the SMR movements, where the the left map represents the robot direction and the right map represents the robot trajectory. As can be seen, the five possible actions are clearly differentiated by means of different colors and the corresponding explanatory legend. Additionally, the frequency of arrow plotting is suitable, since the image is not too dense and the progression of the robot can be observed in an understandable way. While the direction map provides comprehensive information of the experiment, the trajectory map is especially useful to show a clearer representation when the robot follows more complex paths where the direction arrows may appear mixed or overlapped. That being said, it can be concluded that the robot

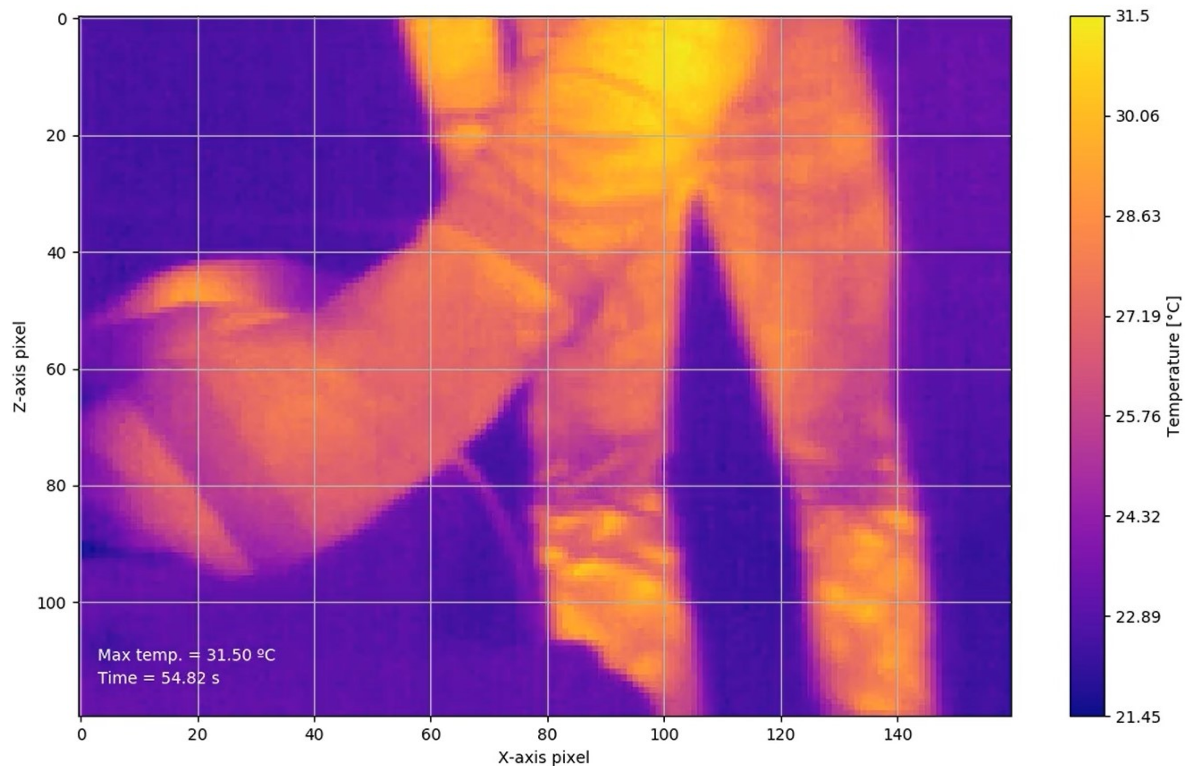


Figure 5.3: Representation of an image captured by the Lepton 3.5 infrared camera when no hot object is detected.

map simulation also fulfills its intention satisfactorily.

5.5 Summary

This chapter has addressed all meaningful considerations regarding the visualization of data collected by the two devices that make up the system. On the one hand, the best way to save the frames obtained by the infrared camera and convert them to readable images with some improvements has been described. On the other hand, the designed simulator has allowed for the representation of the sequence of movements followed by the robot, displaying the trajectory and direction throughout the different stages of the experiment.

Finally, a test module has been included showing that both the thermal imaging animation and the robot map simulation fully display the data of interest in a clear and understandable way. Accordingly, they are powerful tools to consider when analysing any result obtained during the experimental phase.

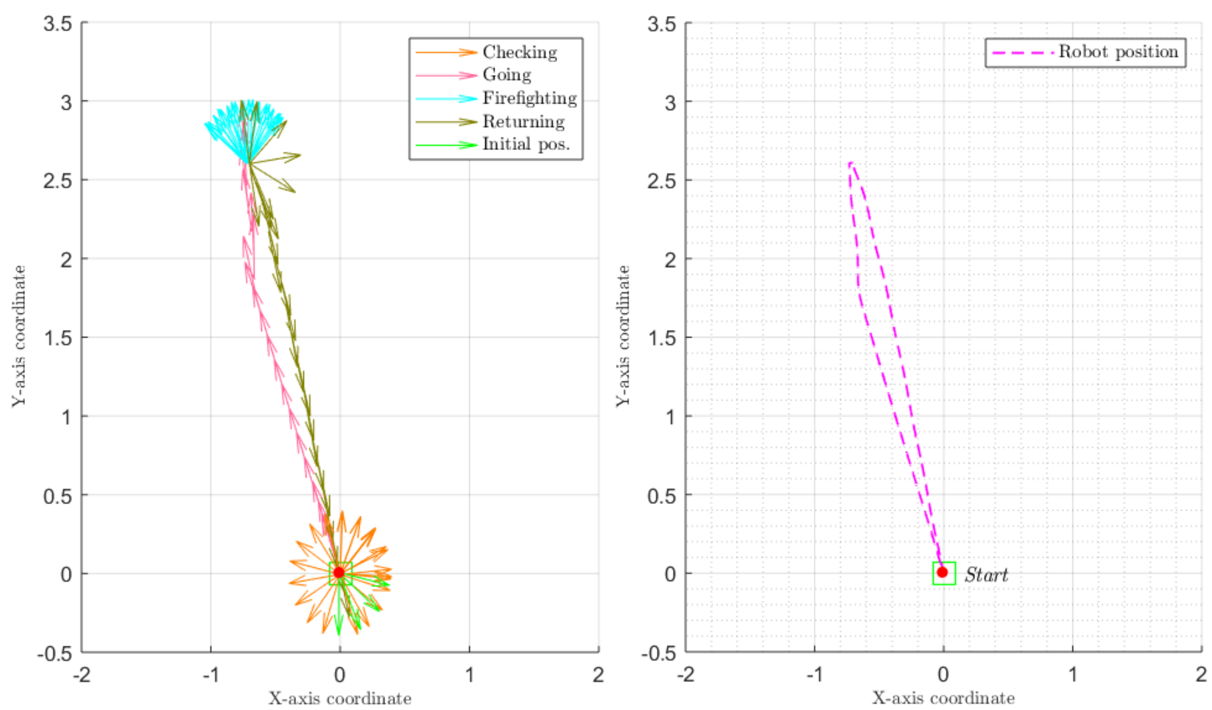


Figure 5.4: Representation of the direction (left panel) and trajectory (right panel) map of the Small Mobile Robot.

CHAPTER 6
System communication

6.1 Introduction

The system communication plays a core role since the rest of the modules depend to a great extent on it. Therefore, it guarantees that the other modules communicate with each other for the correct functioning of the overall system. This chapter describes all interactions between different modules and addresses their implementation by referring to the most relevant considerations at the software level.

The content is split into three sections covering the three connections between the hot object detection, hot object tracking and data representation modules. As regards the testing part, some comments are included within each of these sections to verify that every implemented communication works as expected. However, as all other modules have already been completed, the overall operation of communications is addressed directly in the system test (Chapter 7).

6.2 Detection - Tracking modules

This section deals with the communications between the hot object detection and tracking modules, including all required steps in both directions.

