BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INTELLIGENT SYSTEMS
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

DETECTION OF PEOPLE IN ROOM USING LOW-COST THERMAL IMAGING CAMERA

DETEKCE LIDÍ V MÍSTNOSTI ZA POUŽITÍ NÍZKONÁKLADOVÉ TERMÁLNÍ KAMERY

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR **MICHAL CHARVÁT**
AUTOR PRÁCE

SUPERVISOR **prof. Ing., Dipl.-Ing. MARTIN DRAHANSKÝ, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2018

Bachelor's Thesis Specification/21293/2017/xcharv16

Brno University of Technology - Faculty of Information Technology

Department of Intelligent Systems　　Academic year 2017/2018

Bachelor's Thesis Specification

For: **Charvát Michal**
Branch of study: Information Technology
Title: **Detection of People in Room Using Low-Cost Thermal Imaging Camera**
Category: Image Processing

Instructions for project work:

1. Study the literature oriented on thermal image processing and indoor detection of people. Familiarize yourself with the low-cost thermal imaging camera available at FIT BUT.
2. Propose an algorithmic approach for counting people in a room and detection of their positions, using the low-cost thermal imaging camera(s).
3. Implement the proposed solution from the previous point.
4. Perform experiments and summarize the achieved results, including discussion devoted to these results.

Basic references:

- Kwasniewska A., Ruminski J. *Face Detection in Image Sequences Using a Portable Thermal Camera*. Proceedings of the 13th Quantitative Infrared Thermography Conference 2016, pp. 493-499, DOI 10.21611/qirt.2016.071.
- Umbaugh S.E. *Digital Image Processing and Analysis*. 2nd Ed., CRC Press, p. 956, ISBN 978-1-4398-0206-9.

Requirements for the first semester:
Items 1 and 2.

Detailed formal specifications can be found at http://www.fit.vutbr.cz/info/szz/

The Bachelor's Thesis must define its purpose, describe a current state of the art, introduce the theoretical and technical background relevant to the problems solved, and specify what parts have been used from earlier projects or have been taken over from other sources.

Each student will hand-in printed as well as electronic versions of the technical report, an electronic version of the complete program documentation, program source files, and a functional hardware prototype sample if desired. The information in electronic form will be stored on a standard non-rewritable medium (CD-R, DVD-R, etc.) in formats common at the FIT. In order to allow regular handling, the medium will be securely attached to the printed report.

Supervisor: **Drahanský Martin, prof. Ing., Dipl.-Ing., Ph.D.**, DITS FIT BUT
Beginning of work: November 1, 2017
Date of delivery: May 16, 2018

Petr Hanáček
Associate Professor and Head of Department

Abstract

As we are approaching times of Industry 4.0 and smart homes, there is a need for automatic reliable mechanisms for detecting living beings. There are many use cases—from aiding elderly or disabled people with their everyday life to increasing safety in dangerous workplaces by guarding hazardous areas. We propose a method for counting, detecting and locating people using a single low-cost thermal camera module FLIR Lepton 3 and Orange Pi computer. This paper describes the process of necessary hardware configuration of the whole system and a software solution to the problem of detecting and locating people. The method, we utilize in order to detect and locate people, is based on applying human temperature filter, image processing with object detection using OpenCV library, and 3D scene reconstruction with known environment parameters. This approach, even in its simplest form, provides accuracy around 90 % on our data set with various possibilities for improvement.

Abstrakt

S příchodem průmyslu 4.0 a chytrých domovů se zvyšují nároky na automatické a spolehlivé technologie umožňující detekci živých bytostí. Tyto technologie můžou například pomáhat starším či postiženým osobám s každodenním životem nebo zvyšovat úroveň bezpečnosti na pracovištích tak, že budou hlídat rizikové zóny. Představujeme metodu počítání, detekce a lokalizace osob za použití jediné nízkonákladové termální kamery FLIR Lepton 3 a počítače Orange Pi. Tato práce popisuje proces konfigurace hardwarových součástí celého systému a softwarové řešení problému detekce a lokalizace osob. Naše řešení je založeno na aplikování teplotního filtru s rozmezím lidských teplot, zpracování obrazu s detekcí objektů s pomocí knihovny OpenCV a rekonstrukci 3D scény o známých parametrech. Tento přístup, i ve své jednoduché podobě, dosahuje na naší kolekci dat přesnosti kolem 90 % a nabízí řadu příležitostí na vylepšení.

Keywords

thermal, detection, counting, locating, person, people, low-cost, camera, image processing, object detection, Orange Pi, Lepton3, OpenCV, scene reconstruction

Klíčová slova

termální, detekce, počítání, lokalizace, osoba, lidé, nízkonákladová, kamera, zpracování obrazu, detekce objektů, Orange Pi, Lepton3, OpenCV, rekonstrukce scény

Reference

CHARVÁT, Michal. *Detection of People in Room Using Low-Cost Thermal Imaging Camera.* Brno, 2018. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor prof. Ing., Dipl.-Ing. Martin Drahanský, Ph.D.

Rozšířený abstrakt

S příchodem průmyslu 4.0 a chytrých domovů se zvyšují nároky na automatické a spolehlivé technologie umožňující detekci živých bytostí. Tyto technologie můžou například pomáhat starším či postiženým osobám s každodenním životem ovládáním chytrých domovů nebo zvyšovat úroveň bezpečnosti na pracovištích tak, že budou hlídat rizikové zóny. Takové zóny si můžeme představit například jako kolejiště na vlakovém nástupišti nebo ve výrobních halách u nebezpečných přístrojů. Mimo jiné se metody detekce a počítání lidí používají také k analýze toku lidí u hromadné dopravy, či správě prostředků a personálu na úřadech nebo v obchodech.

V této práci představujeme metodu počítání, detekce a lokalizace osob za použití jediné nízkonákladové termální kamery FLIR Lepton 3 a počítače Orange Pi PC2. Práce je rozdělena na několik částí.

Úvodem práce rozebírá motivaci projektu, možnosti použití mechanismů pro detekci a lokalizaci osob, aktuálně používané technologie, které srovnává navzájem i s naším řešením a popisuje i jejich konkrétní použití. Tyto mechanismy můžeme obrazně rozdělit do dvou kategorií: jedna, sloužící primárně k jednoduššímu počítání průchodů osob—k analýze toku lidí, a druhá, která do jisté míry zahrnuje zpracování obrazu a detekci objektů, popřípadě lokalizaci objektů ve známé scéně. Druhá kategorie představuje dražší pokročilejší technologie, jako například stereovize nebo pokročilé zpracování videa z kamery. Do této kategorie patří i naše detekce a lokalizace termokamerou. Oproti ostatním řešením má však výraznou výhodu, a to, že přirozeně s malým termálním kamerovým modulem nelze provést rozpoznávání osob, tudíž může být vhodným řešením na místa, kde soukromí hraje důležitou roli, jako domovy nebo pracoviště.

Další část práce popisuje hardwarové součásti systému: termální kameru Lepton 3 a malý počítač Orange Pi PC2 z rodiny Raspberry Pi, konkrétně jejich parametry, z nichž nejdůležitější je způsob komunikace. Lepton 3 používá SPI rozhraní pro odesílání video snímků a I^{2}C pro řízení kamery. Pro komunikaci s kamerou pomocí těchto nízkoúrovňových rozhraní slouží právě malý počítač Orange Pi, jehož procesor obsahuje hardwarové moduly pro komunikaci právě přes tato rozhraní. Práce obsahuje popis konfigurace počítače Orange Pi, která umožňujě komunikaci přes daná rozhraní. Proces aktivace rozhraní není dvakrát transparentní a může být záludný. Práce obsahuje přesný popis nutných akcí pro aktivaci všech rozhraní a poznámky z vlastních zkušeností.

Kromě popisu hardwaru práce obsahuje také podrobný popis protokolů pro přenos videa přes SPI rozhraní a pro ovládání kamery přes I^{2}C, které byly nastudovány z oficiálního produktového manuálu. Znalosti obou protokolů byly využity v další části projektu, který se zabývá implementací knihovny pro přenos video snímků a ovládání kamery *v4l2lepton3*. Vývoj vlastní knihovny byl zvolen po několika neúspěšných pokusech o použití existujících knihoven pro kamery Lepton. Většinou byla existující řešení navrhnuta pro nekompatibilní (starší) verze kamery.

Výsledkem je plně funkční knihovna v C++, která umožňuje přenos snímků beze ztráty synchronizace s kamerou. Součástí knihovny je i Python skript pro čtení jediného snímku a skript pro ovládání kamery.

V následující části práce představujeme algoritmus detekce lidí z jednoho termálního snímku pořízeného kamerou Lepton 3. Náš přístup je založen na zpracování obrazu z jediného termálního snímku s pomocí knihovny OpenCV. Proces detekce se dělí na několik částí: termální snímek se surovými daty senzoru převedeme na reálnou teplotu pomocí experimentálně určené převodní funkce, provedeme vymaskování částí snímku, které neleží v rozmezí teplot lidského těla, na snímek aplikujeme binární adaptivní prahovací funkci, která

převede snímek na černobílý, kde černé pixely jsou pozadí a bílé pixely reprezentují možné osoby, neboť mají teplotu z rozmezí lidského těla. Černobílý snímek filtrujeme morfologickými transformacemi, které odstraní šum, a aplikujeme na něj algoritmus hledání kontur z knihovny OpenCV (`cv2.findContours`). Výsledkem hledání jsou záznamy kompaktních bílých oblastí snímku na černém pozadí. Kolem těchto oblastí můžeme vykreslit ohraničující obdélníky, které v podstatě obklopují detekovaný objekt. Dále lze využít heuristik k dalšímu filtrování detekovaných objektů. Například můžeme pomocí plochy obdélníku filtrovat drobné předměty a podobně. Popsaná metoda s jednotlivými funkcemi je implementována v pomocné Python třídě `ThermoHelper` a je součástí projektu, a to včetně ukázkových skriptů.

Jakmile získáme detekované obdélníky z termálního snímku, můžeme přejít na finální část projektu, a to lokalizaci objektů ve známé scéně, kterou popisuje závěrečná kapitola, a to jak teoreticky, tak prakticky. Metoda lokalizace je založena na rekonstrukci scény zpětnou projekcí obrazových souřadnic do scény.

Nezbytnou podmínkou pro tuto zpětnou projekci je znalost pozice kamery ve scéně. Pozicí kamery rozumíme její natočení (rotaci) a posunutí (translaci) v dané scéně. Určení této pozice popisuje perspektivní problém n bodů (Perspective-n-point problem). Ten řeší perspektivní projekci bodů ze souřadnicového systému scény do systému obrazového (souřadnice obrazových pixelů) pomocí soustavy lineárních rovnic. Pokud dosadíme do rovnice alespoň 4 body představující projekci z bodů scény do bodů obrazových, jsme schopni vypočítat translaci a rotaci kamery v dané scéně, a tím i popsat transformaci všech bodů ze scény do obrazového souřadnicového systému.

Tuto transformaci poté můžeme obrátit a popsat přímku v souřadnicovém systému scény, jejíž body by se promítly do jediného obrazového bodu. Pokud najdeme přímku tvořenou body, které se promítnou do nejspodnějšího bodu detekovaného obdélníku, kde se nohy detekované osoby pravděpodobně dotýkají země, můžeme určit průsečík přímky s podlahou a aproximovat tak, kde se detekovaná osoba ve scéně nachází. Popřípadě jde ještě počítat s průměrnou výškou člověka.

Tato část projektu byla implementována v Python třídě `Scene`, která je abstrakcí známé scény. Její instance umožňuje nastavit hranice scény, souřadný systém a po provedení kalibrace (určení pozice kamery) pomocí čtyř mapujících bodů lze zpětně promítat libovolné obrazové pixely zpět do souřadnicového systému scény (za předpokladu explicitního dodání třetí souřadnice) a zobrazit scénu včetně promítnutých bodů a polohy kamery z ptačí perspektivy.

Práce popisuje do detailu všechny jednotlivé části projektu. Každá část byla softwarově realizována, přičemž komunikace s kamerou, čtení snímků videa a ovládání kamery funguje velmi dobře. Zde není mnoho možností na vylepšení.

Použitá metoda lokalizace objektů ve známé scéně je přirozeně matematicky velmi přesná, záleží jen na přesnosti dodaných mapujících bodů při kalibraci modelu scény.

Nejkritičtější část projektu je detekce lidí z termálních snímků. Naše metoda, i přesto, že je relativně jednoduchá, dosahuje úspěšnosti kolem 90 % na zkoumaných datech. Výhodou našeho jednoduššího přístupu může být stabilita vůči bizarním polohám detekovaných osob, ale co se týká dalšího vylepšení a ošetření některých nesprávně vyhodnocených situací, má metoda své limity. V této oblasti se však nabízí celá řada jiných přístupů detekce objektů jako HOG s SVN, řešení s pomocí konvolučních neuronových sítí, či sledování pohybu objektů. Vylepšování metody detekce lidí z termálních snímků či videa bude předmětem dalšího výzkumu s cílem ještě více vylepšit stabilitu detekčního algoritmu pro reálné nasazení například kombinací různých přístupů.

Detection of People in Room Using Low-Cost Thermal Imaging Camera

Declaration

Hereby I declare that this bachelor's thesis was prepared as an original author's work under the supervision of prof. Ing., Dipl-Ing. Martin Drahanský Ph.D. All the relevant information sources, which were used during preparation of this thesis, are properly cited and included in the list of references.

....
Michal Charvát
July 19, 2018

Acknowledgements

I would like to thank prof. Drahanský for supervising this thesis, for his relaxed and constructive approach and for acquiring necessary hardware components, without which the research would never have even started. Furthermore, I would like to thank prof. Guido Kempter, and also prof. Drahanský, for arranging a student mobility at the University of Applied Sciences Vorarlberg, where I continued the second part of my research, wrote this thesis and was allowed to present an article about this project at the uDay XVI technological conference in Dornbirn. I am also grateful to all the staff at the university for their kindness and willingness to help in every situation.

Contents

Chapter 1

Introduction

This thesis deals with utilizing a single low-cost thermal camera module, small single board computer and image processing to solve the problem of detecting and locating people.

Detecting and locating people have found their usage in many areas of everyday life. It is often used for queue management in shops and in marketing for determining the best product placement. We also encounter people detecting mechanisms in smart homes where they aid to control the environment and most importantly, they can help to ensure safety in heavy machinery workplaces, industry halls or often at train stations by guarding hazard zones.

In simple cases it is sufficient to use technologies, that allow only for simple people counting. They are usually a form of a sensor built into a door frame which detects objects passing through. The advantage of such a simple approach is its cost and also it usually does not allow for person recognition. This privacy issue comes into play when dealing with more complex solutions allowing for detecting people in open areas or even locating people.

By using a small thermal camera module approach, we eliminate the possibility of people and/or face recognition while preserving the capabilities of more complex solutions, so we are still able to detect and even locate people in any known environments. Solution to the problem of detecting people based on a thermal camera could be therefore a viable option for places, where privacy plays an important role.

After briefly going through comparison with currently used technologies of people detection, we describe all hardware components of our system—camera specifications, interfaces, low-level computer dedicated to camera interaction, communication protocols and so on.

The next section of the thesis focuses on necessary communication software developed specifically for this project used for capturing frames with the camera and controlling the camera.

In the last section of the thesis, we present a simple people detection method based on image processing. The method consists of temperature filtering, adaptive thresholding, morphology transformations and object shape (contour) detecting.

We also propose a scene reconstruction technique that enables us to not only detect people and therefore count them, but also approximate their location in a known environment by reverse projecting image points of detected objects.

Chapter 2

Utilization of people detection and current technology comparison

This chapter is split into two sections. The first section deals with usage of people detection in general and the second one compares currently used technologies in people counting/detecting/locating with respect to its use case.

2.1 Utilization of people detection

As stated in the introduction, counting/detecting/locating people has found its usage in many areas and is an essential part of many complex systems.

The following listing describe some of the most important areas, where people detection is used.

Mercantile interests

In marketing it is possible to determine the best placement of a product based on a model constructed with data of customer movement. Usable information, extracted from people detection systems, would be for this use case: in which areas they spend majority of time, what path they tend to take in a particular environment (shop, supermarket). The technology provides us with data we can analyze and optimize for example our advertisement.

Queue management

In shops and at public service places in general, counting and detecting techniques are used to measure number of customers in premises, estimate queue length in real time, measure an average wait time to be served, or staff idle time. The data provided might serve to improve customer experience and manage resources more efficiency. For supermarkets, number of open desks can vary over time based on current queue length, we can distribute staff more efficiently or adjust it on the fly.

People transportation

Similarly to the previous paragraph, for relatively same purposes, detection and counting techniques are vastly used at airports, in subway, and sometimes at train or bus stations.

These automatic people counting solutions are mostly used for people flow analysis. Analyzing people flow statistics is the key to maintain user friendly environment and can serve as an initiative for improvement. We count number of passengers being transported, how full a train/bus gets during a day, and again we measure time delays as with the queue management.

Hazard zone guarding

One of the most important areas, where living beings detection is used, is undoubtedly workplace safety mechanisms. This might include hazardous areas at train and subway stations near railway tracks, in heavy machinery, industry halls near dangerous machines, where no living being should be present while operating. We may utilize people detection mechanisms to alert responsible personnel and help preventing severe accidents from happening.

Smart homes

Another usage of people detecting and locating systems is in smart homes. By getting information about people‘s presence, location and/or pose, the system can control their environment to ease their everyday life. This can help elderly or disabled people with turning machines on or disabling them, if there is a high chance of them forgetting to. In smart home environments, there is a possibility to utilize counting, detecting and also locating people techniques. [3] [17]

2.2 Currently available technologies allowing for people detection

Some of the technologies nowadays being used for purposes of counting/detecting/locating people might include:

- **IR/Laser beam interruption**
- **Laser light burst travel time (LIDAR [1])**
- **GPS/WiFi/Bluetooth tracking**
- **Projecting structured light**
- **3D stereo video analysis**
- ***Monocular video analysis***

The following paragraphs describe these technologies, point out their weaknesses and determine their most suitable usage.

