BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INTELLIGENT SYSTEMS
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

SYSTEM FOR PEOPLE DETECTION AND LOCALIZATION USING THERMAL IMAGING CAMERAS

SYSTÉM PRO DETEKCI A LOKALIZACI OSOB POMOCÍ TERMÁLNÍCH KAMER

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR **Bc. MICHAL CHARVÁT**
AUTOR PRÁCE

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

BRNO 2020

Brno University of Technology
Faculty of Information Technology

Department of Intelligent Systems (DITS) Academic year 2019/2020

Master's Thesis Specification

Student: **Charvát Michal, Bc.**
Programme: Information Technology Field of study: Computer and Embedded Systems
Title: **System for People Detection and Localization Using Thermal Imaging Cameras**
Category: Embedded Systems
Assignment:

1. Study the literature oriented on thermal image acquisition, processing and indoor detection of people. Familiarize yourself with the low-cost thermal imaging camera available at FIT BUT.
2. Design an enclosure for a thermal unit consisting of Lepton 3.5 thermal camera, RaspberryPi 3B+ and a custom load switch circuit board allowing the camera to be hardware disconnected remotely.
3. Deploy a system of 3 thermal units to selected real installation with custom software allowing for remote managing and data collection.
4. Use data collected from the real installation for experiments with various algorithms for people detection and location estimation for purposes of zone guarding and building of heatmaps.
5. Perform experiments and summarize the achieved results, including discussion devoted to these results.

Recommended literature:

- HERRMANN, Christian; RUF, Miriam; BEYERER, Jürgen. CNN-based thermal infrared person detection by domain adaptation. In: *Autonomous Systems: Sensors, Vehicles, Security, and the Internet of Everything*. International Society for Optics and Photonics, 2018. p. 1064308.
- SCHULTE, Stefan; DE MUYNCK, Steffen. *Thermal-image based object detection and heat map generation systems and methods*. U.S. Patent Application No 16/014,112, 2018.

Requirements for the semestral defence:

- Items 1 and 2.

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor: **Drahanský Martin, prof. Ing., Dipl.-Ing., Ph.D.**
Head of Department: Hanáček Petr, doc. Dr. Ing.
Beginning of work: November 1, 2019
Submission deadline: July 31, 2020
Approval date: October 31, 2019

Abstract

In today's world, there is an increasing need for automatic reliable mechanisms for detecting and localizing people – from performing people flow analysis in museums, controlling smart homes to guarding hazardous areas like railway platforms. We propose a method for detecting and locating people using low-cost FLIR Lepton 3.5 thermal cameras and a Raspberry Pi 3B+ computers. This thesis describes the continuation of the "Detection of People in Room Using Low-Cost Thermal Imaging Camera" project, which now supports modelling of complex scenes with polygonal boundaries and multiple thermal cameras observing them. In this paper, we introduce an improved control and capture library for the Lepton 3.5, a new person detection technique that uses the state-of-the-art YOLO (You Only Look Once) real-time object detector based on deep neural networks, furthermore, a new thermal unit with automated configuration using Ansible encapsulated in a custom 3D printed enclosure for safe manipulation, and last but not least, a step by step instruction manual on how to deploy the detection system in a new environment including other supporting tools and improvements. The results of the new system are demonstrated on a simple people flow analysis performed in the Czech National Museum in Prague.

Abstrakt

V dnešním světě je neustále se zvyšující poptávka po spolehlivých automatizovaných mechanismech pro detekci a lokalizaci osob pro různé účely – od analýzy pohybu návštěvníků v muzeích přes ovládání chytrých domovů až po hlídání nebezpečných oblastí, jimiž jsou například nástupiště vlakových stanic. Představujeme metodu detekce a lokalizace osob s pomocí nízkonákladových termálních kamer FLIR Lepton 3.5 a malých počítačů Raspberry Pi 3B+. Tento projekt, navazující na předchozí bakalářský projekt "Detekce lidí v místnosti za použití nízkonákladové termální kamery", nově podporuje modelování komplexních scén s polygonálními okraji a více termálními kamerami. V této práci představujeme vylepšenou knihovnu řízení a snímání pro kameru Lepton 3.5, novou techniku detekce lidí používající nejmodernější YOLO (You Only Look Once) detektor objektů v reálném čase, založený na hlubokých neuronových sítích, dále novou automaticky konfigurovatelnou termální jednotku, chráněnou schránkou z 3D tiskárny pro bezpečnou manipulaci, a v neposlední řadě také podrobný návod instalace detekčního systému do nového prostředí a další podpůrné nástroje a vylepšení. Výsledky nového systému demonstrujeme příkladem analýzy pohybu osob v Národním muzeu v Praze.