To start with, the detection module just needs to know when the SMR starts running and when it stops, as already explained in Section 4.4.2, Figure 3.6. In the first case, the problem is solved by simply executing both scripts corresponding to both modules (`CAM_SCRIPT.py` and `SMR_SCRIPT`) at the same time, which is implemented in a Linux environment adding the `&` command in the corresponding terminal window. To know when the robot stops, a for loop using the `psutil.process_iter` function has been added at the end of the capturing loop, which returns all running processes on the local machine. By doing this, it is possible to check if there is a process called MRC (the program implementing the SMR-CL scripts), which means that the robot is still running. If it is not the case, the algorithm exits the capturing loop. The source code regarding this is included in the `CAM_SCRIPT.py` file attached in Appendix A.1.

On the other hand, the data required for the correct operation of the tracking module is whether a hot object has been found and its location in the image, which corresponds to the variables `found`, `x_coordinate` and `z_coordinate` from the detection module.

To have access to these variables handled by the Python script, the first step of the proposed solution is to write their values in a CSV file. This file has a single row and three columns corresponding to each variable, which are updated with new values in every iteration of the capturing loop thanks to the `write_to_HOCN` function (whose implementation is also included in Appendix A.1).

Once the hot object data of interest is updated in an intermediate file, the objective is to continuously store their values in local variables of the SMR-CL script. As the laser scanner is a sensor of the robot that is not employed in this application, it leaves some system variables free, so `$l8`, `$l9` and `$l3` are used for the aforementioned camera variables, respectively. The really important aspect about using laser variables is that it allows for tailoring the data stored in such variables by means of plugins (written in C++). Therefore, a custom made plugin (`camplugin`) has been programmed as a bridge between the CSV file and the SMR-CL script. The idea behind this C++ plugin is to read the values from the CSV file and write them in `$l8`, `$l9` and `$l3` on a continuous loop. The fact of sending the values to system variables of the SMR requires a special syntax, which can be found in the corresponding source code attached in Appendix A.4.

The laser scanner server responsible for these communications is called `ulmsserver` and requires a configuration file for its usage. This file called `ulmsserver.ini` contains some initializations such as the server port and, above all, loads the modules or plugins so that they are available in the SMR-CL scripts. Accordingly, the `camplugin` is loaded in such file, which is also attached in Appendix A.4. The last step is simply calling this plugin, which is done at the beginning of the SMR-CL script through the `laser "camplugin findhotobject"` command, where the first word within quotation marks indicates the name of the plugin and the second the name of the specific function inside that plugin. By doing this, the hot object data from the detection module is already available for the tracking module through the continuously updated variables of the laser scanner.

6.3 Detection - Representation modules

The second connection is between the hot object detection and data representation module. In this case, the data transmission is unidirectional, i.e., from the former to the latter, and there is no need for any intermediate file or plugin. Instead, the call to the function `plot_data` already described in Section 5.2 is the direct transition from one module to the other, within the same Python script (attached in Appendix A.1). In this way, the data representation module can mainly read the array that contains all the frames captured by the infrared camera during the experiment and their corresponding time values, which is provided by the hot object detection module.

6.4 Tracking - Representation modules

Last, the input data that the representation module requires from the tracking module is the position and orientation of the robot and the corresponding state or task, as mentioned in Section 5.3. The bridge between both is a log file in this case, which is first generated by the hot object tracking module using the `log` command followed by the required variables at the beginning of the SMR-CL script (whose complete source code is attached in Appendix A.2). Once the experiment has finished, the Matlab script (attached in Appendix A.3) simply extracts the meaningful data from the log file to create the robot map simulation within the representation module.

6.5 Summary

In this brief chapter, the union of all modules has been addressed, in such a way that a correct coordination between them has been guaranteed in order to achieve a satisfactory overall system operation, which will be finally tested in Chapter 7.

On the one hand, bidirectional communications between the hot object detection and tracking modules have been described in detail. It should be emphasised that the sending of hot object data from the detection to the tracking module is implemented by means of a CSV file and a custom made plugin called `camplugin` that uses free system variables belonging to the robot's laser scanner.

On the other hand, both modules also send data collected by the camera and the robot to the representation module for creating a thermal imaging animation and robot map simulation, respectively. While the data collection is performed during the experiment, its sending occurs a posteriori through the call to a Python function in the case of the detection module, and a log file in the case of the tracking module.

CHAPTER 7

System test

7.1 Introduction

Once each module has been implemented and tested individually in Chapters 3 to 6, this chapter aims to test the overall solution resulting from the coordination of all modules. In this way, at the end of this chapter, it will be possible to verify the fulfilment of every objective outlined in Section 1.1.

During this final testing phase, the implemented solution has been thoroughly tested with multiple parameter settings and environment configurations, covering all possible situations that the system may face. However, only the most relevant results are presented here.

Two different real-life scenarios make up the content of this chapter. On the one hand, the test deals with several static fires that the system should extinguish. On the other hand, the system is intended to track a hot object that is in motion, along with its corresponding firefighting. In both cases, image and video documentation are included to display the results of the experiments in detail and facilitate their analysis.

7.2 Static hot object test

The first test addresses the system's ability to track static hot objects. The hyperlink to the video documentation regarding this test is attached in Appendix B.2 and shows the robot direction map simulation on the left side of the screen and the thermal imaging animation on the right. As for the simulation of the robot, only the direction map has been included in the video because it is the map that provides more information about the experiment. However, the resulting image from the simulation of both direction and trajectory map can be seen in Figure 7.1.

Thanks to the colored arrows and the corresponding dynamic legend, it is possible to know the orientation and task that the robot is performing at all times, which is complemented by the images obtained by the camera. Apart from all representation features already explained in Chapter 5, the video also includes in the map a fire icon representing the thermal resistor used in the real experiments, which facilitates the interpretation of its approximate location in the room and of its three possible states: active (normal colors), being extinguished (changing colors) and inactive (gray color).

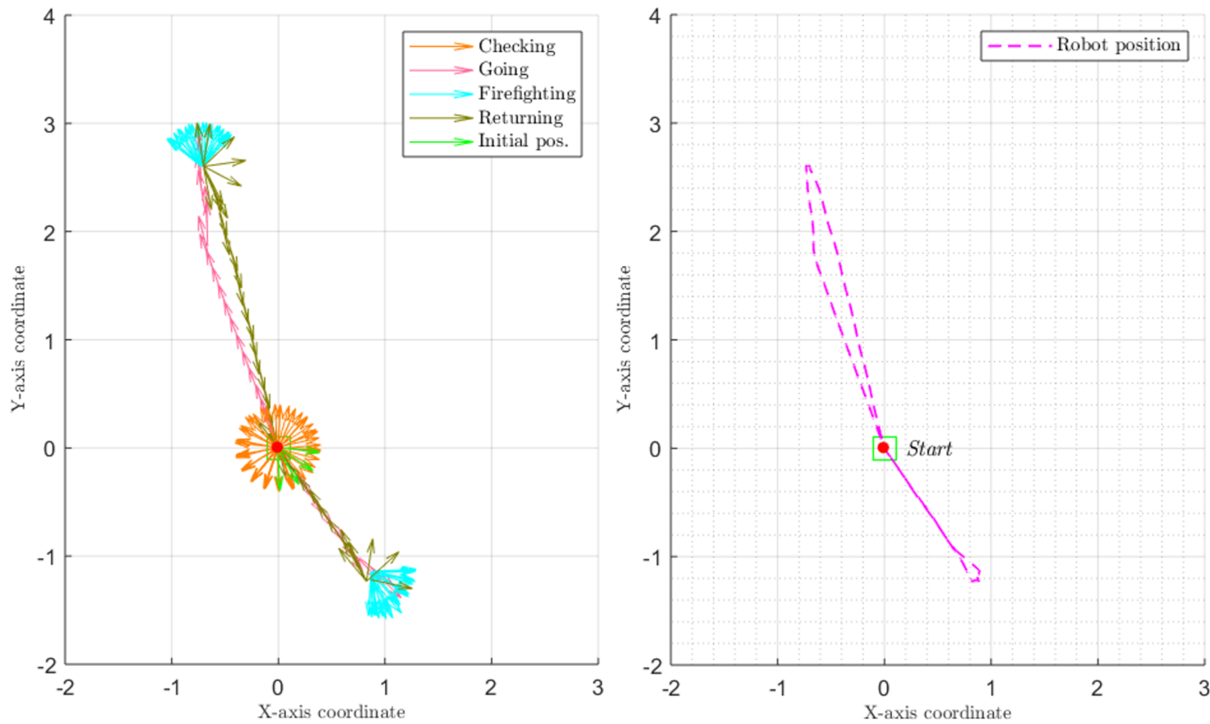


Figure 7.1: Representation of the direction (left panel) and trajectory (right panel) map of the SMR when tracking two static hot objects.