IR/Laser beam interruption

Infrared/laser beam interruption technique can be used only for counting people entering and leaving room, that is why it is usually installed in a doorway. On one side of a narrow

[1] LIDAR – Light Detection And Ranging

passage (entry point, doorway etc.), a transmitter device is installed, a receiver is installed on the other side, as illustrated in figure 2.1. The receiver and the transmitter are connected together and form an invisible barrier of light. When an object steps into the barrier, it breaks the connection between the transmitter and receiver and the system registers plus one count. This general solution yields inaccurate results, when there are more people passing through the sensor close to each other or when they decide to turn around.

This problem is usually solved by installing more advanced sensors. The accuracy of the system can be increased by using multiple barriers and analyzing measured intensities of each sensor to detect special cases, as when people turn around in the doorway. [29] In order to achieve even higher success rate, we might consider using for example a LIDAR 1 based solution.

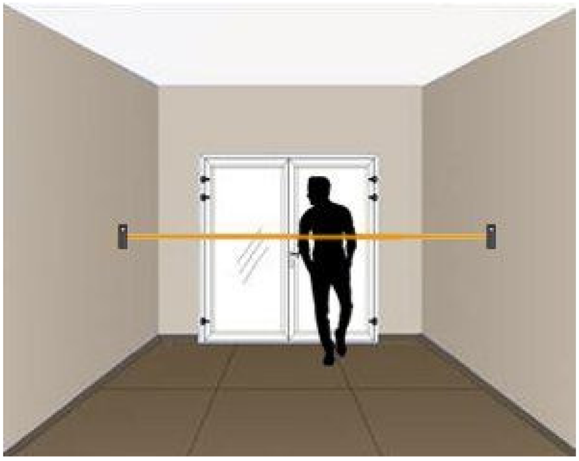


Figure 2.1: Infrared/laser barrier sensor for counting people passing through. (Source: [24].)

LIDAR

Figure 2.2 shows an example of such device. The LIDAR device consists of only a single sensor usually placed above a passage. The sensor is a transmitter and a a receiver at the same time. The device casts laser beams into several directions and precisely measures time required for the reflected beam to get back into the sensor. This way it is possible to calculate the distance the laser beam has traveled and therefore create a depth map.

The LIDAR approach makes counting people in a doorway way more accurate than the beam interruption method. The device can differentiate entering and leaving through the area, and since the sensor is usually installed above the passage, it has no problem with detecting multiple people passing next to each other, which is a major problem with the infrared/laser barrier. This solution is widely used at airports and has a guaranteed accuracy over 95 %. [2]

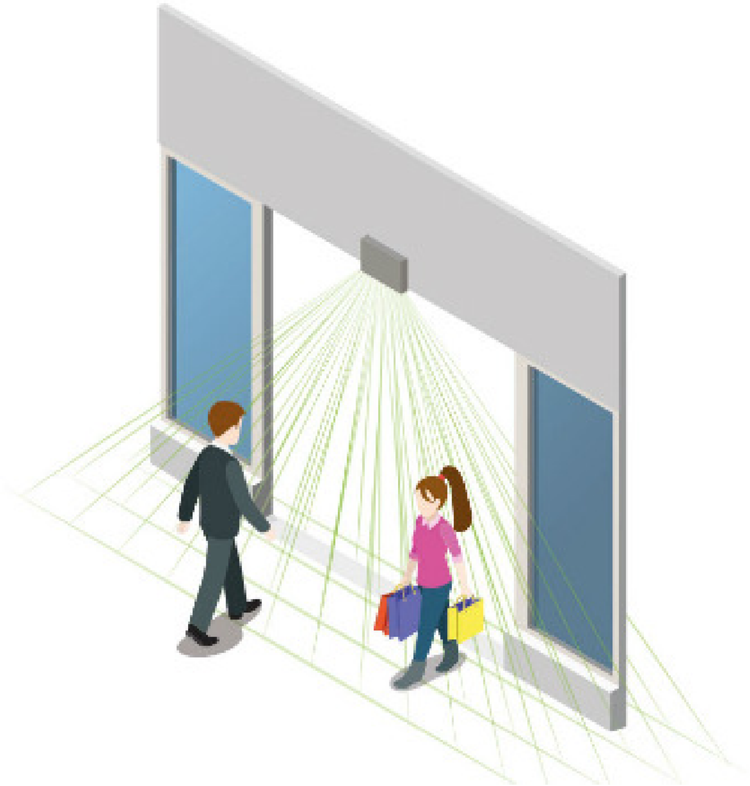


Figure 2.2: LIDAR based device for counting people entering and leaving through a doorway. (Source: [2].)

GPS/WiFi/Bluetooth smart device tracking

With the growth of smart devices supporting wireless technologies such as WiFi or Bluetooth, a new method of tracking people arises. Devices with these technologies make it easy to triangulate their position. Obviously, not every living being needs to carry this kind of a smart device, however, still for merchants, this kind of tracking their customers offers valuable information about their whereabouts. Although it can not be reliably used to detect people, as we would get plenty of false negatives, tracked paths of detected devices can be used to create a map of customer movements, which can help to promote products or optimize advertisement in general. An illustration of a map, representing different durations of people's presence throughout a shop is depicted in figure 2.4.

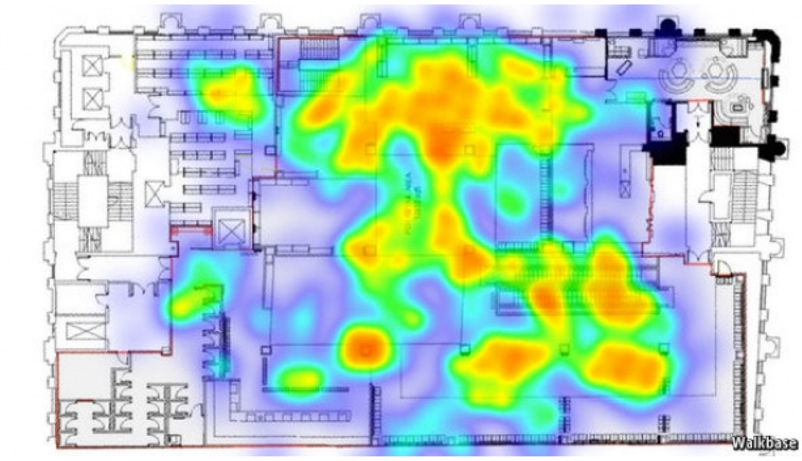


Figure 2.3: Intensity map representing duration of people's presence at a certain location. (Source: [33].)

Projecting structured light

Method of projecting structured light is not used for purposes of detecting or locating people. This technology is often used for obtaining 3D models of relatively small objects or continuous depth maps.

The system consists of usually two parts: a camera and a projector. The projector casts structured light on the scene. The structured light is usually a horizontal black and white line pattern or a checkerboard pattern. The camera is then used to view the scene, and by analyzing deformations in the projected pattern, a depth map is constructed.

On its own, this technique may be used in the same use case as LIDAR sensors—people counting. This setup has one advantage, when compared to the LIDAR solution. LIDAR sensors casts rays only into several directions. By projecting the light pattern, this technique covers larger continuous area and has the potential to be more accurate. As with LIDAR devices, this sensor is placed above a passage and would often use structured light from the invisible infrared spectrum.

The structured light sensors, however, have major disadvantages. Such system requires extreme calibration and is bigger in size and more complex. In order to detect living beings with such sensor, it would have to be joint with other methods such as video analysis.

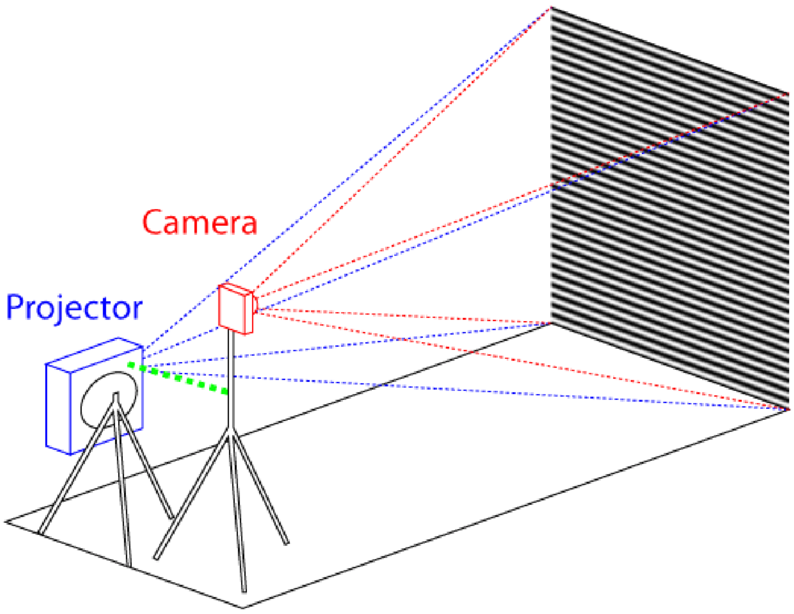


Figure 2.4: Example setup of depth mapping system composed of single camera and a projector casting structured light. (Source: [18].)

3D stereo video analysis

3D stereo video analysis is quite often used in high end solutions. The system consists of two precisely calibrated cameras viewing a scene. The technology is somewhat similar to human vision. Two eyes viewing a scene with the brain extracting depth information from differences in the two images, caused by a different position of each eye. By combining video frames from two cameras, we receive a depth dimension, which generally provides accurate distance measurement. With the usage of such depth map, it is generally more accurate to perform people detection/tracking by image analysis.

This approach can be seen in cutting edge solutions provided for airports, big train stations, where accurate detection and location of as well as counting people is a necessity. The ready made solutions using this technology tend to be very expensive, due to its complexity and calibration requirements.

When compared to the LIDAR and structured light approach for counting people, it has one drawback. It has much narrower field of view, which means, that at places with low ceiling, this is challenging to find a proper location for installing the stereoscopic device. Another drawback, which also applies to the monocular video analysis, is a privacy issue. With the use of regular cameras for purposes of detecting/locating/counting people, it is possible to perform facial recognition, which may be unacceptable in certain situations.

Monocular video analysis

The monocular video or single frame analysis is quite similar to the 3D stereoscopic vision, however, for monocular video analysis, no depth map is used, as the monocular video analysis uses only a single camera.

By video analysis and digital image analysis (used in both monocular and stereoscopic vision), we understand a process of extracting meaningful information from video or images respectively. Today various techniques and approaches are being used. The process can be somewhat generalized into few steps:

1. **Image preprocessing** – preparing the image for analysis by usually digital image processing. This might include filtering, adjusting contrast, dynamic range of the image and so on.

2. **Feature extraction** – extracting indices, that are meaningful for the type of analysis, we want to perform. This might include finding binary contours in the image, lines, corners, extracting histogram of oriented gradients (HOG [9]) and so on.

3. **Final stage of the analysis** – the actual algorithm used to achieve the goal of video/image analysis. The goal might be object detection, classification, recognition and similar. In this stage, we encounter various algorithmic approaches, often from the machine learning family, from simple thresholding, linear binary classifiers all the way to support vector machines and deep neural networks.

Using a thermal imaging camera module to help solving the problem of detecting people also belongs to the monocular video analysis section and brings several advantages, when compared with other approaches. [22] Firstly, we need only a single camera, so there is no need for extremely precise calibration, as with stereo vision or structured light projection. We can detect and also locate living beings in contrast with infrared/laser beam or light travel techniques, which can only count objects entering and leaving an area. The biggest advantage, however, is a fact, that by using a thermal camera, it is *impossible* to perform facial or person recognition. This makes this approach more suitable for places, where privacy plays an important role like workplaces or homes.

Chapter 3

Hardware part of the detection system

This chapter describes all hardware parts of the project, which are: a thermal camera module, Raspberry Pi-like low-level computer and their interfaces, along with the necessary procedure on how to set up individual parts as well as the whole system, to be able to receive video frames from the used camera module.

3.1 Thermal camera module Lepton® 3

This section describes the thermal camera module we use in this project. We will go sequentially through its specifications, modes of capture and communication protocol for both video frames transfer and camera control. Information about the camera in the following section comes from the official documentation [11] [13] [12].

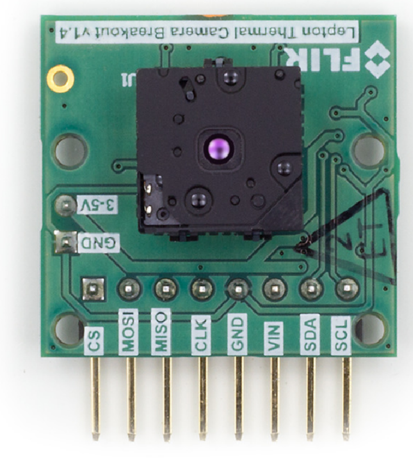


Figure 3.1: Lepton 3 with breakout board. (Source: [14].)

3.1.1 Specifications

In this project we use a Lepton 3 thermal camera module[2] made by the company FLIR[3] which is currently one of the top manufactures of thermal camera solutions. It contains a sensor sensitive to long wave infra red (LWIR) light in range from 8 to 14 μm. The camera module is smaller than a dime and provides decent images with its 160 by 120 pixels resolution. The effective frame rate of the camera is only 8.8 Hz, however for our needs this is not a problem. The camera only requires low voltage supply and has small power consumption of around 140 mW. See table 3.1 for more specifications.

For better manipulation with the camera module we use a breakout board (figure 3.1) with a housing for the Lepton camera module. The breakout board provides better physical accessibility, improves heat dissipation and increases input voltage supply range to 3-5 volts, as it has its own regulated power supply. This power supply provides the camera module with three necessary voltages: 1.2, 2.8 and 2.8-3.1 volts. The breakout board also supplies the camera with master clock signal. [14] The camera uses two interfaces for communication:

- **SPI** for transferring video frames from the camera to a SPI master device.
- **I^{2}C** for receiving control commands from the I^{2}C master device.

Even though the name of the project include the keyword *low-cost*, we need to think of this statement with respect to the thermal camera market. The Lepton 3 thermal camera module can be considered low-cost when compared to other thermal camera devices available—as it costs around $250 (2018) [4]. This could however be considerably more when compared to other *nonthermal* solutions, however way more than if we would utilize a simple infrared counting sensors for example.

Spectral range	8 to 14 μm
Array format	160 × 120 pixels
Pixel size	12 μm
Thermal sensitivity	<50 mK
FOV horizontal	56°
FOV diagonal	71°
Depth of field	28 cm to ∞
Lens type	f/1.1 silicon doublet
Output format	14-bit Y14 raw flux or 24-bit RGB888 false color
Clock speed	25 MHz
Input voltage	2.8 V, 1.2 V, 2.8-3.1 V
Power dissipation	140 mW operating, 5 mW shutdown mode
Dimensions	11.8 × 12.7 × 7.2 mm

Table 3.1: Lepton 3 camera module specifications. [11]

[2] Lepton homepage https://lepton.flir.com/

[3] FLIR homepage http://www.flir.eu/

[4] FLIR Lepton 3 supplier e-shop https://groupgets.com/manufacturers/flir/products/lepton-3-0

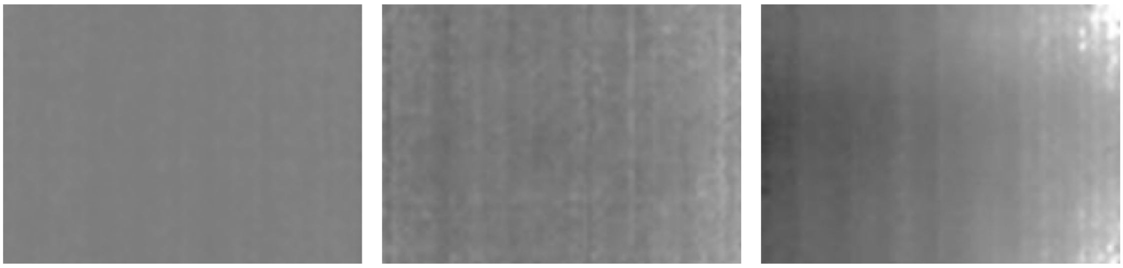
3.1.2 Modes of operation

Lepton 3 module outputs two video formats—standard **24-bit RGB false color** video and **raw 14-bit**, whose pixels correspond to raw input light flux. Both formats will be described in more detail in video transfer subsection 3.1.3. Several features are available with the Lepton 3 camera module—most important ones being:

- flat-field correction (FFC)
- telemetry
- radiometry
- automatic gain control (AGC)

Flat-field correction (FFC)

The Lepton 3 camera should produce highly uniform image when viewing a uniform-temperature scene. Drift effects over longer periods of time degrade imagery and make it look more grainy and/or blotchy (see figure 3.2). In order to fix these issues we can use the flat-field correction (FFC) feature. The FFC is a process of briefly exposing camera's sensor to a uniform thermal scene allowing its signal processing engine to automatically recalibrate to produce optimal image quality. The uniform thermal scene can be provided using Lepton's integral shutter The whole recalibration process takes less than a second. The FFC mode can be set to be performed manually upon command or automatically after a specified period of time.[11][20] In this project we are using automatic FFC with Lepton's integral shutter.



(a) Highly uniform image. (b) Grainy image. (c) Blotchy image.

Figure 3.2: Example of drift effects degrading imagery without the use of FFC. (Source: [11][20].)

Telemetry

Telemetry mode provides us with additional information about the current state of the camera. This data is packed and transmitted with every video frame as a part of it. The packed data is 656 or 976 bytes in size per frame and for example contains information like *power-up time, frame counter, internal temperature, time counter at last FFC, FFC state, AGC clip limits* (more in following section 3.1.2), *overtemperature shutdown imminent flag* and so on. For our purposes we do not require any information provided by the telemetry so we can keep the feature disabled and save some time during video frame transfer.

Radiometry

Radiometric modes affect the translation from incident flux to pixel output of the camera. When radiometry is enabled, the camera ensures that the conversion from camera's output to scene temperature yields constant results over the full operating temperature range of the camera. When disabled, output pixels of the camera, viewing a constant temperature scene, would have different values for different camera temperature. See hypothetical illustration of the negative effect in figure 3.3. As we need to convert incident flux to real scene temperature in this project, we will be using the radiometry feature.