Keywords

Lepton, thermal, detection, YOLO, localization, person, people, Lepton 3.5, YOLOv4, real-time, detector, Raspberry Pi, neural network, deep learning, AI, artificial intelligence, FLIR, scene reconstruction, reverse projection, security system, people flow analysis

Klíčová slova

Lepton, termální, detekce, YOLO, lokalizace, osoba, lidi, Lepton 3.5, YOLOv4, reálný čas, detektor, Raspberry Pi, neuronová síť, hluboké učení, UI, umělá inteligence, FLIR, rekonstrukce scény, zpětná projekce, bezpečnostní systém, analýza pohybu lidí

Reference

CHARVÁT, Michal. *System for People Detection and Localization Using Thermal Imaging Cameras.* Brno, 2020. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor prof. Ing., Dipl.-Ing. Martin Drahanský, Ph.D.

Rozšířený abstrakt

V dnešním světě je neustále se zvyšující poptávka po spolehlivých automatizovaných mechanismech pro detekci a lokalizaci osob. Tyto mechanismy mohou například pomáhat starším či zdravotně postiženým osobám s každodenním životem ovládáním chytrých domovů nebo zvyšovat úroveň bezpečnosti na pracovištích tím, že budou hlídat rizikové zóny. Takové zóny mohou představovat např. kolejiště na vlakových nástupištích nebo oblasti v blízkosti nebezpečných strojů ve výrobních halách. Mimo jiné se metody detekce a počítání lidí používají také k analýze toku lidí v hromadné dopravě, v obchodech či v muzeích, kde se zjišťuje, které části expozice jsou pro návštěvníky nejzajímavější.

Tento projekt navazuje na předchozí bakalářský projekt "Detekce lidí v místnosti za použití nízkonákladové termální kamery" a klade si za cíl odstanit některé jeho nedokonalosti a implementovat navrhovaná vylepšení. Představujeme vylepšenou metodu detekce a lokalizace osob s pomocí nízkonákladových termálních kamer FLIR Lepton 3.5 a malých počítačů Raspberry Pi 3B+.

Tento text v úvodu připomíná motivaci projektu, obecné možnosti použití mechanismů pro detekci a lokalizaci osob a aktuálně používané technologie, které navzájem srovnává, a popisuje i jejich konkrétní použití. Tyto mechanismy můžeme obrazně rozdělit do dvou kategorií: jedna, sloužící primárně k jednoduššímu počítání průchodů osob, 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 tento systém pro detekci a lokalizaci osob termokamerou. Oproti ostatním řešením má však výraznou výhodu, a to, že s termálním kamerovým modulem o malém rozlišení přirozeně nelze provést rozpoznávání obličeje, tudíž může být vhodným řešením na místa, kde soukromí hraje důležitou roli, jako domovy nebo pracoviště.

Další část práce popisuje novou termální jednotku systému, složenou z termální kamery Lepton 3.5, malého počítače Raspberry Pi 3B+ a vlastního obvodu pro řízení napájení kamery.

Použitý řídicí počítač Orange Pi PC2 byl v novém projektu nahrazen tradičním jednodeskovým počítačem Raspberry Pi 3B+, který je na rozdíl od Orange Pi aktivně vyvíjen velkou komunitou, je kompletní a snadno konfigurovatelný, co se týká hardwarových rozhraní a linuxových kernelů.

V novém projektu byla použita novější kamera Lepton 3.5 (namísto 3.0), která podporuje tzv. pravou radiometrii. To znamená, že má kamera v sobě zabudovanou a zkalibrovanou funkci pro převod dopadajícího světelného toku infračerveného záření na teplotu v Kelvinech. Starší verze kamery vyžadovala dodání takové funkce od uživatele a byla častým zdrojem nepřesností. Lepton 3 má pouze funkci nepravé radiometrie, což znamená, že firmware kamery zaručuje při různých teplotách kamery a okolí stále stejnou výstupní hodnotu pixelu pro stejnou teplotu pozorovaného objektu.