That being said, the test under consideration is described step by step as follows. First, the robot rotates in the starting position until a fire is detected by the IR-camera (*checking task*) and then it tracks and reaches its location in the map (*going*). The robot makes an animation pretending to extinguish the fire, that is, it turns from side to side covering the fire area (*firefighting*). When it is done, which is simulated by hiding the thermal resistor with the hand, it goes back to the starting position (*returning*). As in Figure 7.1 the order cannot be seen, it should be mentioned that this first sequence of movements correspond to the upper closed path.

Next, the robot makes a rechecking in the starting position and it identifies a second hot object representing a fire, so the same cycle of tasks is repeated in order to extinguish it. Once the robot has returned, it checks again to make sure there is no more fire in the room and the process ends by turning back to the initial angle (*initial position*). Therefore, the sequence of tasks carried out by the system throughout the entire experiment is: checking, going, firefighting, returning, checking, going, firefighting, returning, checking and initial position.

By observing the thermal imaging animation in the attached video, it is possible to verify that the aforementioned sequence of tasks matches what the IR-camera perceives at all times. The captured images show the position of the fire in the frame using a clear red mark and indicates the maximum temperature. In this regard, both the fire position and maximum temperature are accurately updated during all the experiment. Equally,

when there is no fire around or it has already been extinguished, no hot object mark appears in the animation. Thanks to this differentiation, it is clear that the camera performs the comparison with the preset temperature threshold in the expected way.

The most important aspect to address regarding this test is the system communication. While it was already known that the detection, tracking and representation functions worked correctly as demonstrated in their respective module tests, this experiment allows for checking the communications between the modules in all possible directions and, accordingly, verifying the overall solution. The first and most obvious point is that both detection and tracking modules are able to send the collected data by the camera and the robot to the representation module, since it has been possible to create the corresponding robot map simulation and thermal imaging animation. Displaying these representations allows, in turn, to check the bidirectional communications between the detection and tracking modules. As can be easily seen in the video, the camera is able to read the robot status, both devices start and ends their operation at the same time. On the other side, the robot successfully reads the hot object data from the camera. This implies that the robot knows if there is a fire or not, since it starts tracking the fire as soon as it is detected by the camera, and stops extinguishing it once the camera no longer detect it. Furthermore, the robot also knows the fire position in the frames captured by the camera, given the fact that it finally reaches the correct fire location in the room. By dealing with two different fires in this test, all the considerations mentioned here are verified twice.

7.3 Moving hot object test

This second test handles a more complex scenario, where the hot object to track is in motion. In the same way as before, the hyperlink to the video documentation regarding this test is attached in Appendix B.2 and the resulting image from the robot map simulation can be seen in Figure 7.2. However, the video shows more comprehensive information in this case, since both the direction and trajectory map are included and a video of the real experiment is added. This is done with the intention of getting a better understanding of how the system works in every regard through this challenging experiment.

When displaying the results in the attached video, the robot map simulation appears at the top of the screen, the thermal imaging animation at the bottom right and the video of the real experiment at the bottom left. The latter is recorded using a normal camera operating in the visual spectrum, which is placed in a high place to have a view of the experiment similar to that offered by the robot map simulation but in real life. This is greatly advantageous in the sense that it allows to check if the simulated robot movements match the real ones as well as the equivalence between what is represented in the camera animation and what it is seeing in reality.

With respect to the test under consideration, the sequence of tasks is described as

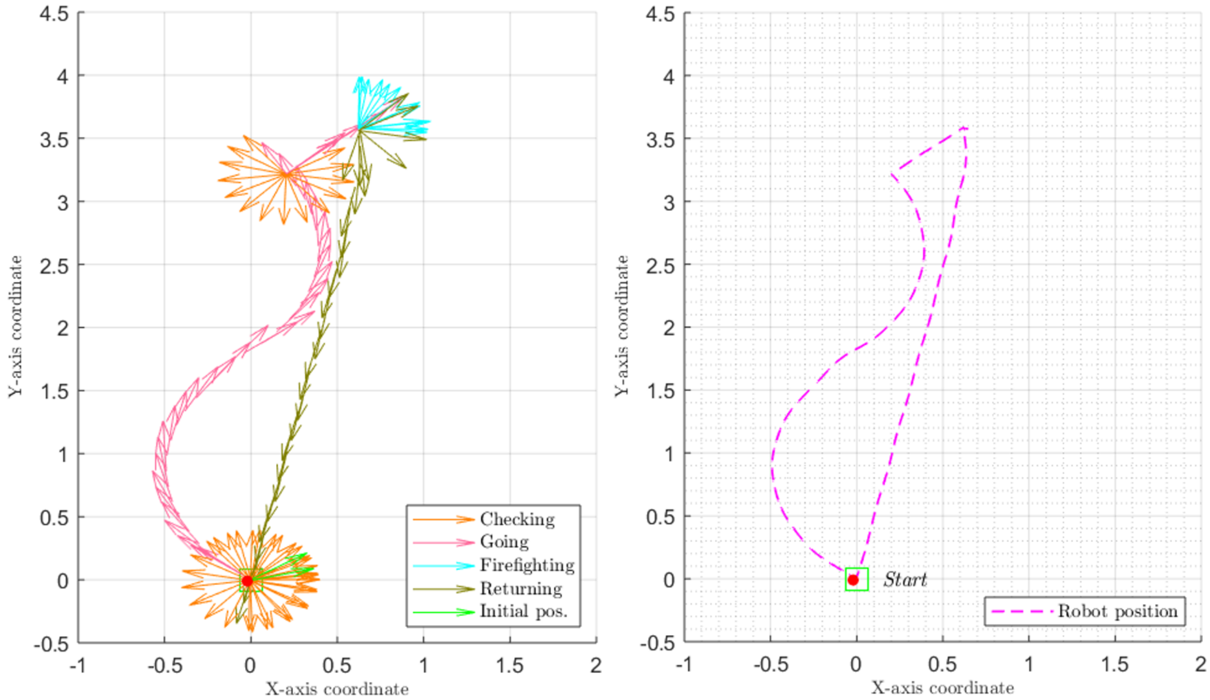


Figure 7.2: Representation of the direction (left panel) and trajectory (right panel) map of the SMR when tracking a moving hot object.

follows. To start with, the robot carries out a rotation from the initial location until a hot object is detected by the IR-camera and it starts moving accordingly. As shown in the video, it first goes straight but the hot object changes its position and, in response, the robot changes the trajectory in order to track the object. The SMR reacts well to changes in the position of the hot object both to the right and to the left. Suddenly, the hot object disappears from the IR-camera view and the robot performs an intermediate checking as expected. After making almost a full rotation, the system detects the object again and keeps tracking it until finally reaches it. The next steps are carried out in the same way as in the previous experiment, that is: the robot makes a firefighting animation until the camera no longer detects the hot object, returns to the starting position, performs a final checking and, as there are no more fires, the robot goes back to the initial angle and stops running. In consequence, in this case, the sequence of tasks throughout the whole test is: checking, going, checking, going, firefighting, returning, checking and initial position.

Apart from all the considerations discussed in the previous section, which can also be applied to this case, this experiment provides important information on the operation of the implemented algorithms. Thanks to it, it is possible to verify one of the major objectives of the project, which is that the camera works accurately and with high frame rate. This can be clearly seen by the smooth trajectories the robot follows and its real-time reactions to the moving object, which again guarantees that the communications between different modules work successfully in any scenario. With respect to such trajectories, it

is visible from the video that the SMR continuously tries, by means of smooth turns, to keep the hot object in the central part of the frames captured by the IR-camera, which leads it to finally reach the target location. Once there, it can also be verified that the safe distance selected to extinguish the fire is appropriate, as well as the range of action during the firefighting task.