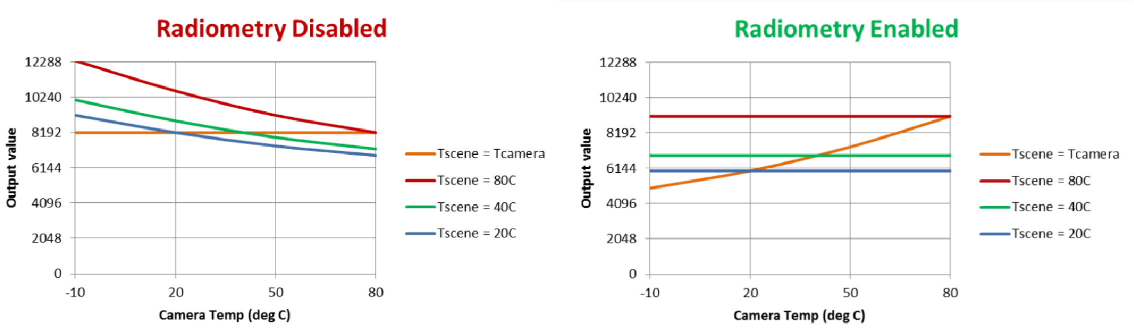
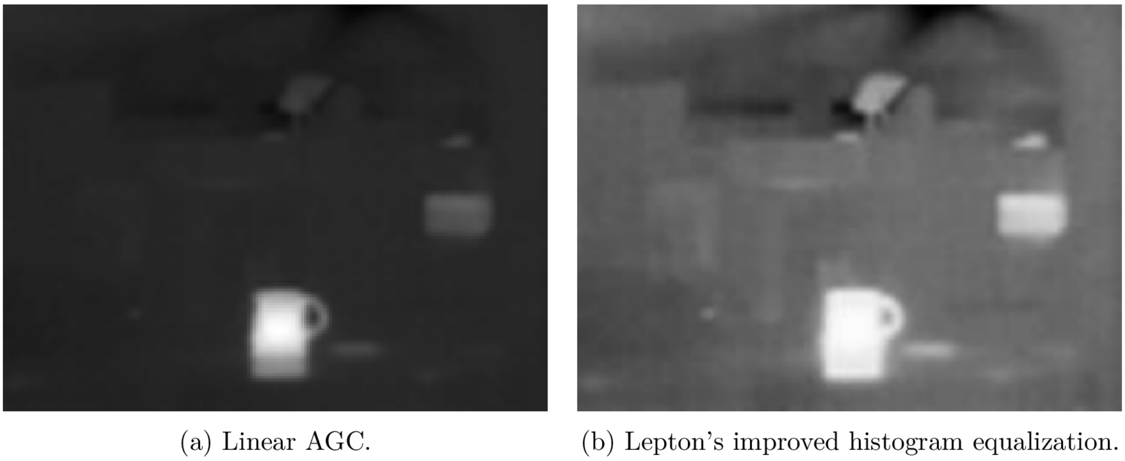


Figure 3.3: Hypothetical illustration of camera output vs. camera temperature. (Source: [11][26-27].)

Automatic gain control (AGC)

AGC, with respect to thermal imaging, is a process of collapsing rather large dynamic range of the infrared sensor to the range more suitable for displaying.

The simplest form of AGC would be linear AGC (scaling). We would find lowest and highest values in the image and map all pixels between these two values to the full range. The linear AGC however has one major drawback: situation of having for example a very hot object in front of a colder background would result in losing almost all details of the background as most pixels would be mapped to either full black or white with very little use of grayshades in between. To eliminate this problem Lepton 3 uses a more sophisticated AGC algorithm, which is an improved version of a *histogram equalization*. Comparison of the linear and Lepton's version of AGC is depicted in figure 3.4. However good the Lepton's AGC algorithm is for presenting a scene to a human eye, we will not be using it in our project, as we would like to measure the real scene temperature and any AGC algorithm completely obscures camera output values, making it impossible to determine the actual temperature of a pixel.



(a) Linear AGC.

(b) Lepton's improved histogram equalization.

Figure 3.4: Comparison of linear AGC and Lepton's histogram equalization. (Source: [11][29].)

3.1.3 Transferring video over SPI

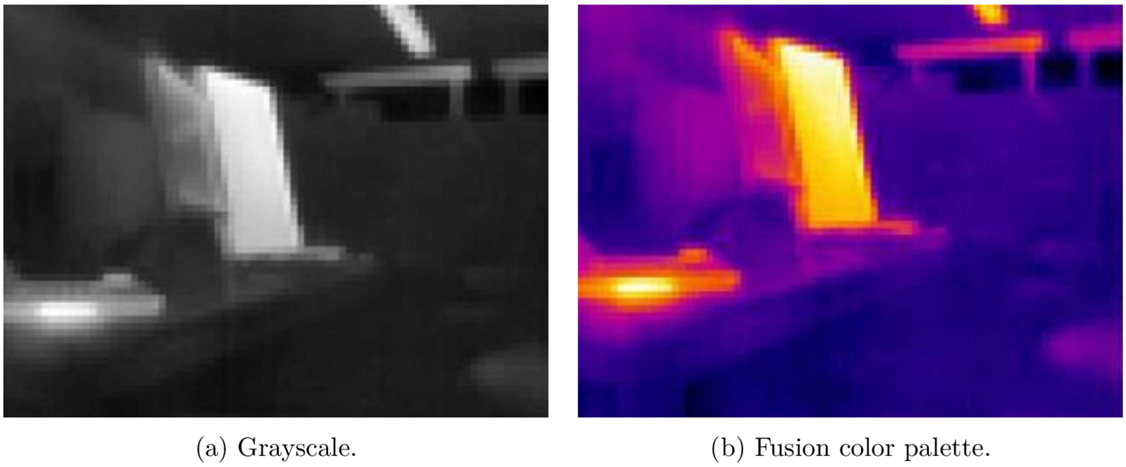
As mentioned previously, the camera can output video frames in two formats:

- Raw Y14 – 1 × 14-bit grayscale – raw flux
- RGB888 – 3 × 8-bit RGB – false color

The first mode is appropriate for viewing or processing raw data captured by the camera module. In this case it only make sense not to use AGC feature, as we would be interested only in unaltered data.

On the other hand, for presenting images to the user, one can preferably use the RGB888 mode, which allows for colorizing the captured image, according to a selected look-up color table (false color palette). Lepton 3 comes with 8 color palettes by default, however, it is possible to load a custom one. See illustration comparing grayscale and false color image in RGB888 mode 3.5.

Note,that AGC feature must be enabled in order to use colorization. Output formats can be switched between using control commands—more in control interface section 3.1.4.



(a) Grayscale.

(b) Fusion color palette.

Figure 3.5: Comparison of grayscale and false color palette (*fusion*). (Source: [11][32].)

For transferring video frames from the camera module to a master control device, the Lepton 3 VoSPI (Video over SPI) protocol is used to transfer data over Serial Peripheral Interface bus (SPI). The protocol is packet-based with no timing signals nor requirement for flow control. The host computer device (SPI master) initiates all transactions and provides clock signal to the Lepton 3 camera (SPI slave).

This way the master can pull data from the camera at a flexible speed. The speed of the transmission is however flexible only up to a certain point: unless you are able to transfer the whole data segment before a new one is available, the camera desynchronizes and restarts the transmission. This happens to be a common problem in the community, caused by the transmission not being fast enough. More on that in software section 4.

The VoSPI utilizes three lines:

- **SCK** – clock signal
- **CS** – chip select
- **MISO** – master In/Slave Out

Typical fourth line **MOSI** (Master Out/Slave In) is not used in this case as there is no need for data transfer from the host to the camera. The VoSPI protocol uses SPI mode 3 (CPOL=1, CPHA=1)[5] and transmits data—most-significant byte first, in big-endian order.

The VoSPI protocol is build on a collection of object data types:

- **VoSPI Packet**: Packet represents the minimum transaction between master and slave. Each video packet contains data for one half of a video line. In case there are no new video segments available, the camera transmits packets with garbage data. Depending on output video format, the packet is 164 or 244 bytes long.
- **VoSPI Segment**: Segment is defined as a continuous sequence of VoSPI packets, representing one quarter of a video frame. To avoid losing synchronization with the camera, each segment must be read out before next one is available. Every segment is 60 packets long. If the telemetry feature is enabled, the number is 61.
- **VoSPI Frame**: VoSPI frame represents the whole real video frame, additionally it could also contain telemetry data. The frame is composed from 4 VoSPI segments.

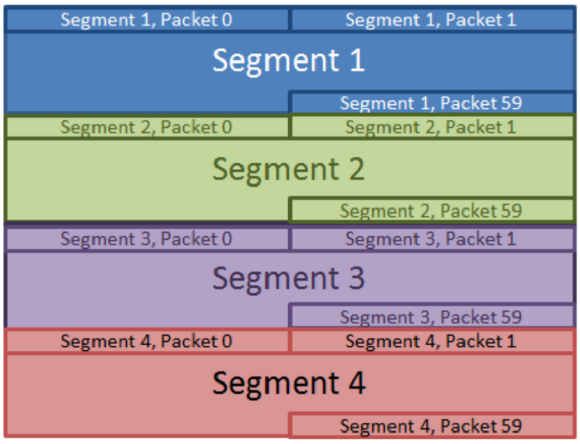


Figure 3.6: VoSPI frame structure illustration. (Source: [11].)

[5] SPI modes – clock polarity and phase https://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus#Clock_polarity_and_phase

Table 3.2 summarizes data object size comparison. Knowing the total size of each frame and having a frame rate of the camera approximately 26.4 frames per second, we can calculate the absolute minimum clock rate for SPI transmission at which the host needs to read frames from the camera to maintain synchronization. For Raw14 mode without telemetry, the minimum SPI clock rate is 8.3 MHz. For a different combination of the video format and telemetry feature, the clock speed will be higher. The minimum SPI clock rate as well as maximum hardware SPI clock speed limit 20 MHz, create a boundary, in which the host device must communicate with the camera.

Telemetry diabled		
Data object	**Raw Y14**	**RGB888**
VoSPI Packet	164 B	244 B
VoSPI Segment	60 packets, 9.84 kB	60 packets, 14.640 kB
VoSPI Frame	4 segments, 240 packets, 39.36 kB	4 segments, 240 packets, 58.56 kB

Telemetry enabled		
Data object	**Raw Y14**	**RGB888**
VoSPI Packet	164 B	244 B
VoSPI Segment	61 packets, 10.004 kB	61 packets, 14.884 kB
VoSPI Frame	4 segments, 244 packets, 40.016 kB	4 segments, 244 packets, 59.536 kB

Table 3.2: Summary of VoSPI data objects size with respect to used features.

The first 4 bytes of every packet contain a packet header. (figure 3.7.) The header consists of 2-byte ID and 2-byte error-detecting cyclic redundancy check code (CRC). The rest of bytes in the packet contain data payload.

ID	**CRC**	**Data payload**
2 bytes	2 bytes	160 or 240 bytes

Figure 3.7: VoSPI packet structure.

According to figure illustrating the packet header 3.8, the first bit is always zero, following three TTT bits encode segment number (1-4)—but only for packet number 20—and next twelve bits hold a packet number. For every other packet except number 20, the TTT bits do not have to contain valid segment information. Example of a valid VoSPI packet can be seen in figure 3.9.

0	TTT	Packet number	CRC
1 bit	3 bits	12 bits	16 bits

Figure 3.8: VoSPI packet header structure.

0	TTT	Packet number	CRC
0	011	0000 0001 0100	CRC

Figure 3.9: Example of a valid VoSPI packet number 20 belonging to a segment 3.

In case a packet number 20 contains TTT bits with value 0, the whole segment is invalid and should be discarded. If the second 4 bits of ID section (the first 4 bits of the *packet number*) have a value 0xF (as depicted in figure 3.10), the whole packet is invalid and should be discarded.

ID	**CRC**	**Data payload**
?F??	????	160 or 240 bytes of garbage data

Figure 3.10: Example of an invalid VoSPI packet.

The procedure to establish synchronization with the camera is following:

1. De-assert CS line and idle SCK for at least 185 ms. This step ensures VoSPI timeout and sets Lepton 3 to a proper state to re-establish synchronization.
2. Assert CS and enable clock. This makes Lepton 3 camera start transmitting packets.
3. Read out the entire packet. Check the ID field of a packet header to recognize an invalid packet.
4. Continue reading packets. The first valid video packet should be transmitted within 10 ms.

There are three violations when reading frames that causes a loss of synchronization:

- Once a transmission of a packet is started, the packet must be completely read out within three line periods, given that CS is not de-asserted nor clock signal disrupted.
- All packets of a segment must be read out before a next segment is available.
- All segments of a frame must be read out before a next frame is available. This also applies to invalid frames.

3.1.4 Controlling the camera module over I^{2}C

Lepton 3 provides a *command and control interface* (CCI) via a two-wire interface almost identical to I^{2}C. The only small difference, when compared to a traditional I^{2}C, is that all transfers must be 16-bit long. All Lepton 3 registers are 16 bits wide. Lepton's CCI address is 0x2A and behaves as a slave device.

Some features, that can be affected by command and control interface are: *AGC mode, AGC settings, telemetry mode and its location, select/upload false color palette, output video format, FFC mode, FFC parameters, radiometry mode, camera current status information, statistics* and many more.

CCI registers

The command and control interface consists of registers through which a host (master device) issues commands—writes data to the camera and retrieves responses. CCI registers can be seen in figure 3.11. Each register is 16 bits wide and alongside 4 control registers, the CCI has up to 16 DATA registers. In addition, the CCI also has two block data registers for big transactions (such as for uploading color palettes), each being 1024 bytes wide.

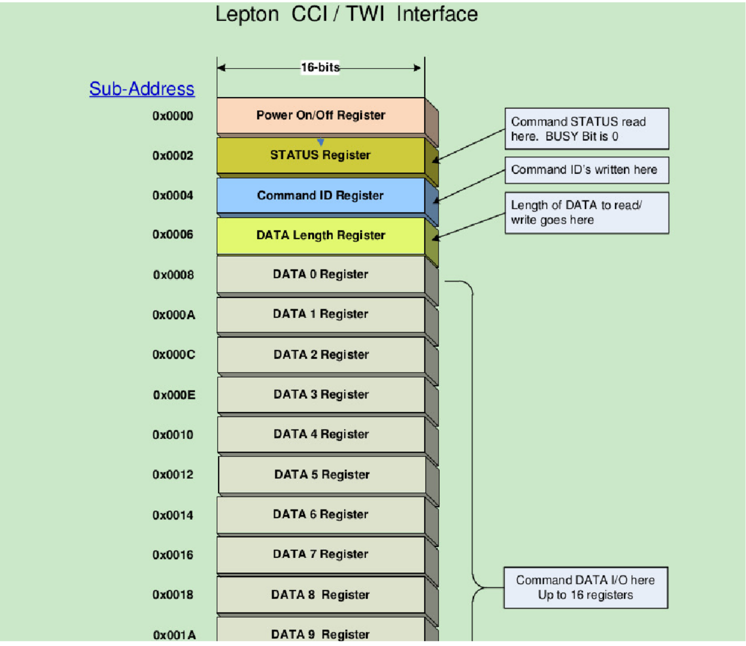


Figure 3.11: Partial Lepton 3 CCI register illustration. (Source: [13][9].)

Power On/Off register is used to wake up the camera after putting it into shutdown mode. We achieve this by writing zero to the register.

Lepton CCI supports three types of commands:

- GET – host requests data from the camera (e.g. retrieves camera statistics).
- SET – host alters configuration of the camera (e.g. sets output video format).
- RUN – host makes camera execute an action (e.g. manually issues FFC calibration process).

Most commands require some data to be transfered as a part of a command either from master host to the camera or the other way around. This data is written into the DATA registers. To signalize how many DATA registers were set—either by the host when executing a *SET* command or by Lepton when extracting information using a *GET* command—we use *DATA length register*. Number written into the DATA length register signifies how many 16-bit DATA registers are in use, and thus valid for current command.

Commands are grouped and identified by a module they affect. Lepton CCI modules are: *AGC, SYS, VID, OEM, RAD*. Camera modules encapsulate properties, attributes and methods of a camera sub-system. For example commands corresponding to the AGC module alter the process of automatic gain control process (AGC), analogically commands in the RAD module control radiometry features and so on.

Command ID register

In order to specify and issue a command over the CCI, a *Command ID register* must be set properly. See command ID register overview in figure 3.12. A command is identified by a *module ID* (bits 8-11) and then by a *command ID* (bits 2-7) with respect to the module specified. For each Lepton 3 module, a unique command ID identifies a command. The OEM bit (bit 14) serves as a safety measure—this bit must be set in order to run commands from OEM and RAD modules. Last thing to set is the command type (bits 0-1).

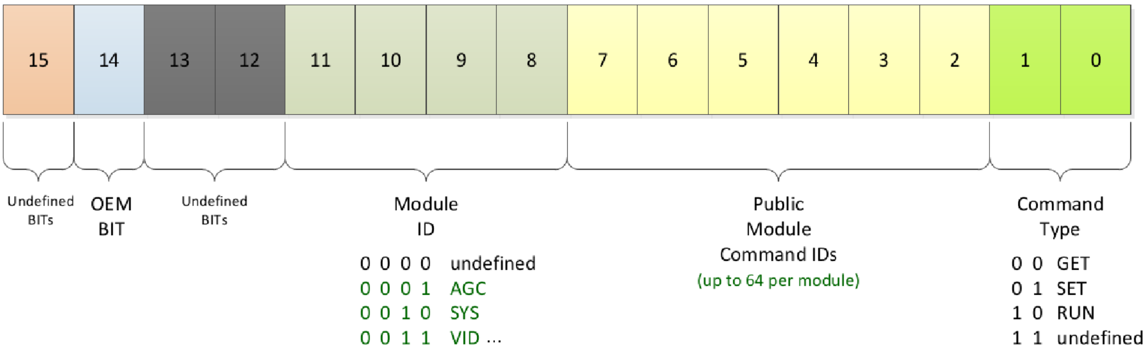


Figure 3.12: Command ID register overview. (Source: [13][17].)

Status register

The Lepton's status register contains information about command and camera boot status. See status register illustrated in figure 3.13.

Boot status bit (bit 2) is set after a successful boot. A master device can monitor this bit to know when the camera has booted up.