Lepton 3, a bohužel, i 3.5 se potýkají s nepříjemným problémem. Čas od času se stane, že kamera přestane posílat termální snímky a přijímat řídicí příkazy, tím pádem nemůže být ani restartována pomocí příkazu ovládacího rozhraní `OEM_REBOOT`. V takových případech pomůže pouze kameru odpojit a připojit zpět, což je v naší situaci nepřijatelné, neboť termální jednotka systému musí být kompletně spravována vzdáleně přes síť; fyzický přístup k jednotkám není možný. Z tohoto důvodu byla termální jednotka nově vybavena vlastním obvodem pro řízení napájení kamery pomocí GPIO výstupu počítače Raspberry Pi. Tento obvod je umístěn na desce plošných spojů a pomocí NPN a P-MOSFET tranzistorů přivádí napájecí napětí 5 V do kamery. Tento obvod umožňuje kameru úplně odpojit, když není

zrovna potřeba, a také ji restartovat, pokud přestane odpovídat. Občas se stane, že ani odpojení hlavního napájení nevynutí restart kamery, neboť je spínaný zdroj uvnitř kamery stále nápájen z datových sběrnic I^2C nebo SPI. Při odpojení sběrnic i napájení je však kamera vždy spolehlivě restartována. Jelikož je tento řídicí obvod ovládán z 3,3V výstupu počítače Raspberry Pi, je možné kameru zapínat a vypínat vzdáleně.

V předchozím projektu byla kamera a řídicí počítač umístěny na nepájivém poli a spojeny pouze vodiči. To neumožňovalo jakékoliv přenášení nebo demonstraci, protože bylo toto snímací zařízení velmi křehké. V novém projektu figuruje pojem *termální jednotka* jako označení pro Lepton 3.5 kameru, Raspberry Pi 3B+ řídicí počítač a řídicí obvod napájení umístěné ve speciální schránce vytisknuté na 3D tiskárně. Tento ochranný box byl navržen a vyroben v rámci tohoto projektu a zajišťuje ochranu pro jeho tři vnitřní komponenty, dostatečné chlazení pro řídicí počítač a příhodné uchycení termální kamery tak, aby bylo možné měnit pozorovací úhel scény pomocí ladicího šroubu.

V práci jsou popsány kroky instalace všech potřebných softwarových komponent na Raspberry Pi k správné funkčnosti termální jednotky. Konfigurační proces byl automatizován pomocí agilního konfiguračního nástroje Ansible. Pro každou novou termální jednotku je tento nástroj spuštěn a ten podle předem definovaného předpisu zajistí, že na cílovém zařízení jsou nainstalovány všechny nezbytné součásti, nástroje a knihovny.

Termální kamery Lepton používají SPI rozhraní pro odesílání video snímků a I^2C sběrnici pro řízení kamery. Pro komunikaci s kamerou pomocí těchto nízkoúrovňových rozhraní slouží právě malý řídicí počítač, jehož procesor obsahuje hardwarové moduly pro jejich řízení. Protokoly pro přenos videa a vydávání příkazů pro kameru se ve své podstatě mezi verzemi 3 a 3.5 nezměnily. Nová verze kamery obsahuje více příkazů, které jsou spojeny s již zmíněnou pravou radiometrií, a ukázalo se, že v kombinaci s počítačem Raspberry Pi 3B+ vyžaduje urychlení vyčítání snímků až 8 ×, aby nedocházelo k desynchronizaci.

Knihovna pro řízení a snímání `v4l2lepton3`, která se vyskytovala již v předchozím projektu, byla zcela přeprogramována a rozdělena na C++ aplikaci a Python3 knihovní balíček. Software pro ovládání kamery byl přesunut do Python3 balíku a byl výrazně zgeneralizován. Nově obsahuje definice metodyod a nastavitelných hodnot všech podporovaných příkazů, výrazně větší množství příkazů a automaticky generované metody pro všechny příkazy, což výrazně redukuje redundanci předchozího řešení.

Knihovna `v4l2lepton3` byla v předchozím projektu navržena pro lokální komunikaci. Jelikož požadavky nového systému pro detekci a lokalizaci osob zahrnují podporu více kamer a agregaci detekce na jednom místě, je nutné, aby knihovna zvládala přenášet termální snímky v surové podobě přes síť, což doposud nebylo možné. Nová C++ aplikace pro snímání následuje klient-server model a je tedy rozdělena na dvě části. Centrální vyhodnocující počítač se chová jako klient a připojuje se na všechny ostatní termální jednotky, které se chovají jako servery. Serverová část aplikace byla výrazně urychlena redukcí volání OS a double bufferingem snímků i segmentů. Server umožnuje zapnout ZLIB kompresi, zotaví se z jakékoliv chyby a neztrácí synchronizaci s kamerou.