7.4 Summary

In this chapter, the overall system has been tested through two experiments within different scenarios. The first test has led to a detailed understanding of how the system works when tracking static hot objects. By analysing the displayed results in the experiment video, it has been possible to verify that all modules work in a coordinate way and lead the system to achieve both robot and camera goals. Specifically, the IR-camera has accurately perceived the hot object data from the environment and the SMR has carried out the correct sequence of tasks accordingly.

On the other hand, the second experiment has addressed a case with a moving hot object, whose final results have shown that the SMR is able to adapt the trajectory smoothly and quickly based on what the IR-camera perceives. A visual video of the real experiment has also been included in the attached documentation so that comprehensive information about all system features and operating conditions is available.

CHAPTER 8
Conclusion

Once the present project has been completed, a series of conclusions can be drawn regarding the achievement of the objectives set at its beginning as well as other important aspects observed throughout the different phases of its execution.

Particularly remarkable is the fulfilment of the main purpose of the project, since a system consisting of an infrared camera and an autonomous mobile robot coordinated with each other to detect and track hot objects has been properly developed and implemented, primarily focusing on a firefighting application. To do this, the solution has been divided into four modules, leading to the corresponding satisfactory results that are highlighted below:

- **Hot object detection**. A Python script has covered the main objectives of the Lepton 3.5 IR-camera in such a way that it captures images identifying if there is a hot enough object in the room and getting its temperature and location in the frame. The achieved frame rate allows hot object data to be available quickly enough for the tracking module.

- **Hot object tracking**. A SMR-CL script has been responsible for the sequence of tasks to be performed by the SMR robot, that is, to check for any fire in the room, track and reach its location, extinguish it and return to the starting position. By using the IR-camera, IR-distance devices and wheel encoders as sensors, and the wheel motors as actuators, it has been possible to implement a fundamentally behavior-based algorithm to achieve the robot objectives.

- **Data representation**. This module has been included to fulfill the goals of the IR-camera and the robot concerning the representation of the data collected by both devices. The thermal imaging animation implemented in Python has shown easily readable results in terms of temperature and location of the hot objects detected by the camera. Additionally, the map simulation developed in Matlab has displayed both trajectory and direction of the robot during the experiments, along with comprehensive representation features such as a dynamic legend indicating the robot active task at all times.

- **System communication**. Lastly, the system communication has integrated all components of the solution and achieved its coordinated operation accordingly. By mainly using intermediate files and plugins, it has addressed the objectives

regarding the continuous transfer of hot object data between the IR-camera and the robot, as well as the sending of data for results representation.

Finally, according to the obtained results, it is worth to conclude that the solution constitutes a robust system capable of tracking both static and moving hot objects within different environments while the design is fully customizable to a real application.

As a brief comment with respect to the United Nations World goals, many applications connected to detection and tracking with infrared cameras can be related to a sustainable society, such us prevention and localisation of energy losses.

Appendices

APPENDIX A

Source code

This appendix contains the source code of the different scripts programmed in this project grouped according to the module.