After a command is issued to the camera, the busy bit (bit 0) is set automatically. When the busy bit is set, the camera is processing a command and does not accept new commands. When the command is completed, an error response code is written into the high 8 bits (bits 8-15) of the status register and the busy bit is cleared. If an error response code is 0, the command has completed successfully. In any other case, an error has occurred during execution of the command and an error specific code is written to the status register according to the CCI documentation [13][21].

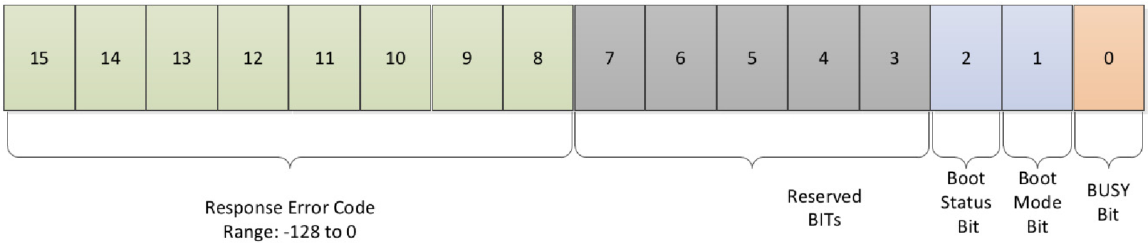


Figure 3.13: Lepton 3 status register overview. (Source: [13][18].)

Writing to the camera

The camera supports write access to any random 16-bit aligned location. The address of a target register to write to is specified at the beginning of the transmission process.

After setting the target address, we can start writing bytes to the specified location. The number of bytes written must be even (16 bits registers and transmission units). In the transmission, the most-significant-byte-first order is used. After writing a byte, the target register address automatically increases by one. This way we perform sequential writings easily. Figure 3.14 illustrates sequential writing to the camera.

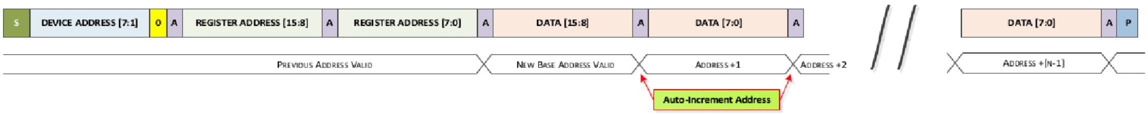


Figure 3.14: Sequential writing to the camera using the CCI. (Source: [13][15].)

A necessary sequence of bytes for a successful write using the CCI is the following:

1. Byte with device address (higher 7 bits) and bit determining a write operation (bit 0 set to `0`).

2. Higher byte of the address to write to.

3. Lower byte of the address to write to.

4. Higher byte of the 16-bit value to be written to the address specified.

5. Lower byte of the 16-bit value to be written to the address specified.

6. Repeat steps 4 and 5 for writing a 16-bit values to the next register (address + 2).

7. End transmission.

Note, that these bytes must be transmitted without interruption. If the transmission is stopped by sending a *stop bit (P)*, it is necessary to resend the device address and the target register address.

Reading from the camera

Performing a read from the camera is a bit more complicated as two transmissions are necessary. First, we have to perform a write operation to set a target address to read from. Secondly we initiates a second transmission—this time a read transmission and we begin to read bytes from the address we have specified. Again after each byte the target address automatically increases. Figure 3.15 illustrates reading from the camera using the CCI.

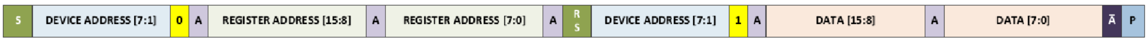


Figure 3.15: Sequential reading from the camera using the CCI. (Source: [13][14].)

A necessary sequence of bytes for a successful read using the CCI is the following:

1. **Write operation – set the target address (same as regular writing, but without writing data to the address).**

   (a) Byte with device address (higher 7 bits) and bit determining a **write** operation (bit 0 set to `0`).
   (b) Higher byte of the address to read from.
   (c) Lower byte of the address to read from.
   (d) End transmission.

2. **Read operation – read from the target address specified**.

   (a) Byte with device address (higher 7 bits) and bit determining a **read** operation (bit 0 set to `1`).
   (b) Higher byte of the 16-bit value read from the address specified.
   (c) Lower byte of the 16-bit value read from the address specified.
   (d) Repeat steps 2 and 3 for reading a 16-bit values from the next register (address + 2).
   (e) End transmission.

Startup sequence

When accessing the CCI of the Lepton camera after power up, the following sequence is recommended:

1. Wait 950 *ms* minimum after power up/master clock applied. (In case of automatic FFC mode, extend the wait time to 5 *s*.)
2. Read the *boot bit* from the status register. If the bit is `0`, the camera has not booted yet, extend wait time and repeat this step.
3. Read the *busy bit* from the status register. If the bit is `1`, keep polling status register until the value becomes `0`.
4. Lepton camera is ready to accept commands.

Process of execution of GET, SET, RUN commands

Issuing a typical **SET** command follows a typical sequence of read write CCI operations:

1. Poll status register until busy bit becomes `0`.
2. Write data to the DATA registers.
3. Write number of data registers written into the data length register.
4. Write command ID to the command ID register.
5. Poll status register until the busy bit becomes `0`.
6. Process a command error response code from the status register.

Issuing a typical **GET** command follows a typical sequence of read write CCI operations:

1. Poll status register until busy bit becomes `0`.

2. Write number of data registers to be read into the data length register.

3. Write command ID to the command ID register.

4. Poll status register until the busy bit becomes `0`.

5. Process a command error response code from the status register.

6. If command completed without any errors, perform a sequential read from the DATA registers.

Issuing a typical **RUN** command follows a typical sequence of read write CCI operations:

1. Poll status register until busy bit becomes `0`.

2. Write command ID to the command ID register.

3. Poll status register until the busy bit becomes `0`.

4. Process a command error response code from the status register.

3.2 Master controlling device Orange Pi$^{\text{TM}}$

In order to be able to communicate with the Lepton 3 module, we have to chose an appropriate low-level computer which is able to communicate over both SPI and I^{2}C interfaces. The computer has to be powerful enough to operate SPI interface with relatively high frequency of about 20 MHz.

While choosing the type of the computer, we must consider the fact that a single video frame coming from the Lepton camera is at least 38 kB in size. This requirement disqualifies many well known boards like the Arduino. An average Arduino board with the fastest crystal-based clock can transceive data over SPI at a rate of 20 MHz, however, it is usually not equipped with more than a few kilobytes of RAM memory. This limitation would make it almost impossible to receive and process video stream from the Lepton 3 camera.

Another aspect is undoubtedly the comfort of developing software support for these computers. After considering all pros and cons, it seemed only reasonable to choose a single-board computer of a Raspberry Pi type—a powerful ARM based developing board with capabilities of communicating over several low-level hardware interfaces providing the comfort of high-level Linux abstraction.

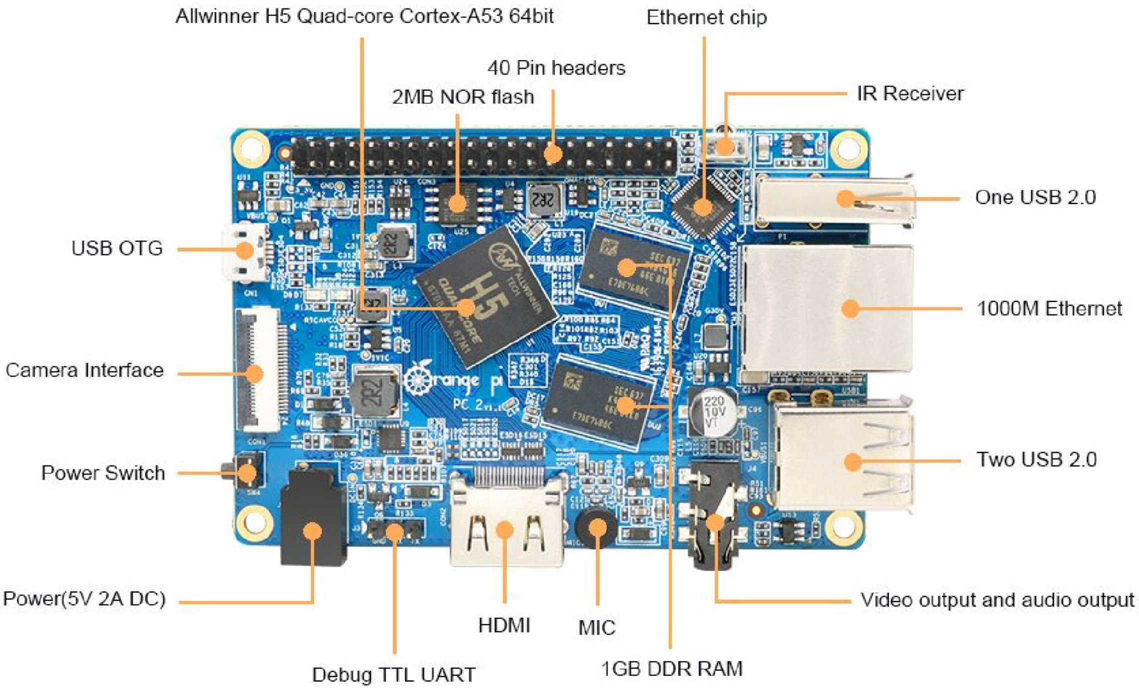


Figure 3.16: Orange Pi PC 2 computer overview. (Source: 6.)

In this project we are using the Orange Pi PC2[6] ARM developing board which is a cheaper however slightly more powerful brother of the traditional Raspberry Pi. The Orange Pi is running quad-core 64-bit Cortex A53 CPU with 1 *GB* of RAM, and as usual with Raspberry Pi-like boards, it has 40 general I/O pins connected to various hardware modules. Full specifications of the developing board Orange Pi PC2 can be found in table 3.3. Specific information about the computer comes from [35].

In this project we are mostly interested in SPI and I^{2}C hardware modules, that enable us to receive video frames from, as well as control the thermal camera directly from a high-level Linux system running on the board.

The Orange Pi PC2 board also provides connectivity to three UART[7], two TWI[8] and one SPI[9] hardware modules.

The CPU has enough processing power to maintain smooth operation without delays, which turned out to be crucial for maintaining synchronization with the camera module when transferring video frames.

[6] Orange Pi PC 2 official product page http://www.orangepi.org/orangepipc2/

[7] UART – Universal Asynchronous Receiver-Transmitter

[8] TWI – Two wire interface – identical to I^{2}C (Inter-integrated circuit)

[9] SPI – Serial peripheral interface bus

Manufacteur	Shenzhen Xunlong Software CO., Limited
System on a chip CPU GPU	Allwinner H5 (sun50i) Quad-Core Cortex-A53 ARM @ 1.3 GHz Mali450 MP4 GPU
DRAM	1 GB DDR3
Power	DC 5 V @ 2 A
Video Audio	HDMI 1.4, CVBS 3.5 mm Jack, HDMI, mic
Network Storage USB	10/100/1000 $Mbps$ ethernet (Realtek RTL8211E) micro SD, 16 Mb NOR flash 3 × USB 2.0 host, 1 × USB 2.0 OTG

Table 3.3: Orange Pi PC2 specifications. [35]

There are several operating system officially supported by the Orange Pi. Official images include Android 5.1, Ubuntu, Debian, Raspbian as well as Banana Pi. Next to official images, we can also chose an unofficial Linux operating system Armbian[10]. After exploring several images, we have decided to go with the unofficial Armbian operating system, even though only an experimental build was available. Three main aspects were in favor for the Armbian OS:

- **Stability** – some images seem to by unstable on our board, especially the Android OS and unfortunately also Raspbian.
- **Community** – Armbian has plenty active forum users and is reasonably well documented.
- **Hardware interface access** – from all available and working system images, Armbian image seemed to have the best support for accessing low-level interfaces on the board.

In this project we are using the Linux Armbian 5.37 image based on desktop version of the Ubuntu Xenial 16.04.

3.3 Enabling Orange Pi hardware interfaces

Most of the times, operating systems, that are available for Raspberry Pi-like boards, come with hardware interfaces like I^{2}C or SPI disabled by default and I/O pins, that are connected to them, are used as standard general purpose I/O pins. One can access these hardware modules on Linux running machines using so called devices in the `/dev/*` directory and control them with standard system calls like `ioctl()`. If there is not a device for a particular hardware module listed in the `/dev/*` directory, then the module is disabled or not available at all.

Enabling these modules happens not to be a simple task, since there is no officially documented straight forward procedure that could be generalized to every system and board. Steps required to enable them vary not only from one operating system to another, but also from one board to another. Bits and pieces, about how to enable the hardware

[10] Armbian – Linux for ARM development boards https://www.armbian.com/orange-pi-pc2/

interfaces for Armbian operating system, had to be collected throughout multiple Armbian forums. [7] Some well-educated guesswork with the help of the official documentation [6] was also necessary.

As mentioned in previous paragraphs—we use so called devices to access hardware interfaces (mapping of their hardware modules) from the Linux operating system. A *device tree* (DT) is a way of describing the hardware module configuration to the operating system kernel. In order to enable hardware modules of the SoC[11] we need to modify the device tree.

For Armbian, this can be accomplished with help of so called *device-tree overlays*—compiled configuration files loaded at startup and used by the kernel to create requested hardware module pin mappings. The Armbian OS comes with several DT overlays already included. The first step would then be to try to use the provided DT overlay at startup. This can be achieved by modifying the `/boot/armbianEnv.txt` configuration file, respectively, by adding the DT `spi-spidev` overlay to the used overlays as well as specifying spidev parameters:

- `param_spidev_spi_bus` – ID of the SPI interface according to the pinout of the board.
- `param_spidev_max_freq` – setting a hard limit for the maximum speed of the SPI interface module.

The edited section of the boot configuration illustrating usage of `spi-spidev` and `i2c0` DT overlay can be seen in listing 1.

```
overlays=spi-spidev i2c0
param_spidev_spi_bus=0
param_spidev_max_freq=100000000
```

Listing 1: Device-tree overlay configuration file `/boot/armbianEnv.txt` loaded at startup.

These lines ensure, that the precompiled device-tree overlay maps the `/dev/spidev0.0` device into the system DT.

The first zero of `0.0` in the device name corresponds to an SPI bus ID. This ID depends on the pinout of a specific board and the SPI module used. The ID usually ranges from 0 to at most 3. The second zero corresponds to a *chip select* (CS) pin number. To every SPI interface on the Orange Pi board, we can connect usually up to two devices. Devices would share the same SCK, MOSI, MISO lines, however each device would have its own CS line (separate hardware pins). In case there is the possibility of connecting two devices to the same SPI interface, the CS number is either `0` or `1`.

To summarize, after enabling the SPI 0 interface, we should be able to see two devices available in the device tree: `/dev/spidev0.0` and `/dev/spidev0.1`—one of them using CS0 line, the other CS1 line.

In case there is still no such device visible in the DT after rebooting, that could mean that the interface is not available or that the provided precompiled DT overlay `spi-spidev` is incorrectly configured. In the second case, we may try to take the longer way of decompiling the provided overlay first, reconfiguring it, compiling it back, and only then will the device hopefully appear in the DT.

[11] SOC – System on a chip—integrated circuit integrating computer components with other electronics

For every board there is a default hardware module configuration file stored in the `/boot/dtb/allwinner/` directory starting with a SoC code name. For Allwinner H5 SoC, the prefix is *sun50i-h5*. The default configuration file for the Orange Pi PC2 is therefore `sun50i-h5-orangepi-pc2.dtb`. This file contains default mapping for every hardware module on the board for a specific SoC. This might include modules controlling for example: *sound, clocks, SPI, HDMI, DMA, LCD controller, memory, USB, ethernet etc.*

After configuration is loaded from this base file at startup, we override parts of configuration by applying DT overlays specified in the `/boot/armbianEnv.txt` file. The DT overlay files are stored in the `/boot/dtb/allwinner/overlay` directory. The overlay file name consists of SoC code name and overlay name. For already mentioned `spi-spidev` overlay the file name is `sun50i-h5-spi-spidev.dtbo`.

The `.dtb` and `.dtbo` extensions stand for *device tree blob* or *binary*. These files contain a database, that represents the hardware components on a given board. If we want to view or edit a device tree blob, firstly, we must decompile it into the *device tree source* form (extension `.dts`), which is basically a form of a text file. We can decompile DT blob files using so called *device tree compiler* – `dtc`. The compiler is usually provided with the embedded Linux distribution. [15]

When troubleshooting the problem with a missing device, at first, we might take a look into the main base configuration file, whether a configuration for a given hardware module is provided. We decompile the file using the following command:

```
$> dtc -I dtb -O dts -o sun50i-h5-orangepi-pc2.dts
 ↪  sun50i-h5-orangepi-pc2.dtb
```

Listing 2: Command used to decompile device tree blob using device tree compiler.

In the file we can find sections belonging to SPI and I^{2}C modules. See snippet from configuration file in listing 3. In this file, nothing looks suspicious, as instead, everything looks configured properly—names of the pins match and so on.

```
spi0 {
        pins = "PC0", "PC1", "PC2", "PC3";
        function = "spi0";
        Linux,phandle = <0x20>;
        phandle = <0x20>;
      };

i2c0 {
        pins = "PA11", "PA12";
        function = "i2c0";
        Linux,phandle = <0x26>;
        phandle = <0x26>;
      };
```

Listing 3: Snippets of SPI and I^{2}C module sections in the base configuration file.