Klientská strana je implementována v C++ i v Pythonu jako součást `v4l2lepton3` Python3 knihovního balíčku. C++ klient následuje myšlenku využití virtuálního video zařízení `v4l2loopback` a zpřístupňuje vzdálený termální stream v lokálním video zařízení. Python implementace klienta je součástí knihovny, jde ji snadno zahrnout ve vlastních projektech a je využívána ve zbytku detekčního systému. Součástí repozitáře s knihovnou `v4l2lepton3` jsou také spustitelné Python3 skripty `lepton3client.py`, `lepton3capture.py` a `lepton3control.py`, které zpřístupňují implementaci snímání videa, jednotlivých snímku a ovládání z knihovny.

Následující část práce popisuje nedokonalosti v detekčním algoritmu, který byl testován ve větších prostorách s více lidmi, kde se ukázalo, že v takto náročných prostředích původní jednoduchý detekční algoritmus selhává. Bylo tedy nutné provést průzkum a srovnání dostupných alternativních detekčních algoritmů, ze kterého byl vybrán nejmodernější YOLO detektor objektů v reálném čase založený na hlubokých neuronových sítích. V rámci nového projektu bylo natrénováno a porovnáno několik verzí a velikostí YOLO detektoru, z nichž nejlépe vyšel nejnovější YOLOv4 o velikosti 320 $\times$ 320. Výsledný detektor je výrazně lepší než původní jednoduchý detektor a bez problému si poradí i s těmi nejnáročnějšími velkými scénami s velkým množstvím lidí a spoustou překrytí, a tedy představuje významné vylepšení celého detekčního systému.

K trénování detektoru byla použita speciální databáze termálních snímků, která vznikla spojením oficiálního datasetu od firmy FLIR a vlastní databáze nasnímané pomocí Lepton 3.5 kamer v Národním muzeu v Praze v průbehu několika měsíců. Celkem k trénování modelu, založeném na hluboké neuronové síti, bylo využito 13416 termálních snímků s 53628 oanotovanými osobami.

Po aplikaci detektoru na termální snímek získáme obdélníky ohraničující detekované osoby. Tyto obdélníky se využijí v dalším kroku – lokalizaci objektů ve známé scéně. Metoda lokalizace je založena na rekonstrukci scény zpětnou projekcí obrazových souřadnic do 3D modelu 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. 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. Způsob projekce obrazových bodů do modelu scény je matematicky přesný a nemění se od původního bakalářského projektu.

Naproti tomu softwarová implementace abstrakce modelované scény byla přeprogramována a výrazně vylepšena. Původní model podporoval pouze jednu obdélníkovou scénu s jednou kamerou, kde korespondujicí body mezi termálním obrazem a scénou musely být zadány manuálně. Novou implementaci lze nalézt v druhém Python3 balíčku `ThermoDetection` v druhém repozitáři projektu `thermo-person-detection`. Scéna je uložena v `JSON` konfiguračním souboru, ve kterém lze vyjádřit libovolný počet polygonálních hranic scény se jménem a barvou pro účely zobrazování. V souboru jsou taktéž uloženy všechny kamery se jmény, barvami, IP adresami a polohami v 3D prostoru scény. Druhý repozitář dále obsahuje vizuální kalibrátor scény, který zobrazí buď živý nebo statický pohled z kamery a nabídne uživateli vybrat významné body přímo v termálním obrazu a zadat jejich 3D souřadnice. Tento nástroj poté všechny takto vytvořené mapující body uloží do konfiguračního souboru scény a při každém načtení scény jsou tyto body použity pro vypočítání pozice kamer ve scéně a vytvoření projekčních matic pro zpětnou projekci obrazových bodů do scény.

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

Závěrečná část práce popisuje typické kroky při instalaci nových termálních jednotek v novém prostředí sloužící taktéž jako návod a reálné testování výsledného detekčního a lokalizačního systému v Národním muzeu v Praze, kde byla v rámci vzájemné spolupráce v průběhu několika měsíců vytvořena rozsáhlá databáze termálních snímků z probíhající

expozice. Tato databáze byla použita jak k trénování nového detektoru, tak následně k analýze pohybu návštěvníků prostory expozice za pomoci nového detekčního a lokalizačního systému. Cílem analýzy bylo vytvořit mapy nejčastějšího výskytu návštěvníků, které by pomohly odhalit ty části expozice, které návštěvníci považují za nejzajímavější.