A.1 Hot object detection script

Listing A.1: CAM_SCRIPT.py

```
1  ##-----------------------------------------------------------------------------------
2  ## IMPORTS
3  from uvctypes import *
4  import time
5  import cv2
6  import numpy as np
7  import numpy
8  from queue import Queue
9  import platform
10 from matplotlib import pyplot as plt
11 import time
12 from collections import deque
13 from matplotlib import cm
14 import matplotlib.animation as animation
15 import csv
16 import datetime
17 import os
18 import psutil
19 from matplotlib import ticker
20
21
22 ##-----------------------------------------------------------------------------------
23 ## FRAME CALLBACK
24 BUF_SIZE = 2  # max number of items that can be placed in the frames queue
25 q = Queue(BUF_SIZE)
26
27 def py_frame_callback(frame, userptr):
```

```
28
29    array_pointer = cast(frame.contents.data, POINTER(c_uint16 *
      ↪  (frame.contents.width * frame.contents.height)))
30    data = np.frombuffer(array_pointer.contents,
      ↪  dtype=np.dtype(np.uint16)).reshape(frame.contents.height,
      ↪  frame.contents.width)
31
32    if frame.contents.data_bytes != (2 * frame.contents.width *
      ↪  frame.contents.height):
33      return
34
35    if not q.full():
36      q.put(data)
37
38  PTR_PY_FRAME_CALLBACK = CFUNCTYPE(None, POINTER(uvc_frame),
    ↪  c_void_p)(py_frame_callback)
39
40
41  ##-----------------------------------------------------------------------------------
42  ## FUNCTIONS DEFINITION
43
44  # Temperature unit conversions:
45  def centikelvin_to_celsius(t):
46      return (t - 27315) / 100
47  def celsius_to_centikelvin(t):
48      if t==None:
49          return None
50      return (100 * t) + 27315
51  def to_fahrenheit(ck):
52      c = centikelvin_to_celsius(ck)
53      return c * 9 / 5 + 32
54
55  # Data logging over time (not used):
56  Hot_Object_Coordinates_Path = "/shome/31388/h1/purethermal1-uvc-capture/python/li⌋
    ↪  buvc/build/Testing/Hot_Object_Coordinates.csv"
    ↪  #customizable
57  trajectory_x = deque()
58  trajectory_z = deque()
59  csv.register_dialect('myDialect', delimiter=';')
60  if os.path.exists(Hot_Object_Coordinates_Path)==False:
61      with open(Hot_Object_Coordinates_Path, 'w', newline='') as file:
62          writer = csv.writer(file, dialect='myDialect')
63          writer.writerow(["DATE", "TIME","X","Z", "TEMPERATURE (ºC)"])
64  def append_row_to_HOC(file_name, x_coordinate, z_coordinate, temperature):
```

```
65        with open(file_name, '+a', newline='') as file:
66            writer = csv.writer(file, dialect='myDialect')
67            datetime_now = datetime.datetime.now()
68            writer.writerow([datetime_now.strftime("%x"),
          ↪   datetime_now.strftime("%X"), x_coordinate, z_coordinate, temperature])
69            trajectory_x.append((x_coordinate)) # in case that drawing the hot object
          ↪   trajectory is needed
70            trajectory_z.append((z_coordinate)) # in case that drawing the hot object
          ↪   trajectory is needed
71
72    # Drawing hot object trajectory (not used):
73    def draw_trajectory(trajectory_x, trajectory_z):
74      plt.plot(trajectory_x, trajectory_z, '--', color='r', marker='s', ms=6,
      ↪   mfc='none', mec='r')
75
76    # Plotting histogram (not used):
77    def plot_histogram(arr):
78      plt.hist(arr.ravel(), bins=256, fc='k', ec='k')
79      plt.title('Histogram',fontweight="bold",fontsize=15)
80      plt.xlabel('Temperature [ck]')
81      plt.savefig("Testing/Test 0/Histogram.png")
82      plt.clf()
83
84    # Data logging for communication with the camplugin:
85    Hot_Object_Coordinates_Now_Path = "/shome/31388/h1/purethermal1-uvc-capture/pytho
    ↪   n/libuvc/build/Testing/Hot_Object_Coordinates_Now.csv"
    ↪   #customizable
86    def write_to_HOCN(file_name, found, x_coordinate, z_coordinate):
87        with open(file_name, 'w', newline='') as file:
88            writer = csv.writer(file, dialect='myDialect')
89            writer.writerow([found])
90            writer.writerow([x_coordinate])
91            writer.writerow([z_coordinate])
92
93    # Data plotting for the video:
94    height = 120
95    start_index = 0
96    end_index = start_index + height
97    def plot_data(arr_full, image_filter, hot_threshold_celsius, tic, toc):
98        print("Processing video...\n")
99        global height
100       global start_index
101       global end_index
102       global total_frames
```

```
103
104     total_frames = round(arr_full.shape[0]/height)
105     print("Total number of frames: {}\n".format(total_frames))
106
107     # Frame initialization:
108     arr = arr_full[start_index:end_index ,  :]
109
110     # Figure initialization:
111     fig = plt.figure(figsize=(11,8))
112     ax = fig.add_subplot(111, aspect='equal')
113     spots, = ax.plot([], [], 's',fillstyle='none', markeredgewidth=4, ms=30,
        ↪  color='r')
114     im = ax.imshow(arr, cmap=image_filter)
115     plt.grid()
116     time_text = fig.text(0.08, 0.155, '', color='w')
117     temperature_text = fig.text(0.08, 0.18, '', color='w')
118
119     # Colorbar initialization:
120     cbar = fig.colorbar(im, label='Temperature [°C]', shrink=0.79)
121     tick_locator = ticker.LinearLocator(numticks=8)
122     cbar.locator = tick_locator
123     cbar.update_ticks()
124     plt.tight_layout()
125
126     try:
127       def updatefig(i):
128             global height
129             global start_index
130             global end_index
131             global total_frames
132
133             # Taking a frame out of the full array which represents the entire
                ↪  experiment:
134             arr = arr_full[start_index:end_index , :]
135             im.set_array(arr)
136
137             # Taking the pixels with the minimum and the maximum temperature:
138             temperature_min_celsius = centikelvin_to_celsius(arr.min())
139             temperature_max_celsius = centikelvin_to_celsius(arr.max())
140
141             # If a hot enough object is found in the room:
142             if temperature_max_celsius > hot_threshold_celsius:
143               # Marking the hottest spot:
144               i_index,j_index = numpy.unravel_index(arr.argmax(), arr.shape)
```

```
145                 spots.set_data(j_index,i_index)
146                 spots.set_visible(True)
147                 temperature_text.set_color('r')
148                 temperature_text.set_fontweight("bold")
149               else:
150                 spots.set_visible(False)
151                 temperature_text.set_color('w')
152                 temperature_text.set_fontweight("normal")
153 
154               # Text update:
155               try:
156                 tocc = toc.get_nowait()
157               except:
158                 return
159               time_text.set_text('Time = %.2f s' % (tocc-tic))
160               temperature_text.set_text('Max temp. = %.2f ºC' %
                  ↪  temperature_max_celsius)
161 
162               # Colorbar animation:
163               cbar.set_clim(arr.min(),arr.max()) # colorbar range update for every
                  ↪  frame
164               cbar_ticks = np.linspace(temperature_min_celsius,
                  ↪  temperature_max_celsius, num=8, endpoint=True)
165               cbar_ticks = np.around(cbar_ticks, 2)
166               cbar.ax.set_yticklabels(cbar_ticks)
167 
168               plt.tight_layout()
169 
170               print("Frame {} / {}".format(i,total_frames))
171 
172               # Index update to take next frame:
173               start_index += height
174               end_index += height
175 
176               return im,
177 
178         plt.ioff() # turns off interactive mode to hide rendering animations
179 
180         # Making animation:
181         try:
182           ani = animation.FuncAnimation(fig, updatefig, frames=9999999,
              ↪  interval=10, blit=True, repeat=False)
183         except:
184           return
```

```
185
186         plt.title('IR-cam tracking',fontweight="bold",fontsize=15)
187         plt.xlabel('X-axis pixel')
188         plt.ylabel('Z-axis pixel')
189
190         # Saving video of the animation:
191         writermp4 = animation.FFMpegWriter(fps=10)
192         ani.save("Testing/Test 0/IR_cam_tracking.mp4", writer=writermp4)
193
194       except:
195         return
196
197
198 def main():
199   print("\n /////////////// START OF THE CAM SCRIPT\n")
200
201   ##------------------------------------------------------------------------------
202   ## PRE SETTINGS
203   image_filter = cm.plasma   # plasma, viridis, bwr...
204   hot_threshold_celsius = 50 # do not store any hot spot data if there is not a
      ↪ hot enough object in the room (if every pixel is below this threshold)
205
206
207   ##------------------------------------------------------------------------------
208   ## FINDING AND OPENING THE CAM
209   ctx = POINTER(uvc_context)()
210   dev = POINTER(uvc_device)()
211   devh = POINTER(uvc_device_handle)()
212   ctrl = uvc_stream_ctrl()
213
214   res = libuvc.uvc_init(byref(ctx), 0)
215   if res < 0:
216     print("uvc_init error")
217     exit(1)
218
219   try:
220     res = libuvc.uvc_find_device(ctx, byref(dev), PT_USB_VID, PT_USB_PID, 0)
221     if res < 0:
222       print("uvc_find_device error")
223       exit(1)
224
225     try:
226       res = libuvc.uvc_open(dev, byref(devh))
227       if res < 0:
```

```
228            print("uvc_open error")
229            exit(1)
230
231        print("Device opened!")
232
233        frame_formats = uvc_get_frame_formats_by_guid(devh, VS_FMT_GUID_Y16)
234        if len(frame_formats) == 0:
235            print("Device does not support Y16")
236            exit(1)
237
238        libuvc.uvc_get_stream_ctrl_format_size(devh, byref(ctrl),
       ↪   UVC_FRAME_FORMAT_Y16,
239          frame_formats[0].wWidth, frame_formats[0].wHeight, int(1e7 /
           ↪   frame_formats[0].dwDefaultFrameInterval))
240
241        res = libuvc.uvc_start_streaming(devh, byref(ctrl), PTR_PY_FRAME_CALLBACK,
       ↪   None, 0)
242        if res < 0:
243            print("uvc_start_streaming failed: {0}".format(res))
244            exit(1)
245
246
247        ##----------------------------------------------------------------------------
248        ## CAM CAPTURING
249        try:
250          # Trying to capture the first frame:
251          try:
252            arr = q.get(True, 10)
253          except:
254            exit(1)
255          q2 = np.array(arr,dtype=np.uint16)
256
257          mrc = 1
258          i=1
259          tic = time.perf_counter()
260          toc = Queue()
261
262          # Capturing while the SMR script is running:
263          while mrc == 1:
264              # Frame capturing:
265              try:
266                arr = q.get(True, 10)
267              except:
268                exit(1)
```

```
269
270             # Concatenating every frame in a big array for later animation:
271             arr_copy = np.array(arr,dtype=np.uint16)
272             q2 = np.concatenate((q2, arr_copy))
273             toc.put(time.perf_counter())
274
275             temperature_max_celsius = centikelvin_to_celsius(arr.max())
276
277             found = 0 # hot object not found
278             i_index = 0
279             j_index = 0
280
281             # If a hot enough object is found in the room:
282             if temperature_max_celsius > hot_threshold_celsius:
283               found = 1 # hot object found
284               i_index,j_index = numpy.unravel_index(arr.argmax(), arr.shape)
285
286             # Sending data about whether a hot object has been found and its X
                ↪  and Z coordinates to the camplugin:
287             write_to_HOCN(Hot_Object_Coordinates_Now_Path, found, j_index,i_index)
288
289             i += 1
290             mrc = 0
291
292             # Checking any process with the name 'mrc', which means that the SMR
                ↪  script is running
293             for proc in psutil.process_iter():
294               if 'mrc' in proc.name():
295                 mrc = 1
296
297         write_to_HOCN(Hot_Object_Coordinates_Now_Path, 3, j_index, i_index) #
            ↪  found=3 means the end of the cam script for the camplugin
298         print(f"\nCam capturing runned in {time.perf_counter() - tic:0.4f}
            ↪  seconds\n")
299
300         # Calling animation function to save the video of the experiment:
301         plot_data(q2, image_filter, hot_threshold_celsius, tic, toc)
302
303
304       ##---------------------------------------------------------------------------
305       ## STREAMING STOP AND CAM RELEASE
306       finally:
307         libuvc.uvc_stop_streaming(devh)
308
```

```
309     finally:
310       libuvc.uvc_unref_device(dev)
311
312   finally:
313     libuvc.uvc_exit(ctx)
314     print("\n /////////////// END OF THE CAM SCRIPT\n")
315
316
317 if __name__ == '__main__':
318   main()
```