Next step is to check the overlay DT blob for each module, we want to troubleshoot. We decompile it the same way as we have decompiled the base configuration file. Contents of the decompiled DT blob of the `spi-spidev` overlay can be seen in listing 4.

```
1  compatible = "allwinner,sun50i-h5";
2
3    fragment@0 {
4      target-path = "/aliases";
5      __overlay__ {
6        spi0 = "/soc/spi@01c68000";
7      };
8    };
9
10   fragment@1 {
11     target = <0xffffffff>;
12     __overlay__ {
13       #address-cells = <0x1>;
14       #size-cells = <0x0>;
15
16       spidev {
17         compatible = "spidev";
18         status = "disabled";
19         reg = <0x0>;
20         spi-max-frequency = <0xf4240>;
21       };
22     };
23   };
24
25   __fixups__ {
26     spi0 = "/fragment@1:target:0";
27   };
```

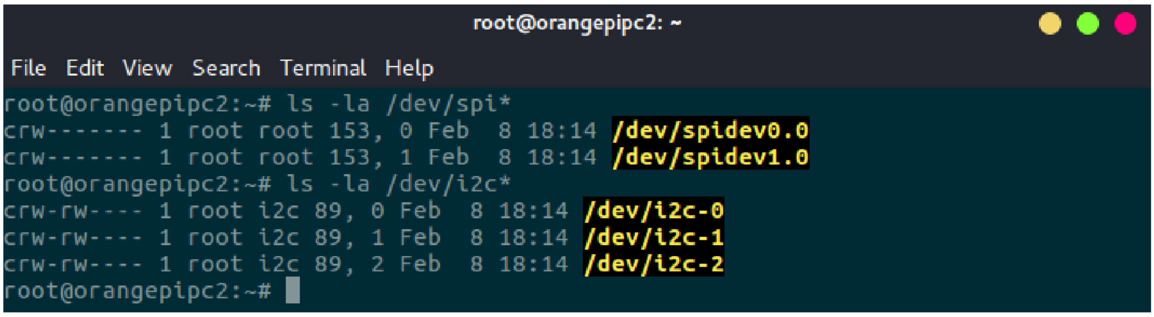
Listing 4: Configuration snippet of SPI 0 from `spi-spidev` overlay DT blob.

As we can notice on the line 18 of the SPI 0 module configuration (listing 4), the module is set to be disabled. In our case, this was causing the problem of the device not being visible in the system DT. What we need to do, is to change the `status` attribute to `"okay"`, recompile the device-tree Source: file back to a binary blob using command depicted in listing 5 and replace the old blob file. (Or configure `armbianEnv.txt` to use the new DT overlay.)

```
$> dtc -I dts -O dtb -o sun50i-h5-spi-spidev.dtbo
 ↪  sun50i-h5-spi-spidev.dts
```

Listing 5: Command used to decompile device tree blob using device tree compiler.

Repeat these steps analogically for I^2C or any other incorrectly configured device. After repairing the configuration of the device tree and rebooting the board, we should be able to see requested devices in the `/dev/*` directory. (Figure 3.17 illustrating correct configuration of SPI 0, 1 and I^{2}C 0, 1, 2 devices.) [5]



```
root@orangepipc2:~# ls -la /dev/spi*
crw------- 1 root root 153, 0 Feb  8 18:14 /dev/spidev0.0
crw------- 1 root root 153, 1 Feb  8 18:14 /dev/spidev1.0
root@orangepipc2:~# ls -la /dev/i2c*
crw-rw---- 1 root i2c 89, 0 Feb  8 18:14 /dev/i2c-0
crw-rw---- 1 root i2c 89, 1 Feb  8 18:14 /dev/i2c-1
crw-rw---- 1 root i2c 89, 2 Feb  8 18:14 /dev/i2c-2
root@orangepipc2:~# 
```

Figure 3.17: Correctly configured SPI 0, 1 and I^{2}C 0, 1, 2 devices.

Chapter 4

Lepton 3 software library *v4l2lepton3*

Once the proper functionality of both I^{2}C and SPI interfaces has been confirmed using an Arduino board as a loop-back device, next step is to develop a software layer, that would enable us to control the camera and capture video frames. This chapter describes a software library `v4l2lepton3` which has been created specifically for purposes of this thesis. The library is available publicly in a git repository[12]. It consists of two parts:

- Frame capture tool written in fast C++ with dual segment buffering.
- Python control script allowing for sending commands to the camera.

4.1 Receiving video frames over VoSPI

Writing a custom software for receiving video from the camera would not have to be necessary, if we could find a decent, working, ready made library solution. Unfortunately, none of the found libraries would work for the newer Lepton 3 camera, because either they would not be finished or would be programmed for the previous version of the camera—Lepton 2, whose VoSPI protocol is not compatible with the camera we are using. In addition, many existing libraries are pointing out problems with losing synchronization with the camera. After few unsuccessful attempts to receive frames using several libraries for example from the Groupgets FLIR solutions git repository[13] [14], we came to realization, that a custom software is required. Even though mentioned libraries did not work with our camera, they served as an inspiration for our custom software.

4.1.1 Single frame capture using Python

The most friendly way to access the SPI interface directly from Linux system would be from Python environment. There are plenty of Python libraries available for communication over SPI interface. For our distribution, we are using the `SPI-Py` library[15]. Simply by using a single import and a setup call to configure SPI mode, speed and SPI device, we can start transferring bytes. Example transmission code is depicted in listing 6.

[12] v4l2lepton3 library official git repository `https://gitlab.com/CharvN/v4l2lepton3`

[13] Groupgets LeptonModule library git repository `https://github.com/groupgets/LeptonModule`.

[14] Groupgets Pylepton library git repository `https://github.com/groupgets/pylepton`.

[15] SPI-Py official git repository `https://github.com/lthiery/SPI-Py`

```python
import spi

spi.openSPI(mode=3, device="/dev/spidev1.0", speed=15000000)
tx_bytes = tuple(0 for x in range(164))
rx_bytes = spi.transfer(tx_bytes)
```

Listing 6: Example code transceiving a VoSPI packet (164 bytes) using Python.

The first step was to capture a stream of raw bytes and try to match the content with information of the VoSPI protocol provided in the Lepton 3 documentation. After grouping captured bytes (depicted in listing 7) by 164 bytes (VoSPI packet in the raw Y14 format), we can immediately recognize a pattern.

According to the documentation, if the highest 4 bits of the *packet number* column in the VoSPI header have a value 0xF, the packet is invalid. We can see the beginning of a VoSPI segment transmission in listing 7 on the line 4 and the last VoSPI packet of a segment (packet number 59) on the line 10.

```
 1  ...
 2  ff ff ff ff 00 00 00 00 ...
 3  ff ff ff ff 00 00 00 00 ...
 4  70 00 7e 72 1f 0b 1f 05 ...
 5  00 01 c3 3b 1e f4 1e f5 ...
 6  00 02 fb cf 1f 00 1e fe ...
 7  00 03 69 6e 1e ed 1e f2 ...
 8  ...
 9  00 3a 0a 30 1f 3d 1f 38 ...
10  00 3b 67 bc 1f 46 1f 4a ...
11  ff ff ff ff 00 00 00 00 ...
12  ...
```

Listing 7: Raw bytes of VoSPI packet stream in hexadecimal format. Beginning and ending of a VoSPI packet visible on line 4 and 10.

Listing 7 illustrates an ideal situation, when the host manages to pull all packets of the segment. At first, this was not our case. In fact, we were able to receive only about one third of the total number of packets. In the middle of the transmission the camera would stop transmitting valid packets, send a few invalid ones and then reset the transmission from packet 0. As mentioned in section 3.1.3, a reasonable explanation would be insufficient speed.

In order to ensure maximum speed during transmission, we have undertaken the following measures:

- Used short wiring—no longer than 15 *cm*.
- Soldered wire connections directly to the Orange Pi board.
- Used additional capacitor close to the Lepton camera.
- Installed extra pull-up resistor on the SCK SPI line.

- Cranked up the SPI speed to 20 MHz.
- Wrote as efficient Python code as possible to ensure minimal delays.

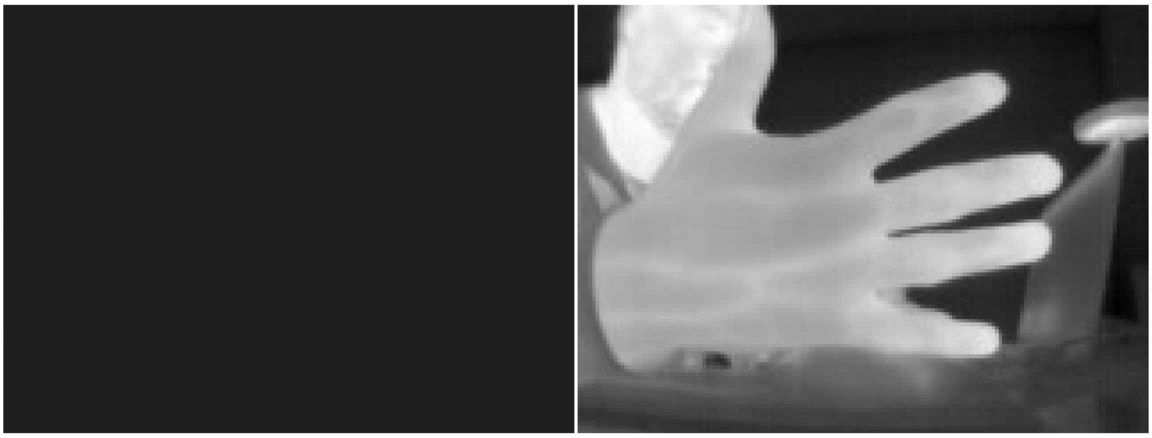
After these adjustments, we were able to read the entire segment without losing synchronization. We could proceed and construct an algorithm, that strictly follows the VoSPI protocol and captures a whole frame.

Based on the information from the VoSPI protocol documentation, we were able to construct an algorithm capturing a single frame. A Python code snippet illustrating the algorithm is depicted in listing 8.

```python
def capture_frame():
  frame = list()
  segment = list()
  tx_bytes = tuple(0 for x in range(164)) # 164 bytes - raw Y14 format

  segment_number = 1
  while(segment_number < 5): #  for every segment number (1,2,3,4)
    packet = spi.transfer(tx_bytes)

    if (packet[0] & 0x0F) == 0xF: #  if packet invalid
      #  recapture segment
      segment = list()
      continue

    segment.append(packet)

    if (packet[1] == 59): #  if segment completed (telemetry disabled)
      if (segment[20][0] >> 4) == segment_number: #  if segment valid and in order
        #  continue capturing next segment
        frame.append(segment)
        segment_number += 1
      segment = list()

  return frame
```

Listing 8: Python algorithm for capturing a VoSPI frame.

This way, we are able to capture a single frame at a time. See example image captured by the camera in figure 4.1. The image is captured in raw Y14 video format mode. The left image (a) displays pure data, the image on the right side (b) is identical to the one on the left side, only with a linear AGC applied externally for viewing detail enhancement. As you can see, there is almost nothing visible to the human eye on the image without AGC applied, as the pixel values are very close to each other (scene temperature does not cover the whole Lepton temperature range).



(a) Raw Y14 mode, no adjustments, pure data

(b) Raw Y14 mode, linear AGC applied

Figure 4.1: Image captured by by Lepton 3 camera. Comparison between raw data and when AGC is used.

Despite the fact, that now we are able to capture a single frame, maintaining synchronization with the camera and capture frames in a continuous manner seems to be a quite difficult task. The root of this issue is the nature of Python language. Running the Python interpreter on an ARM board, making system calls to transfer 164 bytes 60 times for each segment gets simply **too slow**, even when running the SPI transfer at maximum speed of 20 MHz, which should give us some time reserve.

4.1.2 Capturing frames using *v4l2lepton3*

This speed issue can be solved only by implementing the capture software in a faster language. In this subsection, the first part of the v4l2lepton3 library is described, which implements frame capture in fast C++.

Dual segment buffering

Just by itself, switching to C++ code improves the performance significantly. Furthermore, we are utilizing mulithreading to minimize system call delays. The technique is an analogy for dual frame buffering, however in this case, we shall call it *dual segment buffering* as we have two buffers for segments, not frames.

One thread performs system calls, receives VoSPI packets and writes them into a segment buffer. After the segment transmission is completed, the two segment buffers are swapped, the first thread continues capturing new packets—now into the second buffer—and the second thread performs post-processing for every packet in the finished segment, and copies the segment to the correct location in the frame buffer.

Using these techniques, the library does not lose synchronization with the camera while transferring video frames, and thus allows for continuous operation.

Packet and frame post-processing

Packet post-processing, next to stripping VoSPI packet header, might include reordering bytes for raw mode, as bytes are transfered in the opposite order (most significant byte first). When the last fourth segment is received, and therefore the whole frame completed, frame post-processing is performed. This might include performing a custom AGC algorithm or applying custom color palette. Both packet and frame post-processing routines are implemented as `virtual` methods and can be overridden for custom requirements.

In our implementation of packet post-processing, we strip out the packet header, and only for raw mode, we change order of every two bytes.

In the frame post-process method, there is no action performed as we need unaltered data for raw mode. For RGB888 video format, we preferably use camera's internal AGC with preloaded color look-up table.

Virtual video loopback device

When the transmission of the whole frame is completed, we need a way to forward this frame to the user. The `v4l2lepton3` library works in hand with the kernel module `v4l2loopback` [16], which allows us to create `V4L2`[17] virtual loopback devices in the Linux system.

Our library writes frames from the Lepton 3 camera module into a virtual video device, which then provides accessibility to the video stream for all standard Linux tools like **`ffmpeg`**, **`gstream`**, **`vlc`** and so on. The video can also be streamed over a network. Because the virtual video device behaves in the same way as any standard real-time video capture device in Linux (web cameras, TV tuners), it is also possible to simply extract video frames from the device using the OpenCV library [30]. Code extracting an image frame from a virtual video device 0 using Python-ported OpenCV can be seen in listing 9.

```python
import cv2
cap = cv2.VideoCapture(0)
ret, img = cap.read()
```

Listing 9: Example code for extracting an image frame from a virtual video device `0` using Python-ported OpenCV library.

In order to get the `v4l2loopback` kernel module up and running, it is necessary to compile it directly on the machine. We can acquire the kernel module a using package handling tool `apt-get`.There is, however, one requirement, that needs to be fulfilled before setting up the kernel module. In order to build the module, there must be *kernel headers* installed in the system and they must match the Linux kernel which you want to use it with.

For Armbian, finding and installing the proper kernel headers is a bit tricky and require some trial and error approach. We can find currently used kernel release name by running a `uname -r` command. What also might help, while figuring out the proper kernel header version to download, is to browse official Armbian remote `apt` repository available at `apt.armbian.com`.

For our board and Armbian version, we have managed to successfully install Linux kernel headers by using the command:

[16] `v4l2loopback` virtual video loopback device kernel module for Linux official git repository `https://github.com/umlaeute/v4l2loopback`

[17] V4L2 – Video 4 Linux 2

```
$> sudo apt-get install linux-headers-next-sunxi64
```

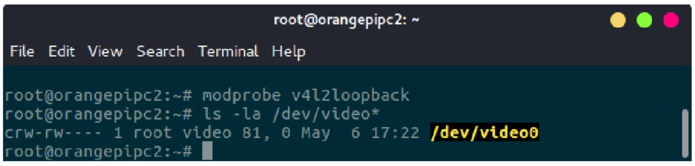
Listing 10: Command used to download and install Linux kernel headers for our version of Armbian and Orange Pi.

After acquiring kernel headers, the `v4l2loopback` kernel module can be downloaded, build and loaded with its control tool by executing commands depicted in listing 11. While loading the module with `modprobe` command, we can specify number of virtual devices, their ID and other parameters. More options available in the description of `v4l2loopback` available in the official git repository 16.

```
$> sudo apt-get v4l2loopback-dkms
$> sudo apt-get v4l2loopback-utils
$>
$> sudo modprobe v4l2loopback
```

Listing 11: Command used to download, build and load the `v4l2loopback` kernel module and its control tool.

When the kernel module is loaded, we should be able to see a new video device available in the system device tree, as illustrated in figure 4.2.



```
root@orangepipc2:~# modprobe v4l2loopback
root@orangepipc2:~# ls -la /dev/video*
crw-rw---- 1 root video 81, 0 May  6 17:22 /dev/video0
root@orangepipc2:~#
```

Figure 4.2: Correctly configured virtual video loopback device `0`.

Using v4l2lepton3 library to capture video

We can build the library using the standard `make` command, as the `Makefile` file is provided, or we can use `CMakeLists.txt`.

The `v4l2lepton3` executable can be run with following parameters:

- `-s --spi <SPI device name>` name of a SPI device e.g. `/dev/spidev1.0`
- `-v --video <video device name>` name of a virtual video loopback device e.g. `/dev/video0`
- `-f --format <format name>` lepton3 format output video format. Possible choices are:
  - `raw` assumes, that the Lepton 3 VoSPI packets are 164 bytes long. Sets the video device to Y16[18] format.

[18] Y16 and Y14 formats are identical, only Y14 does not use the highest two bits (14-bit resolution)

- `rgb` assumes, that the Lepton 3 VoSPI packets are 244 bytes long. Sets the video device to RGB24 format. See example image extracted using this method in figure 4.3.
- `raw_over_rgb` **[Experimental]** assumes, that the Lepton 3 VoSPI packets are 164 bytes long in raw format (resolution 160×120×2 bytes) and sends the data to video device configured to RGB24 format (resolution 160×80×3 bytes) for purposes of raw frame extraction to the Python-ported OpenCV, as this version is unable to read frames from a video device without transforming the image to the RGB24 format and thus losing the original raw flux values present in the Y14 format. Unfortunately the data passed through are slightly altered (probably due to anti-aliasing or frame smoothing), making the extracted frame unusable for raw value analysis (after reconstructing back to Y16) due to unwanted artifacts mostly among the edges in the image. Such artifacts are visible in figure 4.4.

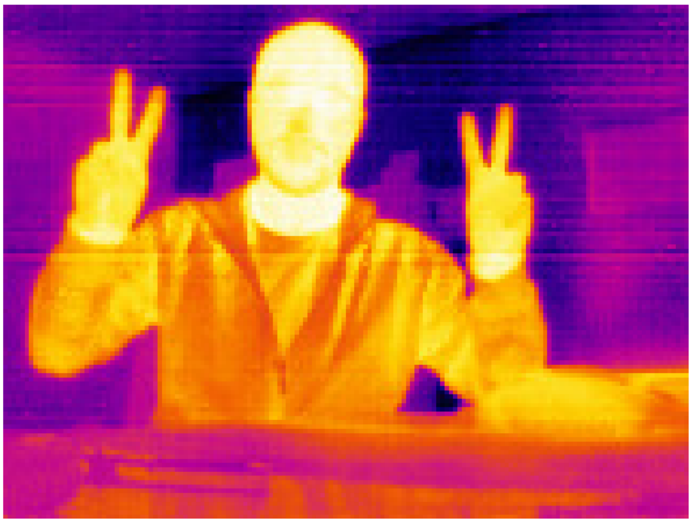


Figure 4.3: Example image captured from the virtual video device using RGB888 method.