Nejkritičtější částí celého procesu detekce a lokalizace je detekce lidí z termálních snímků a nově použitá metoda výrazně zvyšuje přesnost systému i ve velmi komplexních situacích. Další kritickou částí systému ovlivňující přesnost je lokalizace lidí z detekovaného obdélníku. Zde se nabízí celá řada možných vylepšení. Samotná matematická lokalizace s pomocí projekční matice je velmi přesná, ale významnou roli zde hrají zastíněné osoby, deformace obrazu čočkou, malé rozlišení kamery způsobující až metrovou odchylku na jeden pixel při vzdálenosti 16 metrů od kamery a další. Lze s jistotou říci, že systém pro menší místnosti (do 8 metrů) s nevelkým počtem osob a kamerou umístěnou v horním části místnosti systém pro drtivou většinou případů funguje velmi dobře. Pro komplexní scény se spoustou osob (příkladem může být velká místnost muzea se 17 metry na délku a 50 osobami) dává smysl uvažovat další rozšíření systému pro zvýšení přesnosti. Například použití stereovize, u níž by se kamery shodly na detekovaných osobách, by vedlo k významnému zpřesnění nalezené lokace osob. Jiným zajímavým rozšířením by mohlo být sledování objektů například pomocí nového algoritmu DeepSORT.

System for People Detection and Localization Using Thermal Imaging Cameras

Declaration

I hereby declare that this Master's thesis was prepared as an original work by the author under the supervision of prof. Ing. Dipl.-Ing. Martin Drahanský Ph.D. and Prof. (FH) Univ.-Doz. Mag. Dr. habil. Guido Kempter within the scope of international cooperation at the University of Applied Sciences Vorarlberg. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením prof. Ing. Dipl.-Ing. Martina Drahanského Ph.D. a Prof. (FH) Univ.-Doz. Mag. Dr. habil. Guido Kemptera v rámci zahraniční spolupráce na Fachhochschule Vorarlberg. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Michal Charvát
July 30, 2020

Acknowledgements

I would like to thank professor Martin Drahanský for supervising this thesis, for his relaxed yet constructive approach, for his selflessness every time I needed advice or help, for acquiring necessary expensive hardware components, without which the continuation of the research would not have been possible, and last but not least, for his extensive contact list that made the international cooperation with the University of Applied Sciences Vorarlberg possible. Furthermore, I would like to thank professor Guido Kempter and also professor Martin Drahanský for arranging the student mobility at the University of Applied Sciences Vorarlberg, where the work on this project could continue. I am also grateful to my family and my sunshine girlfriend for immense moral support and to all the ever so caring staff at the FH Vorarlberg for their kindness and willingness to help in every situation, especially during the COVID-19 pandemic.

Contents

Chapter 1

Introduction

This thesis deals with the follow-up project to the bachelor's project [18] with the topic of *Detection of People in Room Using Low-Cost Thermal Imaging Camera*, which dealt with utilizing a single low-cost thermal camera module, a small single-board computer and image processing to solve the problem of how to detect and locate people in a known environment.

The problem of people detection and localization has found its usage in many areas of everyday life. It is often used for queue management in shops, people flow analysis in museums, or 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 railway stations by guarding hazard zones.

Using a small thermal camera module eliminates the possibility of person and/or face recognition while preserving the functionality of detecting and even locating people. The solution to the problem of people detection based on thermal imaging is therefore a viable option for places where privacy plays an important role.

The bachelor's project finished with:

- a working capture system composed of a Lepton 3 camera and an Orange Pi PC2 single-board computer,
- a C++ capture library allowing to read thermal images from the camera in both raw format and false color over SPI interface,
- a Python control script allowing to issue commands to the camera over I^2C in order to change a color palette, format, control automatic gain (AGC), perform flat-field-correction (FFC) and other,
- Python scripts for person detection and single-camera rectangular scene abstraction allowing for reverse-projecting image points (of detected people) into a 3D scene model.

Several imperfections and improvement proposals had to be addressed in order to create a rigid marketable product with clear deployment instructions and a friendly interface.

The capture system was extremely fragile since the camera and the Orange Pi PC2 computer were held together only with wires on a breadboard making it impossible to carry around, bring to presentations or offer it as a product. The Lepton 3 thermal camera had a bad habit of seizing indefinitely and did not support true radiometry, meaning, that

the user had to supply a conversion function in order to obtain pixel temperatures. The conversion function, unfortunately, allowed for a large temperature error.