A.2 Hot object tracking script

Listing A.2: SMR_SCRIPT

```
1  % \\\\\\\\\\\\\\\\ START OF THE SMR SCRIPT
2
3  %%-------------------------------------------------------------------------------
4  %% CONSTANTS AND VARIABLES DEFINITION
5  Pi = 3.141592
6
7  % Maximum time to wait when trying to find a hot object:
8  CheckingTime = 10
9
10 % Acceptable limits in the captured frames for hot object tracking:
11 LeftLimit = 60
12 RightLimit = 100
13
14 % Parameters for driving commands:
15 PositiveAngle = 30
16 NegativeAngle = -30
17 Radius = 1
18 VelRotation = 0.4
19 VelForward = 0.4
20 VelTurnR = 0.3
21 VelTurn = 0.2
22 VelFirefighting = 0.1
23 VelInitialPos = 0.15
24 SafeDistance = 0.2
25
26 % Flags:
27 Home = 1
```

```
28  State = 0 % 1=checking, 2=going, 3=firefighting, 4=returning, 5=initial position
29
30  % Initial position:
31  eval $odox
32  eval $odoy
33  eval $odoth
34  x0 = $odox
35  y0 = $odoy
36  th0 = $odoth
37
38
39  %%-------------------------------------------------------------------------------
40  %% FIRSTLY
41  % Camplugin call for constant storage on the internal variables $l8=found,
    ↪  $l9=x-coordinate, $l3=z-coordinate:
42  laser "camplugin findhotobject"
43
44  % Data logging for later animation of the robot trajectory and direction:
45  log "State" "$odox" "$odoy" "$odoth" "$l8" "$l9" "$l3"
46
47
48  %%-------------------------------------------------------------------------------
49  %% ROTATION UNTIL HOT OBJECT IS FOUND DURING THE SPECIFIED CHECKING TIME
50  label "Rotation0"
51  t0 = $time
52  label "Rotation"
53  if ($l8==1) "Check0"
54  State = 1
55  turn 15 @v VelRotation
56  if (($time-t0)<CheckingTime) "Rotation"
57  if (Home==1) "InitialPosition"
58  if (Home==0) "Return"
59
60
61  %%-------------------------------------------------------------------------------
62  %% FACING TO THE HOT OBJECT AND TRACKING IT
63  label "Check0"
64  State = 2
65  Home = 0
66  label "Check"
67  if ($l8==0) "Rotation0"
68  if ($l9<LeftLimit) "TurnLeft"
69  if ($l9>RightLimit) "TurnRight"
70  if (($l9>LeftLimit)&($l9<RightLimit)) "MoveForward"
```

```
71
72 label "TurnLeft"
73 turnr Radius PositiveAngle @v VelTurnR
   ↪ :(($l9>LeftLimit)|($l8!=1)|($irdistfrontmiddle<SafeDistance)|($irdistfrontlef
   ↪ t<SafeDistance)|($irdistfrontright<SafeDistance))
74 if (($irdistfrontmiddle<SafeDistance)|($irdistfrontleft<SafeDistance)|($irdistfro
   ↪ ntright<SafeDistance))
   ↪ "Reached"
75 goto "Check"
76
77 label "TurnRight"
78 turnr Radius NegativeAngle @v VelTurnR
   ↪ :(($l9<RightLimit)|($l8!=1)|($irdistfrontmiddle<SafeDistance)|($irdistfrontle
   ↪ ft<SafeDistance)|($irdistfrontright<SafeDistance))
79 if (($irdistfrontmiddle<SafeDistance)|($irdistfrontleft<SafeDistance)|($irdistfro
   ↪ ntright<SafeDistance))
   ↪ "Reached"
80 goto "Check"
81
82 label "MoveForward"
83 drive @v VelForward
   ↪ :(($l9<LeftLimit)|($l9>RightLimit)|($l8!=1)|($irdistfrontmiddle<SafeDistance)
   ↪ |($irdistfrontleft<SafeDistance)|($irdistfrontright<SafeDistance))
84 if (($irdistfrontmiddle<SafeDistance)|($irdistfrontleft<SafeDistance)|($irdistfro
   ↪ ntright<SafeDistance))
   ↪ "Reached"
85 goto "Check"
86
87
88 %%-----------------------------------------------------------------------------
89 %% HOT OBJECT REACHED
90 label "Reached"
91 eval $irdistfrontleft
92 eval $irdistfrontmiddle
93 eval $irdistfrontright
94 eval $l8
95 eval $l9
96 eval $l3
97
98
99 %%-----------------------------------------------------------------------------
100 %% ANIMATION PRETENDING TO EXTINGUISH THE FIRE
101 label "Firefighting"
102 if ($l8!=1) "Return"
```

```
103  State = 3
104  turn 45 @v VelFirefighting
105  turn -90 @v VelFirefighting
106  turn 45 @v VelFirefighting
107  goto "Firefighting"
108
109
110  %%-----------------------------------------------------------------------------
111  %% FACING HOME
112  label "Return"
113  State = 4
114  AngleOdoDeg = $odoth*180/Pi
115  eval AngleOdoDeg
116
117  AngleObjective = atan2(-$odoy,-$odox)
118  AngleObjectiveDeg = AngleObjective*180/Pi
119  eval AngleObjectiveDeg
120
121  AngleReturnDeg = normalizeangledeg(AngleObjectiveDeg-AngleOdoDeg)
122  eval AngleReturnDeg
123
124  turn AngleReturnDeg @v VelTurn :($l8==1)
125  if ($l8==1) "Check0"
126
127
128  %%-----------------------------------------------------------------------------
129  %% DRIVING HOME
130  AngleDrive = $odoth*180/Pi
131  eval AngleDrive
132  drive x0 y0 AngleDrive @v VelForward :(($targetdist<0.05)|($l8==1))
133  if ($l8==1) "Check0"
134  Home = 1
135
136
137  %%-----------------------------------------------------------------------------
138  %% FINAL CHECKING
139  goto "Rotation0"
140
141
142  %%-----------------------------------------------------------------------------
143  %% BACK TO INITIAL ANGLE
144  label "InitialPosition"
145  State = 5
146  AngleHome = normalizeangledeg((th0-$odoth)*180/Pi)
```

```
147 eval AngleHome
148 turn AngleHome @v VelInitialPos
149 eval $odox
150 eval $odoy
151 eval $odoth
152
153 % \\\\\\\\\\\\\\\\ END OF THE SMR SCRIPT
```