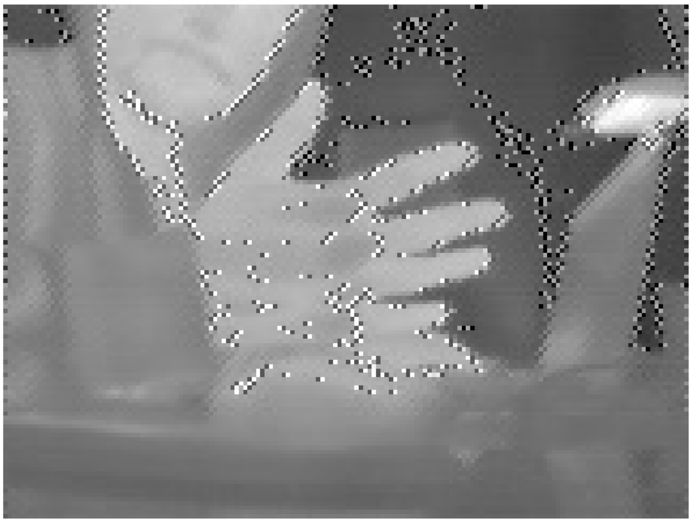


Figure 4.4: Illustration of artifacts after reconstruction back to Y16 format when using experimental method `raw_over_rgb`.

4.2 Controlling the camera over I^{2}C

Second part of the library, dealing with controlling the Lepton 3 camera module, is contained in a single `lepton3.py` script, that allows for sending commands to the camera. The script uses the `smbus2` Python library[19] to access I^{2}C device in the system DT. [1] We can acquire the library using the `pip` tool for Python 3:

```
$> sudo pip3 install smbus2
```

Listing 12: Command used to acquire Python3 library `smbus2` for accessing the I^{2}C interface.

Commands sent to the camera must **strictly** follow the CCI protocol as described in the section 3.1.4. The most challenging part is to send all the bits in correct order—especially correct placement of stop bits and reinitializing transfer from the high-level Python library calls.

The `lepton3.py` script contains prepared infrastructure for sequential reading/writing and setting/reading random registers in the camera as described in detail the section 3.1.4. This infrastructure is implemented in the `Lepton3Control` class and consists mainly of three methods:

- `_read_register(self, register_address, number_of_bytes)` – reads requested number of bytes (must be even) from a register specified by its address.
- `_write_register(self, register_address, data)` – writes `data` to a register specified by its address. `data` can be either a 16-bit integer or an array of bytes (must be even).

[19] Official `smbus2` Python I^{2}C library git repository https://github.com/kplindegaard/smbus2

- `_command(self, module, command_id, command_type, data)` – issues a command with `command_id` in a `module` of a type `SET/GET/RUN` specified by the `command_type`. The `data` depends on the command type—can be `None` for `RUN` command, must be an even integer for `GET` command (number of bytes requested) and must be list of bytes for a `SET` command.

This way, it is fairly easy to add not implemented commands directly from the documentation [13] with help of command templates provided. A command example can be seen in listing 13—the function issues a `SET` command in the `OEM` module with command id `OEM_FORMAT` and the data being set is 16-bit value `0x0007`.

```
def format_raw16(self):
  return self._command(Lepton3Control.OEM, Lepton3Control.OEM_FORMAT,
↪  Lepton3Control.SET, [0x00, 0x07])
```

Listing 13: Command example for setting the video format to raw mode.

At this moment, the control script implements the following commands:

- agc_enable
- agc_disable
- agc_enable_calculation
- agc_disable_calculation
- ping
- colormap_grayscale
- colormap_fusion
- ffc_radiometry
- format_raw16
- format_rgb
- radiometry_enable
- radiometry_disable
- format_get

Usage of the Python control script is described in listing 14. Necessary argument `device_number` corresponds to a I^{2}C device number in the Orange Pi's device tree. For I^{2}C device `/dev/i2c-0` the number would be `0`.

```
$> python3 lepton3.py <device_number> <command>
```

Listing 14: Usage prescription of the `lepton3.py` control script.

Chapter 5

Image processing and object detection

This section describes a method of detecting people from a single raw thermal image and a technique for locating people used in this project.

The software part of this chapter, dealing with detecting and locating people from a single raw thermal image in a known environment, is available along with this thesis in a public git repository[20]. The repository contains Python module `ThermoDetection` with helper class `ThermoHelper` encapsulating helper functions for detecting people. The module also contains class `Scene`, which provides an abstraction over a known environment. We use this environment model to display mutual positions of the camera, scene borders and all detected objectes. Along with the Python module, the repository contains Python scripts used for testing, experimenting, calibrating detection methods and plotting figures in this thesis.

Steps describing the process of detecting people from a single thermal image can be summarized by the following enumeration:

1. Convert raw pixel values into real temperature.
2. Apply human-body temperature filter.
3. Increase dynamic range of the image using custom linear AGC.
4. Apply adaptive binary mean thresholding to the image resulting in a black and white image.
5. Reduce noise using morphological transformations *erosion* and *dilation*.
6. Find contours of compact consistent shapes of warm objects.
7. Apply border boxes around found contours.
8. Filter living being object by border box area threshold.
9. Apply border boxes enclosing found objects on the original image.

[20] *thermo-person-detection* git repository `https://gitlab.com/CharvN/thermo-person-detection` – the second part of the software published with this thesis.

5.1 Pixel values to temperature

Once we obtain a single image in raw format (Y14 gray scale), whose pixels have unaltered values of flux coming to the sensor, it is reasonable to convert the values into real temperature. The raw pixel values range from `0` to `16383`. In order to extract the real pixel temperature, we must create a mapping function from flux to temperature.

Although the Lepton 3 module has radiometry support, it is not in fact true radiometry in a sense, that we would be able to receive temperature readings from the sensor directly, as it is possible with different models of the Lepton series. Lepton 3's radiometry mode only compensates for the change in internal and ambient temperature difference, so that its raw output does not get affected by this change of temperature and the camera always produces the same data with respect to the incoming infrared flux, as described in the section 3.1.2.

For our type of camera, it is convenient to use an external spot thermometer device for calibration purposes such as the MLX90614[21]. Using this device, we can create a precise mapping function between flux values produced by Lepton module and actual temperature.

In this project, since we do not require extreme temperature precision, we can empirically approximate the mapping with a linear function as depicted in equation 5.1.

$$t = 0.016632 * p - 50.93347 + t_a \tag{5.1}$$

Figure 5.1: Simple linear mapping function from raw flux pixel values into real temperature, where t is result temperature, p is a raw pixel value to be converted and t_a is the ambient temperature.

5.2 Image processing using OpenCV

Image processing techniques play a very important role in the detection method we propose. Most of them are based on the *OpenCV*[22]library (Python ported) as it is a very powerful tool in the area of image processing alongside the *Numpy* library[23] for Python. The theory and general techniques of image processing and scene reconstruction are sourced from [34], [32]. The knowledge about possibilities regarding image processing and scene reconstruction provided by the library were gained mostly from [8], [31], [23], [21].

This subsection describes parts of the image processing technique, that we use in this project to detect people in stationary thermal images.

5.2.1 Temperature filtering

Since we already have an image with raw flux values and temperature mapping function, we convert pixels into real temperature and create a binary mask for all pixels with a value in human temperature range using the `numpy.inRange()` function.

Since we are using only an approximated temperature mapping function, there is no point in stating the minimum and maximum human temperature value, as these values highly depend on the chosen temperature mapping function. The edge values, as well as the mapping function, were determined empirically.

[21] Melexis MLX90614 spot infrared thermometer https://www.melexis.com/en/product/MLX90614/Digital-Plug-Play-Infrared-Thermometer-TO-Can

[22] Open Source: Computer Vision Library https://opencv.org/

[23] Numpy scientific computing package for Python http://www.numpy.org/

Before applying the binary mask on the captured image, we need to provide values for image areas outside the human temperature range. Zero value can not be used, because this would negatively affect normalization process later on. (We would be wasting dynamic range of the image.)

Using the inverted binary mask, we fill areas outside human temperature range with the higher value of the following two: lower human temperature range limit and global minimum pixel value. This way we can merge the original image after applying the temperature mask with the second image containing minimal values in areas outside the specified human temperature range.

In summary, we replace all areas of the image with values outside the specified human temperature range by the lowest reasonable value, so that we do not decrease dynamic range of the image.

5.2.2 Normalization

Since this method is using primarily object or shape detection, algorithm it is reasonable to perform some kind of normalization. In this case, we would use in fact a method identical to the linear AGC. We find global minimum and maximum pixel value in the image and map all pixels linearly between the lowest and the highest possible value.

Furthermore we are also lowering the resolution from 14 to 8 bits. Effects of applying the linear AGC on a raw image are depicted in figure 4.1.

5.2.3 Adaptive binary mean thresholding

In the next step, we utilize OpenCV functionality and we perform *adaptive binary mean thresholding*. The `cv2.adaptiveThreshold()` takes two parameters and they were empirically set to `95` for the `block size` and `-30` for `constant difference`. As a result, we get a binary image (black and white) displaying white compact shapes of objects in human body temperature range on a black background. Example image after applying adaptive mean thresholding can be seen in figure 5.2.

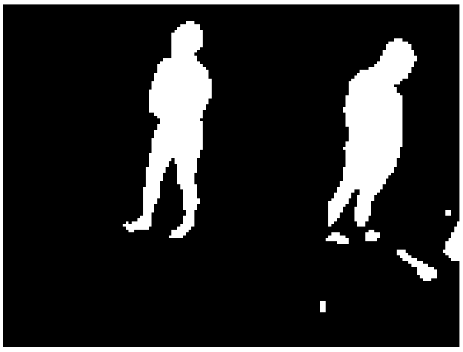


Figure 5.2: Example of thermal binary image after applying the adaptive mean thresholding.

The adaptive mean thresholding has been chosen empirically as it outperformed other thresholding techniques for our use case. The regular thresholding with hardcoded or evaluated threshold performs significantly worse than adaptive thresholding due to its nature—single threshold value. In regular imagery, the single value threshold algorithm yields unsatisfying results, when the light conditions are not the same across the whole image.

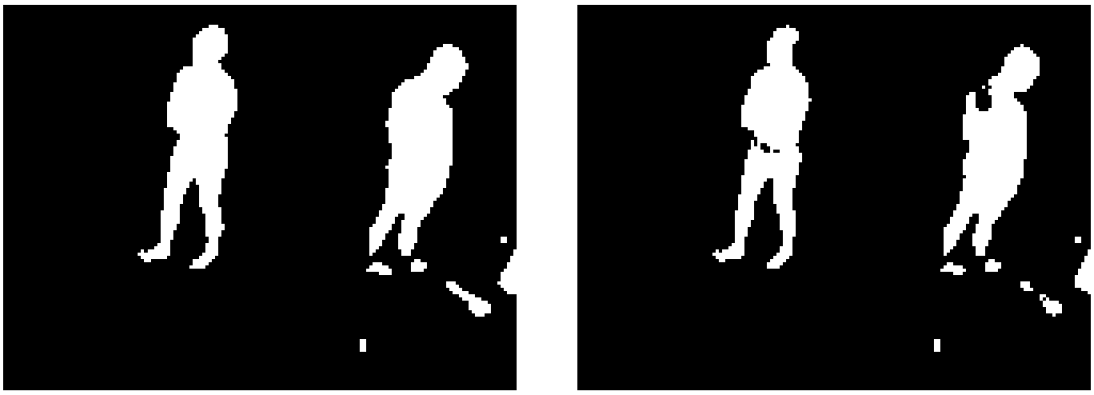
For thermal imagery, this would mean, that objects slightly colder than other, but still in the human temperature range, would not be detected perfectly. Single value threshold algorithms seemed very hard to tune properly. In case of changing ambient temperature of the scene, they do not yield convincing stable results.

The adaptive thresholding differs from the standard thresholding. In the standard thresholding, a single threshold value is used for the entire image, however, adaptive thresholding splits the image into *blocks*—squares of `block size`—and for every block, a local threshold is calculated. The OpenCV functions have, apart from `block size`. also a `constant difference` or `C` parameter. This parameter is simply a constant, that is subtracted from the calculated threshold before applying on the image. In addition, there are two ways to calculate the threshold for every block:

- **Adaptive mean thresholding** – the threshold of a block is a mean of surrounding block values.

- **Adaptive Gaussian thresholding** – the threshold of a block is a weighted sum of surrounding block values, where weights are a Gaussian window.

From the adaptive family of OpenCV's thresholding algorithms, we have also tried the *adaptive Gaussian thresholding*. This algorithm performed almost the same as the one we have chosen, however, it seemed little bit more noisy—warm objects on a colder background were not as uniform and compact as with the mean thresholding. Comparison of mean and Gaussian thresholding with the same block size and constant C can be seen in figure 5.3.



(a) Adaptive mean thresholding

(b) Adaptive Gaussian thresholding.

Figure 5.3: Comparison between thermal binary images after applying different adaptive thresholding with the same parameters introducing different amount of noise.

5.2.4 Noise removal

In order to remove noise in the image, we use morphological transformations, which are also available in the OpenCV library. Two basic operators in the area of mathematical morphology are:

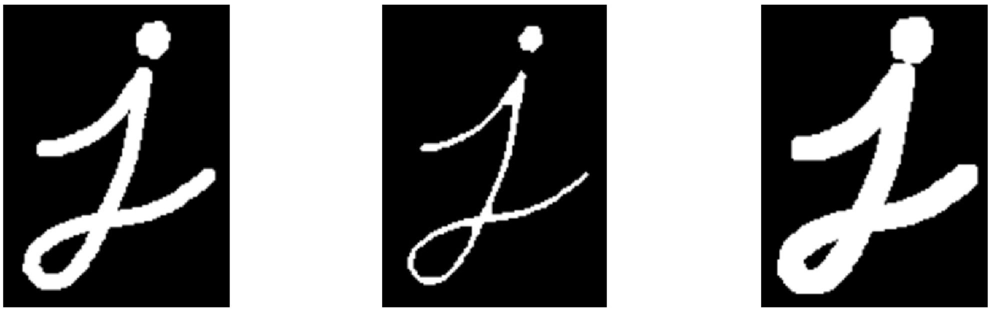
- **Erosion** – erodes away the boundaries of (shrinks) regions of foreground pixels (usually white pixels on black background in binary images).

- **Dilation** – gradually enlarges the boundaries of regions of foreground pixels.

In order to perform the transformation, we have to provide a so called *kernel*. The kernel is a structuring element that determines the precise effect of the morphological operator. The operators on binary images work as follows:

- Erosion – we set an output pixel only if all the pixels under the kernel at the same coordinates are set in the input image.
- Dilation – if an input pixel is set, we set output pixels in a shape of the kernel at the same coordinates as the input pixel.

See illustration of applying erosion and dilation operators in figure 5.4.



(a) Original binary image.

(b) Image after erosion.

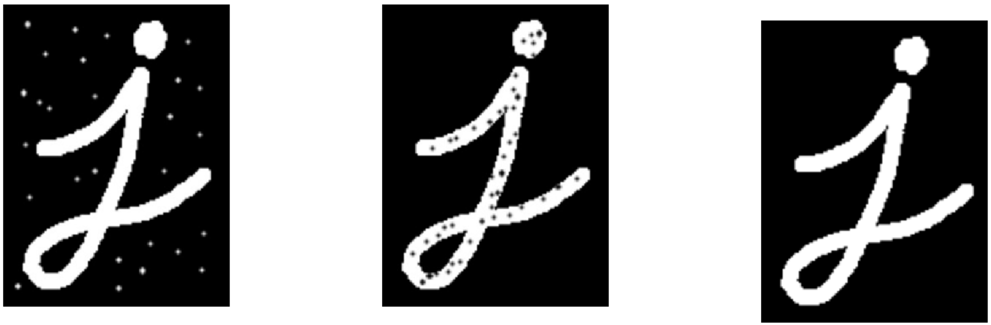
(c) Image after dilation.

Figure 5.4: Effects of erosion and dilation on a binary image. (Source: [21] [Chapter Morphological Transformations].)

The two basic transformations can by applied one after another. Since the order matters, the combinations have different names and separate functions in the OpenCV as well:

- **Opening** – erosion followed by dilation. Removes white noise outside the white compact objects (false positives).
- **Closing** – dilation followed by erosion. Removes black noise inside the white compact objects (false negatives).

Filtration effects of opening and closing morphological transformations are depicted in figure 5.5.



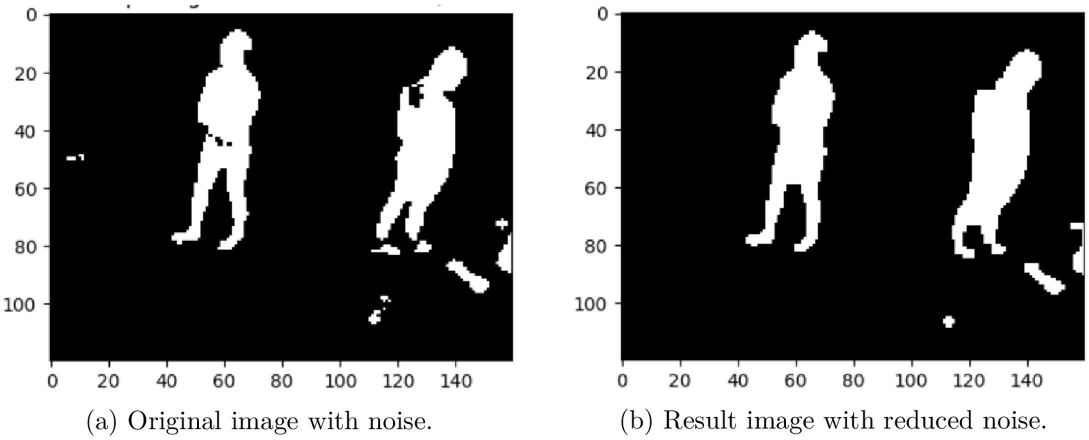
(a) Original noisy image suitable for opening.