The capture library had been designed to work on the small host computer, which would pull thermal frames from the camera and push them into a local virtual video device. Then, the idea was to use a tool like `ffmpeg` to stream the thermal video into a single station with more computational power for processing. This turned out to be impossible to achieve for the raw Y14 thermal video, as no lossless codec and format combination would allow transmitting such stream. Furthermore, the scene abstraction software supported modeling of only simple rectangular areas with a single camera, and most importantly, the thermal detector based on simple image processing has been tested in a large complex environment with a generous number of people, which created all sorts of partial occlusions and overlaps. In this environment the detector did not perform well at all. Deploying it anywhere beyond simple small rooms with up to two persons would probably lead to a critical failure of the detection system.

The objective of this project was to continue the research trying to eliminate flaws listed above and implement all suggested improvements in order to build a robust, salable and easy-to-install detection system, that allows for modeling complex scenes with multiple thermal cameras, minimal possibility of false detection and accurate placement of the detected objects into the scene.

This thesis describes in detail all improvements done to this project, which can be summarized into:

- upgrading the thermal camera to Lepton 3.5, which supports true radiometry,
- changing the processing computer to the standard Raspberry Pi 3B+ with the Raspbian operating system to avoid many hidden pitfalls of the Armbian developed by a small community,
- configuring an automated deployment tool allowing for an easy installation of all required dependencies and configuration of the new Raspberry Pi 3B+ computer to be used inside the thermal unit,
- designing and 3D-printing an enclosure box for the thermal unit consisting of the Raspberry Pi 3B+ computer and the Lepton 3.5 thermal camera,
- designing and constructing an additional camera-control printed circuit board allowing for cutting off the power to the camera remotely in case the camera freezes,
- redesigning and reimplementing the `v4l2lepton3` capture and control library so that it supports the new Lepton 3.5 thermal camera, allows for streaming of the raw thermal video over the network and is significantly faster in order to avoid desynchronization issues, which are even harder to avoid with the Lepton 3.5,
- improving the scene abstraction so that it can model complex boundaries with multiple cameras in a single scene,
- adding a tool for an easy visual calibration of every camera in a new scene,
- and most importantly, capturing an extensive database of thermal images and creating a new rigid person detector with the use of the state-of-the-art YOLO real-time object detector based on deep neural networks trained on the thermal dataset.

The thesis devotes the first chapter to explain once again possible use cases of person detection and localization systems, how the thermal solution differs from other solutions and what are its benefits. The next chapter describes in detail the newly constructed *thermal unit* which consists of the upgraded Lepton 3.5, Raspberry Pi 3B+ and a custom control circuit board, all encapsulated in a 3D-printed enclosure box. Then, the thesis goes through the new design and implementation of the control and capture library `v4l2lepton3` based on a server-client model, following with the new person detector principle, the process of its training, and finally, comparison with the old simple thermal detector. The next chapter reminds the mathematics behind reverse-projecting image points into a 3D scene model and describes the new extended implementation of the scene abstraction, that allows modeling of complex scenes with multiple cameras. The last chapter finishes with explaining the typical steps for deployment of the detection and localization system in a new environment and summarizes results obtained during a real-world deployment in the Czech National Museum for the purpose of building heatmaps of people movement throughout exhibition premises.

The end result of the project should be an easily deployable person detection and localization system based on thermal imaging that supports multiple cameras and can serve as an input for other systems that take actions by knowing positions of people in the monitored environment. For example alerting security staff, analyzing flow of people for marketing purposes or controlling the environment in smart homes.

Chapter 2

Utilization of people detection and current technology comparison

This chapter is split into two sections. The first section deals with the usage of people detection in general and the second one compares currently used technologies in people counting/detecting/locating disciplines with respect to their use case. The theory in this chapter is common for both bachelor's and this project and has been mostly taken over from the bachelor's thesis [18].

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 describes some of the most important areas where these techniques are used.

Mercantile interests

In marketing, it is possible to determine the best placement of a product based on a model constructed with the data of customer movement. From people detection systems, it is possible to extract interesting information, such as in which areas customers spend the majority of time or what path they tend to take in a particular environment (shop, supermarket). The detection and localization system provides us, in this case, with data we can analyze, and based on that for example, optimize the spot for 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 lengths in real time, measure an average wait time to be served or staff idle time. The data from the system might serve to improve customer experience and manage resources of the facility more efficiently. For supermarkets, number of open desks can vary over time based on actual queue lengths, and store personnel can be distributed more efficiently or reallocated on the fly.