A.3 Data representation scripts

Listing A.3: ROBOT_MAP_SIMULATION.mlx

```
1  %%-----------------------------------------------------------------------------
2  %% VIDEO RECORDING
3  clear all; close all
4
5  % Colors to be used:
6  bb = [0 0.2 0.7];
7  rr = [1 0 0];
8  brbr = [0.9 0.5 0.9];
9  gg = [0 1 0];
10 mm = [1 0 1];
11 oo = [1 0.5 0];
12 pp = [1 0.4 0.6];
13 yy = [0.5 0.5 0];
14 cc = [0 1 1];
15
16 % Arrow colors to be used:
17 clrs = [[0.7,0.7,0.7]; oo; pp; cc; yy; gg];
18
19 % Log file from the SMR script:
20 log = load("log"); % "State" "$odox" "$odoy" "$odoth" "$l8" "$l9" "$l3"
21
22 % Taking the relevant data:
23 odo = log(:,2:4);
24 State = log(:,1);
25 N = size(odo,1);
26
27 % Function to draw the arrows representing the robot direction:
28 drawArrow = @(x,y,varargin) quiver( x(1),y(1),x(2)-x(1),y(2)-y(1),0, varargin{:});
29
30 % Initializations:
```

```
31  al = 0.4;
32  del = 3;
33  cx = 1;
34  ri = 1;
35  rf = N;
36  lx = 1;
37  tl = []; lgh = [];
38  p3 = []; p3b = [];
39  aux = [0 0];
40  tlss_old = [];
41
42  % Figure definition:
43  figure('Name', "ROBOT MAPS", "Position", [0, 100, 1000, 500], 'Color','w',
    ↪  'visible','on')
44
45  subplot(122)
46  hold on;
47  scatter(0, 0, 200, 's', 'LineWidth', 1, 'MarkerEdgeColor', gg)
48  text(0.15, -0.02, "\textit{Start}", 'Interpreter',"latex");
49  grid on; grid minor;
50  xlim([-1 2]); ylim([-0.5 4.5]);
51  title("\textbf{Robot trajectory map}", 'Interpreter',"latex")
52  xlabel("X-axis coordinate", 'Interpreter',"latex", 'FontSize',9)
53  ylabel("Y-axis coordinate", 'Interpreter',"latex", 'FontSize',9)
54
55  subplot(121)
56  hold on;
57  scatter(0, 0, 200, 's', 'LineWidth', 1, 'MarkerEdgeColor', gg)
58  grid on;
59  xlim([-1 2]); ylim([-0.5 4.5]);
60  title("\textbf{Robot direction map}", 'Interpreter',"latex")
61  xlabel("X-axis coordinate", 'Interpreter',"latex", 'FontSize',9)
62  ylabel("Y-axis coordinate", 'Interpreter',"latex", 'FontSize',9)
63
64  % Frame initialization:
65  rec(1) = getframe(gcf);
66
67  % Data plotting loop:
68  for i = del+ri:40:rf % it's possible to adjust the data plotting frequency
69
70  subplot(122)
71  if ~isempty(p3), delete(p2); delete(p3); end
72  p2 = plot(odo(1:i,1), odo(1:i,2), '--', 'LineWidth', 1, 'Color', mm);
73  p3 = scatter(odo(i,1), odo(i,2), 30, rr, 'o', 'filled', 'LineWidth', 1);
```

```
74  xlim([-1 2]); ylim([-0.5 4.5]);
75  legend(p2, 'Robot position', 'Location', "southeast", "Orientation","horizontal",
    ↪  'Interpreter',"latex")
76
77  subplot(121)
78  grid on;
79  if ~isempty(p3b), delete(p3b); end
80  q1 = odo(i,1:2);
81  q2 = odo(i,1:2) + al*[cos(odo(i,3)) sin(odo(i,3))];
82  flag = State(i)~=State(lx(end));
83  if flag
84      switch State(i)
85          case 1, cx=2;
86          case 2, cx=3;
87          case 3, cx=4;
88          case 4, cx=5;
89          case 5, cx=6;
90      end
91      lx = [lx; i];
92      tl = [tl; State(i)];
93  end
94
95  h = drawArrow([q1(1), q2(1)],[q1(2), q2(2)], 'MaxHeadSize',30,'Color', clrs(cx,:)
    ↪  ,'LineWidth',0.5); hold on;
96
97  p3b = scatter(odo(i,1), odo(i,2), 30, rr, 'o', 'filled', 'LineWidth', 1);
98
99  if ~isempty(tl)
100     tls = cellstr(num2str(tl));
101     if find(ismember(tls,'1')), aux = find(ismember(tls,'1')); for
        ↪  p=1:length(aux), tls{aux(p)} = 'Checking'; end, end
102     if find(ismember(tls,'2')), aux = find(ismember(tls,'2')); for
        ↪  p=1:length(aux), tls{aux(p)} = 'Going'; end, end
103     if find(ismember(tls,'3')), aux = find(ismember(tls,'3')); for
        ↪  p=1:length(aux), tls{aux(p)} = 'Firefighting'; end, end
104     if find(ismember(tls,'4')), aux = find(ismember(tls,'4')); for
        ↪  p=1:length(aux), tls{aux(p)} = 'Returning'; end, end
105     if find(ismember(tls,'5')), aux = find(ismember(tls,'5')); for
        ↪  p=1:length(aux), tls{aux(p)} = 'Initial pos.'; end, end
106     tlss = remove_repeated_elements(tls,State(i),0);
107     if length(tlss)~=length(tlss_old), lgh = [lgh h]; end
108     tlss_old = tlss;
109     legend(lgh, tlss{1:end}, 'Interpreter', 'Latex', 'NumColumns', 1,'Location',
        ↪  "southeast", 'Orientation', 'horizontal')
```

```
110 end
111 
112 xlim([-1 2]); ylim([-0.5 4.5]); title("\textbf{Robot direction map}",
    ↪  'Interpreter',"latex")
113 
114 % Figure update:
115 drawnow
116 
117 % Frame concatenation:
118 rec =[rec; getframe(gcf)];
119 
120 end
121 
122 tlss = remove_repeated_elements(tls,State(i),1);
123 legend(lgh, tlss{1:end}, 'Interpreter', 'Latex', 'NumColumns', 1,'Location',
    ↪  "southeast", 'Orientation', 'horizontal')
124 
125 rec =[rec; getframe(gcf)];
126 
127 
128 %%-----------------------------------------------------------------------------
129 %% VIDEO CREATION
130 rec2 = [];
131 for i=1:length(rec)
132     if ~isempty(rec(i).cdata), rec2 = [rec2; rec(i)]; end
133 end
134 
135 v_writer = VideoWriter('Robot_animation');
136 v_writer.FrameRate = 2.9; % it's possible to adjust the animation speed
137 open(v_writer);
138 writeVideo(v_writer,rec2);
139 close(v_writer);
```

Listing A.4: remove_repeated_elements.m

```
1 function a = remove_repeated_elements(x,index,N)
2 a=x;
3 k=1;
4 n=length(a);
5 while k<=n
6 j=1;
7 while j<=n
```

```
8   if k~=j
9   if strcmp(a(k),a(j))
10  a(j)=[];
11  n=length(a);
12  end
13  end
14  j=j+1;
15  end
16  k=k+1;
17  end
18  if N==0, a(index) = append('\bf{',a(index),'}');
19  end
```

A.4 System communication scripts

Listing A.5: camplugin.cpp

```
1   /*************************************************************************
2    *   This plugin is derived from the zoneobst files.                     *
3    *************************************************************************/
4
5   #include "ufunczoneobst.h"
6   #include <iostream>
7   #include <algorithm>
8   #include <vector>
9   #include <cmath>
10  #include <fstream>
11  #include <stdlib.h>
12  #include <unistd.h>
13  #include <string>
14  #include <utility> // std::pair
15  #include <stdexcept> // std::runtime_error
16  #include <sstream> // std::stringstream
17
18  using namespace std;
19  unsigned int microsecond = 1000000;
20
21  /// Global variables
22  int found, x_coordinate, z_coordinate, num_errors;
23
24  #ifdef LIBRARY_OPEN_NEEDED
25
```

```
26 /*** This function is needed by the server to create a version of this plugin */
27 UFunctionBase * createFunc()
28 { // create an object of this type
29   return new UFunczoneobst();
30 }
31 #endif
32 
33 bool UFunczoneobst::handleCommand(UServerInMsg * msg, void * extra)
34 {  // handle a plugin command
35   const int MRL = 500;
36   char reply[MRL];
37   bool ask4help;
38   const int MVL = 30;
39   char value[MVL];
40   int i;
41   double zone[10];
42   for(int i=0; i<10; i++) zone[i]=0;
43 
44   // Plugin variables
45   bool calculate_now;
46 
47   // Check for parameters - one parameter is tested for - 'help'
48   ask4help = msg->tag.getAttValue("help", value, MVL);
49   calculate_now = msg->tag.getAttValue("findhotobject", value, MVL);
50 
51   if (ask4help)
52   {
53     std::cout << "\n[ CAMPLUGIN HELP: ]\n";
54     std::cout << "\n>> This plugin is derived from zoneobst files. \n";
55     std::cout << "\n>> Available camplugin options\n";
56     std::cout << "camplugin help      --> This message\n";
57     std::cout << "camplugin findhotobject --> Identifies the object using the
    ↪  results from the scans\n";  }
58   else if (calculate_now)
59   {
60   //-----------------------------------------------------------------------------//
61   std::cout << "\n /////////////// START OF THE CAMPLUGIN\n";
62 
63   // File pointer
64   fstream fin;
65 
66   // Open the file where the cam script is sending the data
67   string Hot_Object_Coordinates_Now_Path = "/shome/31388/h1/purethermal1-uvc-capt
   ↪  ure/python/libuvc/build/Testing/Hot_Object_Coordinates_Now.csv";
```

```
68    fin.open(Hot_Object_Coordinates_Now_Path, ios::in);
69
70    // Read the data from the file as string vector
71    string line;
72    found = 0;
73    num_errors = 0;
74
75
76    while (found!=3) { // found=3 means the end of the cam script
77
78      // Variable 'found' //
79      // Read an entire row and store it in a string variable 'line'
80      getline(fin, line);
81      // Convert string to integer
82      if (!line.empty()) {
83        found = stoi(line);
84        zone[8] = found;
85      } else num_errors++;
86
87      // Variable 'x_coordinate' //
88      // Read an entire row and store it in a string variable 'line'
89      getline(fin, line);
90      // Convert string to integer
91      if (!line.empty()) {
92        x_coordinate = stoi(line);
93        zone[9] = x_coordinate;
94      }
95
96      // Variable 'z_coordinate' //
97      // Read an entire row and store it in a string variable 'line'
98      getline(fin, line);
99      // Convert string to integer
100     if (!line.empty()) {
101       z_coordinate = stoi(line);
102       zone[3] = z_coordinate;
103     }
104
105
106     // SEND DATA TO SMR
107     /* SMRCL reply format */
108     snprintf(reply, MRL, "<laser l3=\"%g\" l8=\"%g\" l9=\"%g\" />\n",
109 ^^I                 zone[3],zone[8],zone[9]);
110     // send this string as the reply to the client
111     sendMsg(msg, reply);
```

```
112      // save also as global variable
113      for(i = 0; i < 10; i++)
114      var_zone->setValued(zone[i], i);
115
116      fin.clear();
117      fin.seekg(0);
118
119      usleep(0.1 * microsecond); // it needs a small delay
120
121    }
122
123    fin.close();
124    std::cout << "Number of errors when reading: " << num_errors << "\n";
125    std::cout << "\n //////////////// END OF THE CAMPLUGIN\n";
126    //---------------------------------------------------------------------------//
127    }
128    return true;
129  }
130
131  void UFunczoneobst::createBaseVar()
132  { // add also a global variable (global on laser scanner server) with latest data
133    var_zone = addVarA("zone", "0 0 0 0 0 0 0 0 0 0", "d", "Value of each laser
     ↪  zone. Updated by zoneobst.");
134  }
```