(b) Original noisy image suitable for closing.

(c) Result image without noise.

Figure 5.5: Illustration of opening and closing morphological operators removing noise in a binary image. (Source: [21] [Chapter Morphological Transformations].)

In this project, we use the following sequence to filter thermal images. Firstly, the *opening* transformation with square kernel of size 2 is used and secondly, the *closing* with square kernel of size 5. These transformations remove noise in the image and make all found shapes more consistent and compact. An example of removing noise in the thermal image after applying adaptive mean thresholding can be seen in figure 5.6.



(a) Original image with noise.

(b) Result image with reduced noise.

Figure 5.6: Example of filtering effects of morphological transformations on a noisy thermal image after thresholding applied.

5.2.5 Object shape detection

The core of the detection method, we propose, lies in function `cv2.findContours()` of the OpenCV library. Given a binary input image, the function returns a list of contours found in the image. These contours are edge points of compact shapes (single color). It is recommended to apply this function only on binary image for best results.

In order to differentiate between persons and random small objects or heat noise, we encapsulate found objects into boxes with the help of function `cv2.contourArea()`. Then, we can calculate border box area for every contour and based on a specified threshold value , we are able to filter out small objects, that are most likely not people. See example of border boxes of found objects applied on the original thermal image in figure 5.7.

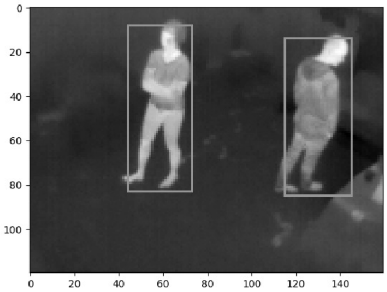


Figure 5.7: A normalized thermal image with drawn border boxes around detected persons.

This technique, however, also detects black objects on a white background—cold objects on warm ones. This behavior is unwanted. We can eliminate this problem by using *oriented* contour area. By setting the `oriented` argument, when calling `cv2.contourArea()`, to `True`, the returned area is a negative number, when the contour is white on black and positive otherwise. This way, we can discard black contours on white background, as we only want warm objects on relatively colder background scene. An example of incorrectly detected cold object in front of a warm one can be seen in figure 5.8.

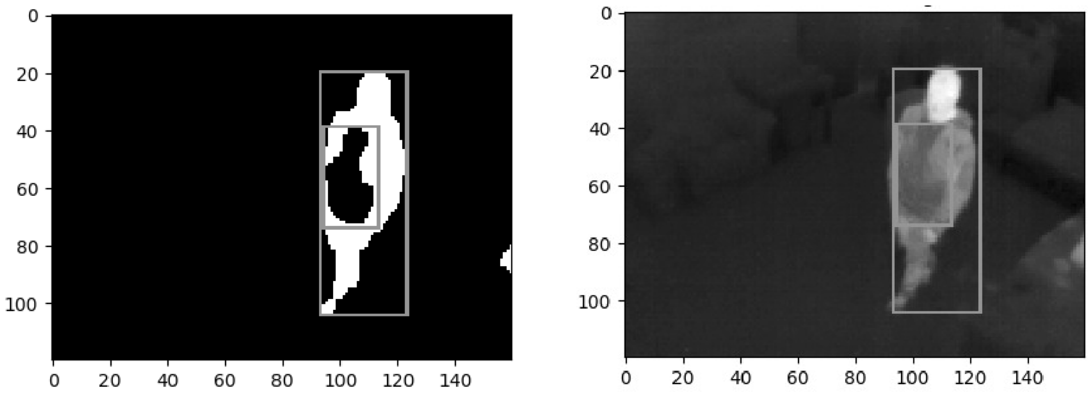
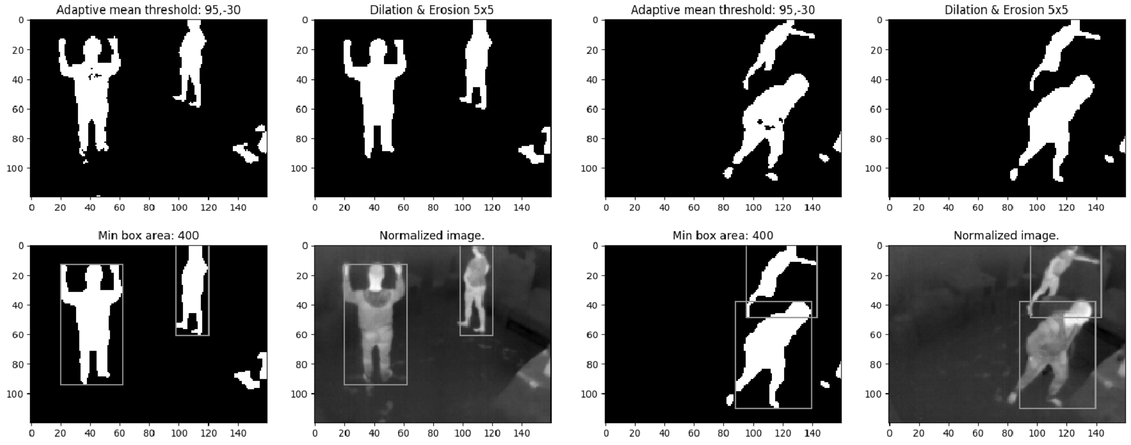


Figure 5.8: Example of incorrectly detected cold object (motorcycle body armor) in front of warm one before fix.

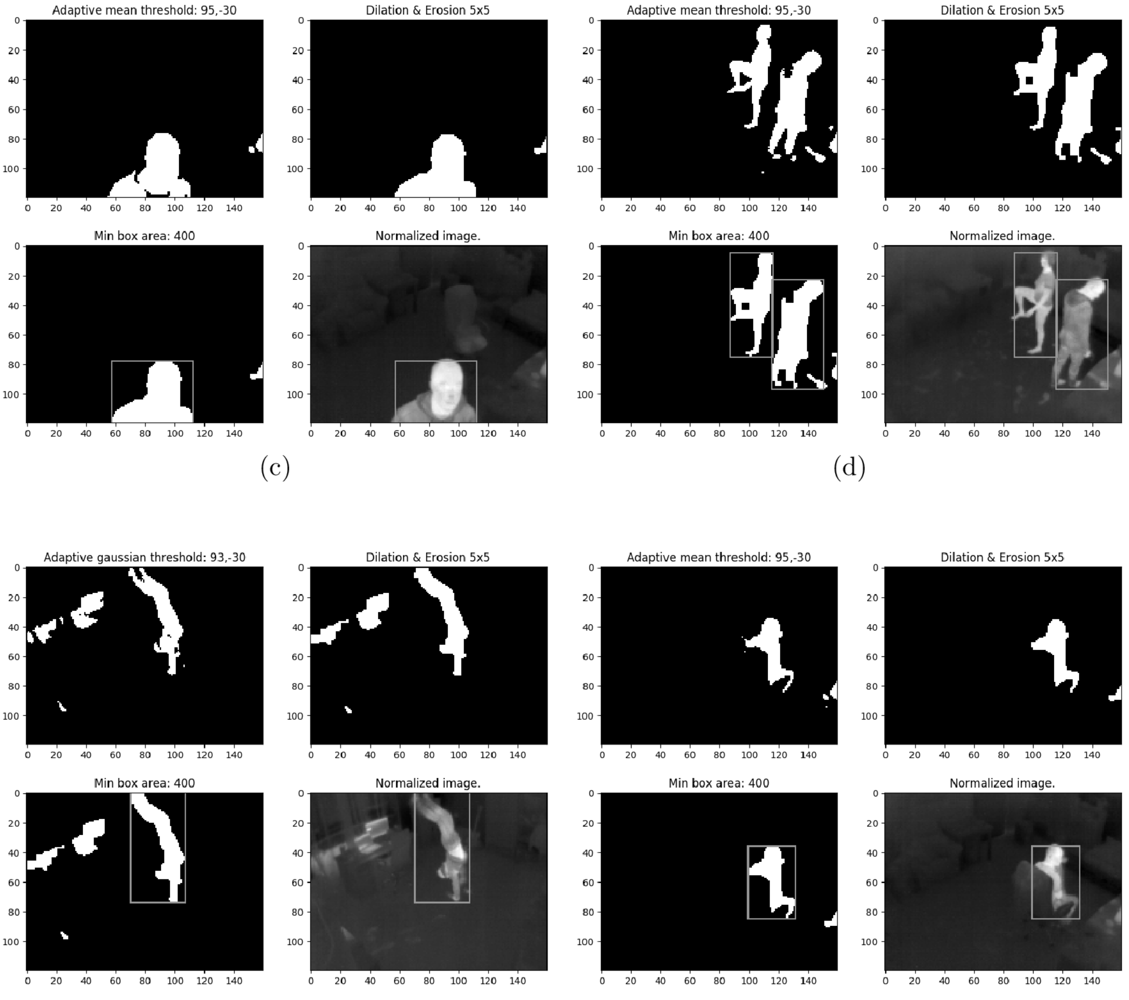
By acquiring a list of border boxes around detected persons in the thermal image, we can naturally straight away determine their count.

There are many more different approaches, when trying to detect objects in a stationary image. Every approach has its advantages and disadvantages. Even though our method of shape detection is very simple, it performs considerably well. In fact, this method correctly detected people in over **90 %** cases in our data set. Additionally, due to its simplicity, it has no trouble with detecting people in various poses—even very bizarre ones, as depicted in figure 5.11.



(a)

(b)



(c) (d)

(e) (f)

Figure 5.11: Examples of person detection algorithm performed on thermal images with people in various poses.

5.3 Scene reconstruction

At this point, we have bounding boxes around detected people with image coordinates from the camera. In order to approximate the object coordinates in world space, we first have to determine the camera pose in world space. Knowing the pose of a camera allows us to reconstruct the 3D scene and display the camera and detected objects in it. The camera pose estimation problem is often referred to as *Perspective-n-point* problem or *PnP*.

After obtaining the pose of the camera—more specifically its rotation and position (translation)—we are able to cast rays from the camera origin into world space using pixel coordinates of detected people, and therefore approximate their location in the observed environment.

Knowledge used in this section comes from [16], [19], [31].

5.3.1 Perspective-n-point problem

The perspective-n-point is the problem of estimating the pose of a calibrated camera [10]. By pose, we understand camera's position and orientation with respect to another coordinate system. We will represent the camera pose by rotation matrix $\mathbf{R}$ and translation vector $\mathbf{t}$. Solving the PnP problem requires pairs of corresponding 3D to 2D (world to image) mapping points. Given mapping points, estimating the pose is a matter of solving a system of linear equations. At least 4 point pairs are required to find a single solution. The perspective-n-point problem can be expressed by equation 5.2, which comes from the perspective projection (world to screen/image).

$$\mathbf{P_i} = \mathbf{K} \left[\ \mathbf{R} \ \middle| \ \mathbf{t} \ \right] \mathbf{P_w} \tag{5.2}$$

where $\mathbf{P_i}$ is an image point (2D), $\mathbf{K}$ is a matrix of intrinsic camera parameters, $\mathbf{R}$ is a rotation matrix, $\mathbf{t}$ is a translation vector and $\mathbf{P_w}$ is a world point (3D).

The expanded form of equation 5.2 can be found in equation 5.3.

$$\begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & \gamma & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \left[\begin{array}{ccc|c} r_{00} & r_{01} & r_{02} & t_x \\ r_{10} & r_{11} & r_{12} & t_y \\ r_{20} & r_{21} & r_{22} & t_z \end{array} \right] \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} \tag{5.3}$$

where the f_x and f_y are focal lengths, c_x and c_y are center point coordinates of the image (principal point) and γ is axis skew (usually assumed 0).

The $[\mathbf{R}|\mathbf{t}]$ matrix is usually extended into a single 4×4 matrix for the sake of convenience (equation 5.4). This matrix allows to project points from world to camera space (coordinate system) and thus is sometimes referred to as **world to camera**, *world to view* or simply *view* matrix.

$$\left[\ \mathbf{R} \ \middle| \ \mathbf{t} \ \right] = \begin{bmatrix} r_{00} & r_{01} & r_{02} & t_x \\ r_{10} & r_{11} & r_{12} & t_y \\ r_{20} & r_{21} & r_{22} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.4}$$

The matrix of intrinsic camera parameters $\mathbf{K}$ represents the transformation of a point from camera to screen (image) space. The matrix can be assembled from known camera parameters such as resolution and field of view or focal lengths (more on that later).

By plugging image points (2D) and corresponding world points (3D) into equation 5.3, it is possible to acquire rotation and translation vectors and therefore construct the *world to camera* transformation matrix used to project world points into camera space.

5.3.2 Scene reconstruction using OpenCV

By solving the *perspective-n-point* problem, we are able to determine rotation and position of the camera in world space based on pairs: 3D world coordinate and 2D corresponding image coordinate. The pose estimation can be done using a function `cv2.SolvePnP()` in the OpenCV library. [20] The function takes along with mentioned 2D and 3D points also the intrinsic camera matrix $\mathbf{K}$ and distortion coefficients.

$$\mathbf{K} = \begin{bmatrix} f_x & \gamma & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

The intrinsic camera matrix can be constructed with focal length in each axis and usually camera resolution—width x and height y. In our case, we calculate focal lengths f_x, f_y from horizontal and vertical field of view (FOV), which are known parameters of the Lepton 3 camera module.

For the intrinsic camera parameters matrix $\mathbf{K}$, we also need the principal point or center point c_x, c_y, which is a relative center point to the image origin. In equation 5.5, the center point is calculated using camera's image resolution x and y.

$$\begin{aligned} c_x &= \frac{x}{2} \\ c_y &= \frac{y}{2} \end{aligned} \tag{5.5}$$

$$\begin{aligned} f_x &= \frac{c_x}{\tan \frac{h_{fov}}{2}} \\ f_y &= \frac{c_y}{\tan \frac{v_{fov}}{2}} \end{aligned} \tag{5.6}$$

The γ variable represents *axis skew* causing shear distortion in the projected image. For our camera and simplicity, we assume $\gamma = 0$. In this project, we are also not taking into count radial nor tangential distortion of the camera, which means, that we keep distortion coefficients equal to 0.

Given array of world points, image points, intrinsic camera parameters and distortion coefficients, the `cv2.solvePnP()` function returns **rotation** and **translation vectors** of the camera, which represent the pose of the camera.

From the rotation vector, we are able to acquire the rotation matrix using the *Rodrigues'* algorithm[24] and express the whole transformation from world to camera space using a single matrix—**world to camera** matrix. The Rodrigues' algorithm is also available in the OpenCV library as `cv2.Rodrigues()`.

5.3.3 Reversing world to screen projection

After obtaining the *world to camera* matrix, we can reverse the whole (world to screen) transformation in order to project points from the image back to world space. To reverse the transformation, we need to rearrange equation 5.2 (or 5.3). The rearranged equation can be seen in equation 5.7.

$$\begin{bmatrix} x_w \\ y_w \\ z_w \\ w_w \end{bmatrix} = \left[\begin{array}{c|c} \mathbf{R}^{-1} & -\mathbf{R}^{-1} \cdot \mathbf{t} \\ \hline 0 & 1 \end{array}\right] \mathbf{K}^{-1} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} \tag{5.7}$$

where

$$-\mathbf{R}^{-1} \cdot \mathbf{t} = - \begin{bmatrix} r_{00} & r_{01} & r_{02} \\ r_{10} & r_{11} & r_{12} \\ r_{20} & r_{21} & r_{22} \end{bmatrix}^{-1} \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}$$

[24] *Rodrigues' rotation formula* – allows to compute a rotation matrix from an axis-angle representation (rotation vector).

Also note, that since the rotation matrix $\mathbf{R}$ is an orthogonal matrix with its determinant equal to 1, its inverse is equal to its transpose.

$$\mathbf{R}^{-1} = \mathbf{R}^T$$

The rearranged equation 5.7 describes the process of transformation of a point from image space back to world space. The intrinsic camera matrix inverse represents the transformation from screen (image) space into camera space. Again, we again can assemble a single 4×4 matrix simplifying the transformation from the camera coordinate system to the world coordinate system. The matrix corresponding to this transformation would be called **camera to world** matrix. This matrix is the key to reverse projecting points.

It is important to note, that when projecting any point from world space (3D) to screen space (2D), we always lose the depth dimension. When reverse projecting a point from screen space, it is impossible to determine the original world point, as infinite number of world points with varying depth gets projected into a single screen point.

What we can do with any screen point, that we want to reverse project, is to cast a ray from camera origin that intersects one arbitrary world point, which is projected to the initial screen point. The ray is effectively a line in world coordinates, whose points all get projected into the same one screen point.

For purposes of locating detected people, the ray would intersect the position of their feet or head. We can then assume, that the person is standing on the ground or we can calculate with an average person height to collapse the ray into a single point in world space. In this case, we would explicitly set the point's z coordinate—height from the ground. For any detected object in the thermal image enclosed with its bounding box, we can either assume its lower edge is touching ground or its upper edge is located the human average height above ground.

Reverse projection steps for any image point

For every image 2D point $\mathbf{P_i}$, we want to reverse project into a single world 3D point $\mathbf{P}$, the following steps are required:

1. Multiply the inverted matrix of intrinsic camera parameters $\mathbf{K^{-1}}$ and image point $\mathbf{P_i} = [x_i, y_i, 1]^T$. This projects the image point into camera space, resulting in point $\mathbf{P_c}$.

2. Project point in camera space $\mathbf{P_c} = [x_c, y_c, z_c, 1]^T$ into world space by multiplying the *camera to world* matrix with it. This results in an arbitrary world point $\mathbf{P_w}$, that gets projected to initial image point $\mathbf{P_i}$.

3. Use step 2 to project camera origin in camera space $\mathbf{C_c} = [0, 0, 0, 1]^T$ into point in world space $\mathbf{C_w}$. (Note, that this step can be done only once for a particular scene.)