Public transport

Similarly to the previous paragraph, detection and counting techniques are vastly used at airports, in subways, 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 maintaining user friendly environment and can start an initiative for improvement. We count the number of passengers being transported, how full a train or bus gets during which hour of the day, and as with the queue management, we measure time delays of for example passengers getting in and out of a vehicle.

Guarding dangerous zones

One of the most important areas, where detection of living beings is vital, is undoubtedly ensuring workplace safety or securing dangerous zones. This might include hazardous areas at train or subway stations near railway tracks, in heavy machinery, industry halls near dangerous machines, where no living being should be present at normal circumstances. In these cases, the people detection mechanism may alert responsible personnel and prevent severe accidents from happening.

Smart homes

Another usage of systems for detection and localization of people can be in smart homes. Nowadays, everything from light bulbs, microwave ovens to washing machines can connect to the network and interact with other devices and humans as well. By getting information about the presence, location and/or pose of people, the detection system is capable of commanding these smart devices, i.e., controlling the environment around the people. This can help elderly or disabled people with everyday life by turning machines on or disabling them if there is a high chance of them forgetting to do so. In smart-home environments, there is a possibility to utilize counting, detecting and also locating people techniques. [11] [31]

2.2 Currently available technologies allowing for people detection

Some of the current technologies being used for purposes of counting, detecting or locating people might include the following:

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

[1]LIDAR (Light Detection And Ranging) – method for measuring distances (ranging) by illuminating the target with laser light and measuring the reflection with a sensor. https://en.wikipedia.org/wiki/Lidar

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

IR/Laser beam interruption

Infrared/laser beam interruption technique can be used only for counting people passing through a narrow passage, for example a doorway. A transmitter device is installed on one side of the narrow passage and a receiver on the other as illustrated in figure 2.1. The two devices are connected together and form an invisible barrier of light. When an object breaks the connection between transmitter and receiver, the system registers plus one count. This general solution yields inaccurate results in case more people are allowed to pass 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 right between the sensors. [53] In order to achieve even higher success rate, we might consider using for example a LIDAR based solution.

Figure 2.1: Infrared/laser barrier sensor for counting people passing through. (Source: [42].)

LIDAR

Figure 2.2 demonstrates the usage of a LIDAR device. A LIDAR-based detection device consists of only a single sensor usually placed above a passage. The sensor acts a transmitter and a receiver at the same time. The device casts laser beams into several directions and measures precisely the time required for each reflected beam to get back into the sensor. This way it is possible to calculate the distance each laser beam has traveled, and therefore create a depth map revealing objects in its field of view.

The LIDAR approach makes counting people in a doorway way more accurate than using the beam interruption method. This type of detection system can deduct the direction of the passing object, and since the sensor is usually installed above the passage, it has no problem with detecting multiple people passing next to each other, which was a major problem with the infrared/laser barrier. This solution is widely used at airports or in sliding doors opening mechanisms and has a guaranteed accuracy over 95 %. [9] [13]

Figure 2.2: LIDAR-based device for counting people entering and leaving through a doorway. (Source: [9].)

GPS/Wi-Fi/Bluetooth smart device tracking

With the growth of smart devices supporting wireless technologies such as Wi-Fi or Bluetooth, a new method of tracking people has been adopted. Devices with these technologies make it easy to triangulate their position. Obviously not every living being needs to carry a smart device at all times, however if they do, their smart device offers valuable information about their owner's whereabouts. Although it can not be reliably used to detect people, as we would get plenty of false negatives, the 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 *heatmap*[2], representing the duration of people's presence throughout a shop, is depicted in figure 2.4.

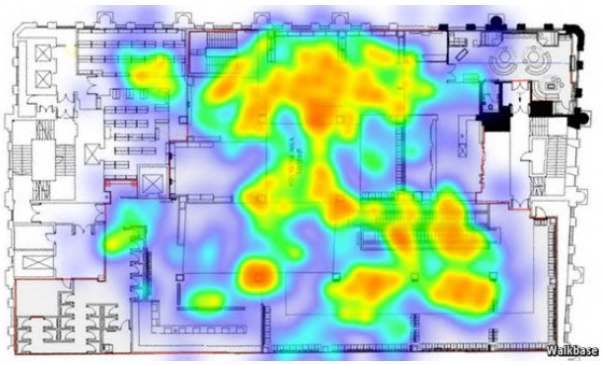


Figure 2.3: Intensity map representing duration of people's presence at a certain location. (Source: [58].)