Listing A.6: ulmsserver.ini

```
1   server imagepath="./"
2   server datapath="./"
3   server replayPath="./log"
4
5   #Setup server for port 20100+N where N is team nr.
6   server port="24919"
7
8   #Load basic modules
9   module load="odoPose"
10  module load="laserPool"
11  module load="v360"
12  # module load for odometry control and global varable access
13  module load=var
14  module load=mappose
15
```

```
16 # live laser scanner on SMR
17 scanset devtype=urg devname="/dev/ttyACM0"
18 scanset def=urg
19 scanset mirror=true
20
21 #Set scanner position with respect of SMR center
22 scanset x=0.255 z=0.04
23 scanset width=180
24 scanset logOpen
25 scanset log=used
26
27 ####################################################
28 ##  Load modules and enter setup commands below   ##
29 ####################################################
30 module load="aupoly.so.0"
31 module load="aulocalize.so.0"
32 module load="auplan.so.0"
33 module load="/shome/31388/h1/purethermal1-uvc-capture/python/libuvc/build/camplug
   ↪  in/camplugin.so.0"
```

APPENDIX B

Online resources

In this appendix, links to online resources of interest for this project are collected.

B.1 Libraries

- libuvc
  `https://github.com/libuvc/libuvc`
- uvctypes.py
  `https://github.com/groupgets/purethermal1-uvc-capture/blob/master/python/uvctypes.py`

B.2 Videos

- Static hot object test
  `https://youtu.be/Jr9R9Qsef0g`
- Moving hot object test
  `https://youtu.be/4Ud4dSFI4ig`

Bibliography

[1] Nils Axel Andersen and Ole Ravn. "SMR-CL, A Real-time Control Language for Mobile Robots." English. In: *2004 CIGR International Conference*. 2004.

[2] Amanda Berg. "Detection and Tracking in Thermal Infrared Imagery." Linköping Studies in Science and Technology. Thesis No. 1744. Sweden: Linköping University, 2016.

[3] Infrared Training Center. *Introduction to Infrared Thermography Basics*. https://cdn.sparkfun.com/assets/f/d/6/2/2/Introduction-to-Infrared-Thermography-Basics.pdf. Accessed: 18-03-2021.

[4] Cutler J. Cleveland. *Encyclopedia of Energy*. 2004.

[5] "Chapter 7 - Electrical heating fundamentals." In: *The Efficient Use of Energy (Second Edition)*. Edited by I.G.C. DRYDEN. Second Edition. Butterworth-Heinemann, 1982, pages 94–114. ISBN: 978-0-408-01250-8. DOI: https://doi.org/10.1016/B978-0-408-01250-8.50016-7. URL: https://www.sciencedirect.com/science/article/pii/B9780408012508500167.

[6] FLIR. *Lepton® 3.5*. https://groupgets.com/manufacturers/flir/products/lepton-3-5. Accessed: 20-03-2021.

[7] Rikke Gade and Thomas Moeslund. "Thermal cameras and applications: A survey." In: *Machine Vision and Applications* 25 (January 2014), pages 245–262. DOI: 10.1007/s00138-013-0570-5.

[8] GetLab. *PureThermal Mini - FLIR Lepton Smart I/O Module*. https://groupgets.com/manufacturers/getlab/products/purethermal-mini-flir-lepton-smart-i-o-module. Accessed: 28-02-2021.

[9] GetLab. *PureThermal Mini - FLIR Lepton Smart I/O Module*. https://groupgets.com/manufacturers/getlab/products/purethermal-mini-flir-lepton-smart-i-o-module. Accessed: 20-03-2021.

[10] Libuvc. *Libuvc: a cross-platform library for USB video devices*. https://ken.tossell.net/libuvc/doc/. Accessed: 21-03-2021.

[11] LYNRED-USA. *Visible vs. Thermal Detection: Advantages and Disadvantages*. https://www.lynred-usa.com/homepage/about-us/blog/visible-vs-thermal-detection-advantages-and-disadvantages.html. Accessed: 18-03-2021.

[12] Vollmer M. and Möllmann K.P. *Infrared Thermal Imaging—Fundamentals, Research and Applications*. 2010.

[13] Kristof Maddelein. "Thermal Imaging Cameras See Through the Smoke." In: (). URL: https://www.techbriefs.com/component/content/article/tb/supplements/it/features/articles/23558.

[14] McQuarrie and Simon. *Blackbody Radiation Cannot Be Explained Classically*. https://chem.libretexts.org/@go/page/13381. Accessed: 17-03-2021. 2020.

[15] Ole Ravn and Nils Axel Andersen. "A Course Programme in Mobile Robotics with Integrated Hands-on Exercises and Competitions." English. In: *Trends in Intelligent Robotics*. Volume 103. FIRA Robot World Congress, FIRA 2010 ; Conference date: 01-01-2010. Springer, 2010, pages 266–273. ISBN: 978-3-642-15809-4. DOI: 10.1007/978-3-642-15810-0.

[16] W. G. Rees. *Physical Principles of Remote Sensing*. 2nd edition. Cambridge University Press, 2001. DOI: 10.1017/CBO9780511812903.

[17] Francisco Rubio, Francisco Valero, and Carlos Llopis-Albert. "A review of mobile robots: Concepts, methods, theoretical framework, and applications." In: *International Journal of Advanced Robotic Systems* 16.2 (2019). DOI: 10.1177/1729881419839596.

[18] Tech Imaging Services. *Application Fields of Infrared Thermography*. https://www.techimaging.com/applications/infrared-thermal-imaging-applications. Accessed: 18-03-2021.

[19] Roland Siegwart and Illah R. Nourbakhsh. *Introduction to Autonomous Mobile Robots*. USA: Bradford Company, 2004. ISBN: 026219502X.

[20] FLIR Systems. *Flir K series different color modes*. https://flir.custhelp.com/app/answers/detail/a_id/931/~/flir-k-series-different-color-modes. Accessed: 15-03-2021.

[21] FLIR Systems. *How Does Emissivity Affect Thermal Imaging?* https://www.flir.com/discover/professional-tools/how-does-emissivity-affect-thermal-imaging/. Accessed: 18-03-2021. 2019.

[22] FLIR Systems. *The Ultimate Infrared Handbook for RD Professionals*. https://www.flirmedia.com/MMC/THG/Brochures/T559243/T559243_EN.pdf. Accessed: 28-02-2021.

[23] Wikipedia. *Atmospheric transmittance*. https://en.wikipedia.org/wiki/File:Atmosfaerisk_spredning.gif. Accessed: 16-03-2021. 2006.

DTU Electrical Engineering
Department of Electrical Engineering
Technical University of Denmark

Ørsteds Plads
Building 348
DK-2800 Kgs. Lyngby
Denmark
Tel: (+45) 45 25 38 00

www.elektro.dtu.dk