4. Calculate euclidean vector $\mathbf{R}$ (ray direction) with the direction from camera origin $\mathbf{C_w}$ to point $\mathbf{P_w}$. This vector with the camera origin effectively describe the ray or line in world space, whose points get projected into initial image point $\mathbf{P_i}$. We obtain the vector by subtracting the camera origin from the projected point.

$$\mathbf{R} = \mathbf{P_w} - \mathbf{C_w}$$

5. By scaling vector **R** with scalar s and adding it to the camera origin, find the single result reverse projected point **P** on the ray in world space.

$$\mathbf{P} = \mathbf{C_w} + s \cdot \mathbf{R}$$

   Scalar s can be adjusted, so that the result point meets our specific requirements. In our case, we want the z coordinate of the point to have a specific value z. We compute the scalar the following way:

$$s = \frac{z - z_{C_w}}{z_R}$$

This method of locating people turned out to work very well. Its weakest point is the necessary precision of the mapping 3D to 2D points, which are required for a proper estimation of the pose of the camera. An example representing detected and located people can be seen in figure 5.12.

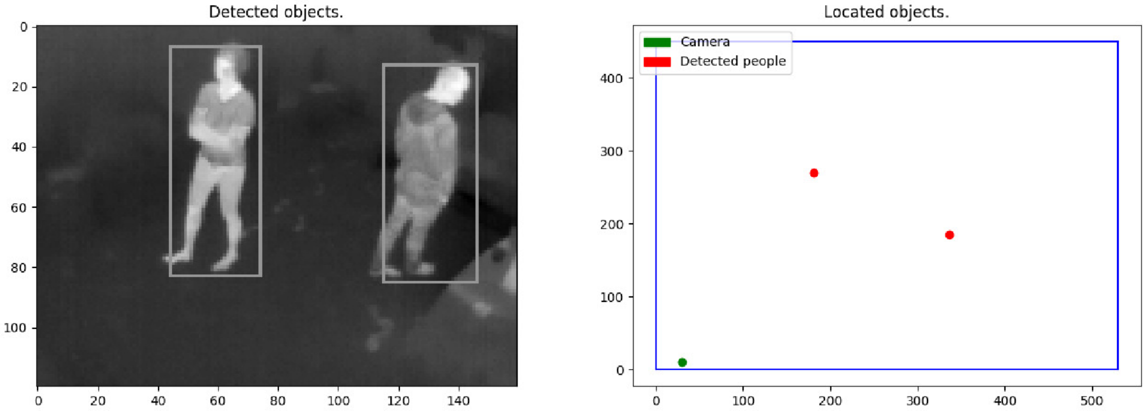


Figure 5.12: Example of a room with detected people from top view.

5.3.4 Scene software abstraction

As mentioned at the beginning of this chapter, in the second part of the software published with this thesis, we provide a scene abstraction, that can be used to model a room with borders, dimensions and a camera inside. The scene model also allows to reverse project image points captured with the camera into that room. The abstraction is provided by Python class `Scene` in the `ThermoDetection` module.

Typical steps, used to calibrate and utilize this scene model, would be the following:

1. Instantiate the `Scene` class.

```
s = Scene()
```

2. Set the position of the camera – (x, y) coordinates. These are the coordinates used for marking the camera in the top view representation of the scene.

```
s.set_camera_position((x, y))
```

3. Set camera intrinsic parameters. This function builds and stores the intrinsic camera matrix $\mathbf{K}$ and its inverse $\mathbf{K^{-1}}$.

   ```
   s.set_intrinsic_camera_parameters(width, height, horizontal_fov,
    ↪  vertical_fov)
   ```

4. Set scene vertices—borders of the scene (edge vertices, corners of the room).

   ```
   s.add_scene_vertices(vertex_list)
   ```

5. Calibrate the scene, find the pose of the camera. The function builds and stores *world to camera* and *camera to world* matrices. It is necessary to provide at least 4 mapping points from 3D world space to 2D image space.

   ```
   s.scene_calibrate(points_3d, points_2d)
   ```

6. Reverse project points from image space to world space with a desired z coordinate.

   ```
   s.reverse_project_zcoord((x,y), desired_z)
   ```

7. Plot the top view representation of the scene with borders, camera and projected points.

   ```
   s.draw_scene()
   ```

Chapter 6

Conclusion

The goal of this project was to utilize a small low-cost thermal imaging camera Lepton 3 with the combination of image processing to create another solution to the problem of detecting and/or locating people, which could find its usage in many areas, especially where privacy plays an important role.

Even though the project title might suggest the main focus to be on techniques, different approaches regarding image processing and ways of detecting people, a big part of the project had to deal with the necessary hardware part—creating a working software abstraction over the Lepton 3 thermal camera module, that would provide capture and control mechanisms.

We have implemented fully functional high-level Python script for controlling the Lepton 3 camera module. The control script can be easily extended to support more commands, thanks to the provided infrastructure. This control script is a smaller part of the `v4l2lepton3` library, which was developed for purposes of this thesis, and is as well ready to use. The other part of the library deals with video capture from the camera.

Furthermore, we include a single frame capturing Python script. The single capture script, altogether with the *v4l2lepton3* C++ library open various options for capturing thermal video stream from the camera. We can access frames from the Lepton 3 camera module directly from Python environment, or if we need to capture frames continuously, we can utilize the faster option—using the provided C++ library, that feeds frames into a virtual video device. From the virtual video device, we can process video frames using standard Linux tools as `vlc`, `ffmpeg` and others or stream it over a network and process them elsewhere.

There is, however, undoubtedly a lot of opportunities to improve and extend current relatively simple people detection algorithm, which uses temperature filtering and shape detection with a help of the OpenCV library. The detection mechanism is the crucial part. Once we obtain boxes enclosing detected people, it is only a matter of math to approximate their location in a known environment. The model used for scene reconstruction displaying position of detected people, which is also included in the software part of this thesis, is accurate and reliable. Its only weakness is the necessity of accurate mapping points for calculating the camera pose (calibrating the model).

The method of detecting people used in this project and described in this thesis, even though it is based on a very simple concept, works well in over 90 % cases on our data set. On one hand, it is common, that simpler solutions can sometimes outperform and be more reliable than complex solutions, however, in this case, I believe, that including other techniques, would improve its accuracy and stability even more.

It is important to note, that even though we have achieved relatively high success rate of detecting people in images collected at different places from different angles under different conditions, the data was collected and examined during winter time. Before using the system in real applications, it would be necessary to make sure the algorithm stays stable even during the summer or in environments with higher ambient temperature. Wearing different types of clothing, especially clothes designed to isolate heat (low thermal conductivity index), might also significantly decrease the success rate of people detection. It is important to bear this in mind, when deciding which people detection mechanism to use in a particular use case and environment.

One of the candidate techniques for improvement, would be implementing motion tracking. [4] This mechanism would increase the success rate and lower the probability of false positives. We could first capture a scene without any people and create a subtraction mask. This would help to filter out static warm objects in the scene, that are the same temperature as human beings, for instance heavy industry machines, computers and so on. We would also be able to detect warm objects entering and leaving the scene. This would improve overall stability and help solving an issue, the system currently has, and that is occluded scenes—when a person is located behind another, the system detects them as a single person, which might be unacceptable in some situations.

There are also various other approaches to object detection itself, mostly in the machine learning field. We could try to use one of older techniques, previously used a lot with face recognition—Histograms of Oriented Gradients (HOG) [9] as feature sets fed into a support vector machine (SVM). A larger data set with features extracted by hand would be required. This method would, however, take more processing time to perform and the effects of the speed decrease would have to be tested.

Since we now live in the deep learning era, we might also incorporate deep learning—specifically convolutional neural networks. [28] There is a new technique called *YOLO* (You Only Look Once)[26] [27]. *YOLO* is a technique which is based on a convolutional neural network and is the current state-of-the-art in real-time object detection [25]. I believe, that *YOLO* with motion tracking would be the ultimate solution to the problem of real-time people detection from thermal imagery. *YOLO* however, since it is based on a neural network, requires even a bigger data set to be trained on than SVMs and would struggle with uncommon person poses. With enough data however, this technique has the potential to outperform all the other methods. *YOLO* with motion tracking shall be a subject for further research and testing in this area.

The following tables sum up all parts of the project with their completion level, quality of the solution or approximated success rate and options for improvement:

Hardware communication between Lepton 3 and Orange Pi	
Description	• SPI and I^2C hardware interfaces enabled on the Orange Pi computer. • Mutual communication established.
Completion	Done.
Improvements	None.

Controlling Lepton 3 camera over I^2C	
Description	• A Python script implemented as a part of the `v4l2lepton3` library. • Supports important commands.
Completion	Done, working with no issues. Provided infrastructure for easy extentions.
Improvements	Adding support for more commands, if necessary.

Capturing video frames	
Description	• A Python script implemented for single frame capture. • Implemented `v4l2lepton3` library in C++ for sequential capture without losing synchronization with the camera. Allows general access to the video stream from a virtual video device.
Completion	Done. Working without losing synchronization.
Improvements	If possible, provide more general access to the Y16 RAW stream of frames in the virtual video device from a Python environment, as not all versions of OpenCV support RAW format.

Algorithm for detecting people	
Description	• Provided Python helper library and example scripts. • Detection is performed from a raw thermal image. Algorithm is based on a simple method: temperature filtering, thresholding, morphological filtering, finding contours, applying border boxes.
Completion	Done. Simple, but working method. 90 % success rate on our data set, however further testing needed. (warm environment, occluded objects, and so on)
Improvements	This algorithm is the crucial part of the project and will be subject to further research. There are various possibilities for improvement and experimenting: 1. Motion tracking with background temperature masking. 2. HOG[25] and SVN[26] based detection. 3. CNN[27] based YOLO[28] object detection.

[25] HOG – Histograms of Oriented Gradients [9]

[26] SVN – Support Vector Machine

[27] CNN – Convolutional Neural Network

[28] YOLO – You Only Look Once [26]

Algorithm for locating people	
Description	• Provided Python scene abstraction class for 3D reconstruction with example scripts. • Method is based on reverse screen to world projection (ray casting). Assuming, that objects touch the ground or calculating with average person height.
Completion	Done. People position estimation is accurate. The only weak point is precise camera pose calibration in a particular scene.
Improvements	Mathematically the algorithm is very accurate, however strongly depends on people detection algorithm. We could include heuristics: people crouching, person too tall, which could mean two people behind each other etc. If needed, thermal stereo vision could be used to improve accuracy even more.

Bibliography

[1] AB Electronics UK: *I2C, SMBus and Armbian Linux*. Feb 2017. [Online; Accessed: 25-05-2018].
Retrieved from: https://www.abelectronics.co.uk/kb/article/1061/i2c--smbus-and-armbian-linux

[2] Airport Suppliers, Acorel SAS: *Automatic High Accuracy Passenger Counting Systems*. 2016. [Online; Accessed: 28-05-2018].
Retrieved from: https://www.airport-suppliers.com/supplier/acorel/

[3] Alhamoud, A.; Nair, A. A.; Gottron, C.; et al.: Presence detection, identification and tracking in smart homes utilizing bluetooth enabled smartphones. In *39th Annual IEEE Conference on Local Computer Networks Workshops*. Sep 2014. pp. 784–789. doi:10.1109/LCNW.2014.6927735.

[4] Andriluka, M.; Roth, S.; Schiele, B.: People-tracking-by-detection and people-detection-by-tracking. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*. Jun 2008. ISSN 1063-6919. pp. 1–8. doi:10.1109/CVPR.2008.4587583.

[5] AnonymousPi: *Using 16x2 ('1602') LCD with I2C connector with Orange Pi PC*. May 2017. [Online; Accessed: 25-05-2018].
Retrieved from: https://forum.armbian.com/topic/4377-using-16x2-1602-lcd-with-i2c-connector-with-orange-pi-pc/

[6] Armbian Community: *Armbian official documentation*. 2017. [Online; Accessed: 25-05-2018].
Retrieved from: https://docs.armbian.com/

[7] Armbian Official Forum, Technical Support: *How to enable hardware SPI – Allwinner H2 & H3*. Mar 2017. [Online; Accessed: 29-05-2018].
Retrieved from:
https://forum.armbian.com/topic/3772-how-to-enable-hardware-spi/

[8] Bradski, G.; Kaehler, A.: *Learning OpenCV*. O'Reilly Media, Inc.. 09 2008. ISBN 978-0-596-51613-0. first edition.

[9] Dalal, N.; Triggs, B.: Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 1. June 2005. ISSN 1063-6919. pp. 886–893. doi:10.1109/CVPR.2005.177.

[10] DeMenthon, D. F.; Davis, L. S.: Model-Based Object Pose in 25 Lines of Code. *International Journal of Computer Vision*. vol. 15, no. 2. June 1995: pp. 123–141.

[11] FLIR Commercial Systems, Inc.: *FLIR LEPTON® 3 Long Wave Infrared (LWIR) Datasheet*. 2014. document number: 500-0726-01-09 rev. 100.

[12] FLIR Commercial Systems, Inc.: *FLIR LEPTON® Lepton vs. Lepton 3 Application Note*. 2014. document number: 102-2012-100-01 rev. 100.

[13] FLIR Commercial Systems, Inc.: *FLIR LEPTON® Software Interface Description Document (IDD)*. 2014. document number: 110-0144-04 rev. 200.

[14] GroupGets LLC: *FLIR Lepton Breakout Board Official Page, Schematics, Specifications*. 2018. [Online; Accessed: 25-05-2018].
Retrieved from: https://groupgets.com/manufacturers/getlab/products/flir-lepton-breakout-board-v1-4

[15] Hallinan, C.: *Embedded Linux Primer: A Practical Real-World Approach, 2nd Edition*. chapter 7. Bootloaders in Embedded Linux Systems. Prentice Hall. second edition. Oct 2010. ISBN 0-13-701783-9. [Online; Accessed: 26-05-2018].
Retrieved from:
http://www.informit.com/articles/article.aspx?p=1647051&seqNum=5

[16] Hartley, R.: *Multiple View Geometry in Computer Vision*. Cambridge University Press. second edition. 2004. ISBN 354049698X.

[17] Ivanov, B.; Ruser, H.; Kellner, M.: Presence detection and person identification in Smart Homes. Jul 2014.

[18] Kyle McDonald: *Structured Light 3D Scanning*. Instructables website. Dec 2009. [Online; Accessed: 27-06-2018].
Retrieved from:
http://www.instructables.com/id/Structured-Light-3D-Scanning/

[19] Kyle Simek: *Dissecting the Camera Matrix: Part 1,2,3 – Extrinsic/Intrinsic Camera Matrix*. Kyle Simek's Computer Vision Blog. 2012-2013. [Online; Accessed: 18-06-2018].
Retrieved from: http://ksimek.github.io/2012/08/14/decompose/

[20] Mallick, S.: *Head Pose Estimation using OpenCV and Dlib*. Learn OpenCV blog. Sep 2016. [Online; Accessed: 25-05-2018].
Retrieved from:
https://www.learnopencv.com/head-pose-estimation-using-opencv-and-dlib/

[21] Mordvintsev, A.; K., A.: *OpenCV-Python Tutorials*. 2013. [Online; Accessed: 26-05-2018, Revision 43532856].
Retrieved from: http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_tutorials.html

[22] Negied, N. K.; Hemayed, E. E.; Fayek, M. B.: Pedestrians' detection in thermal bands – Critical survey. *Journal of Electrical Systems and Information Technology*. vol. 2, no. 2. September 2015: pp. 141–148.

[23] OpenCV Dev Team: *OpenCV 3 official documentation*. Feb 2018. [Online; Accessed: 26-05-2018].
Retrieved from: https://docs.opencv.org/3.4.1/

[24] People Counting PRO: *Smart Counter DATA: wireless, data logging by 365 days, infrared beam – product page*. People Counting PRO eshop. 2018. [Online; Accessed: 27-06-2018].
Retrieved from: https://peoplecounting.pro/product/condor-8-people-counter-with-data-logging/

[25] Raval, S.: *YOLO Object Detection*. Nov 2017. [Online; Accessed: 26-05-2018].
Retrieved from: https://github.com/llSourcell/YOLO_Object_Detection/blob/master/YOLO%20Object%20Detection.ipynb

[26] Redmon, J.; Divvala, S. K.; Girshick, R. B.; et al.: You Only Look Once: Unified, Real-Time Object Detection. *CoRR*. vol. abs/1506.02640. 2015.
Retrieved from: http://arxiv.org/abs/1506.02640

[27] Redmon, J.; Farhadi, A.: YOLO9000: Better, Faster, Stronger. *CoRR*. vol. abs/1612.08242. 2016.
Retrieved from: http://arxiv.org/abs/1612.08242

[28] Rodger, I.; Connor, B.; Robertson, N. M.: Classifying objects in LWIR imagery via CNNs. In *Proc. SPIE: Electro-Optical and Infrared Systems: Technology and Applications XIII*, vol. 9987. 10 2016. pp. 99870–99884. doi:10.1117/12.2241858. winner of Best Student Paper prize.

[29] Ruser, H.; Pavlov, V.: *People counter based on fusion of reflected light intensities from an infrared sensor array*. vol. 1. Jan 2006: pp. 379–383. conference: Informatik 2006 - Informatik für Menschen.

[30] Smith, J. W.: *Capturing a webcam stream using v4l2*. Dec 2014. [Online; Accessed: 26-05-2018].
Retrieved from:
http://jwhsmith.net/2014/12/capturing-a-webcam-stream-using-v4l2/

[31] Solem, J. E.: *Programming Computer Vision with Python: Tools and algorithms for analyzing images*. O'Reilly Media, Inc.. 2012. ISBN 1449316549,9781449316549. first edition.

[32] Szeliski, R.: *Computer Vision: Algorithms and Applications*. Springer Science & Business Media. 2011. ISBN 1848829345,9781848829343.

[33] The Economist: *In-store detecting – A new industry has sprung up selling "indoor-location" services to retailers*. The Economist newspaper. Dec 2016. [Online; Accessed: 27-06-2018].
Retrieved from: https://www.economist.com/business/2016/12/24/a-new-industry-has-sprung-up-selling-indoor-location-services-to-retailers

[34] Umbaugh, S. E.: *Digital Image Processing and Analysis: Human and Computer Vision Applications with CVIPtools, Second Edition*. CRC Press Taylor & Francis Group. 2010. ISBN 14-398-0205-X.

[35] Xunlong Software CO.,Limited: *Xunlong Orange Pi PC 2 – product wiki*. April 2018. [Online; Accessed: 25-05-2018].
Retrieved from: http://linux-sunxi.org/Xunlong_Orange_Pi_PC_2