[2] A heat map (or heatmap) is a data visualization technique that shows magnitude of a phenomenon as color in two dimensions, giving obvious visual cues to the reader about how the phenomenon is clustered or varies over space. https://en.wikipedia.org/wiki/Heat_map

Projecting structured light

The method of projecting structured light is not used directly for purposes of detecting or locating people, but more often for obtaining 3D models of relatively small objects or continuous depth maps.

The system usually consists of 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. The LIDAR sensors cast rays only into several directions. By projecting the light pattern, the device covers a continuous area and has a potential to be more accurate. As with LIDAR devices, this sensor would be 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 extremely precise calibration, is bigger in size and more complex.

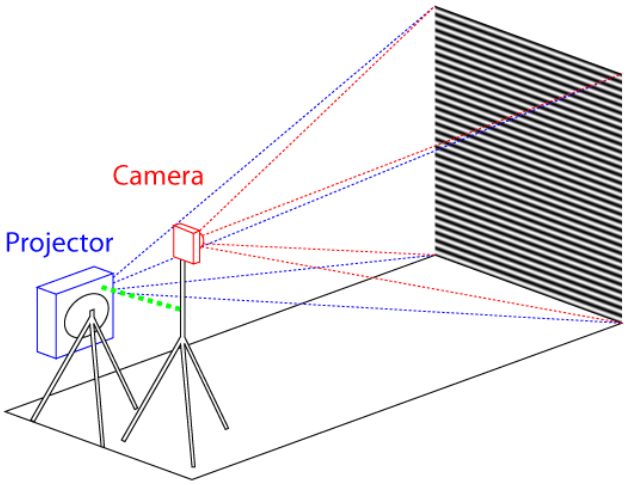


Figure 2.4: Example setup of a depth mapping system composed of a single camera and a projector casting structured light. (Source: [35].)

3D stereo vision

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 shifted images. By combining video frames from two cameras, the system constructs the depth dimension of the scene, which generally provides accurate object-to-camera distance measurement.

This approach can be seen in cutting edge solutions designed for airports, big train stations, where accurate detection and localization is crucial. The current solutions using this technology tend to be very expensive, due to their complexity and calibration requirements.

When compared with the LIDAR and structured light approach for counting people, this technology has one drawback. It has much narrower field of view, which means it is

challenging to find a proper location for installing the stereoscopic device at places with a low ceiling. Another drawback, which also applies to the monocular video analysis, is privacy issues. With the use of regular cameras for purposes of detecting, locating or counting people, it is also 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. Nowadays 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 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 the histogram of oriented gradients (HOG) [19] and so on.

3. **Final stage of the analysis** – the actual algorithm used to achieve the goal of video or image analysis. The goal might be object detection, classification, recognition or 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. [40] Firstly, we need only a single camera, so there is no need for extremely precise hardware calibration of the system, 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 is however the fact that by using a thermal camera with small image resolution, it is *impossible* to perform facial or person recognition. This makes this approach more suitable for places, where privacy plays an important role, e.g. at workplaces or homes. Thermal images are not dependent on lighting conditions of the scene, which makes the system very effective during the night. The largest disadvantage of monocular image analysis is the missing depth dimension. Low camera resolution with the missing depth dimension cause locations of the detected objects to be only a rough approximate, as the missing dimension has to be estimated based on some assumption like that the object is touching ground.

The idea behind detection and localization of people using a thermal camera is to capture a thermal image, detect objects corresponding to people in the image and estimate locations of each object in a model of the scene using perspective projection.

Chapter 3

Thermal capture unit

This chapter describes the thermal capture unit, which can be used for standalone or remote capture of thermal data. The unit can be placed anywhere with electric and network connection and consists of FLIR's Lepton 3.5 [5] thermal camera module, a custom PCB with a circuit controlling the thermal module and a Raspberry Pi 3B+ single-board computer, which communicates directly with the camera and provides the user with higher level access. All these three parts are enclosed in a custom-designed 3D-printed enclosure box. The following sections go in detail through each part of the thermal capture unit.

3.1 FLIR Lepton 3.5

In previous experiments and the bachelor's project [18], the earlier version of the thermal module – Lepton 3 – has been used. In this project, the thermal camera module has been replaced with version 3.5. This section describes the Lepton 3.5 with all differences between the previous version of the thermal camera as well as all necessary adjustments to the capture and control library `v4l2lepton3` designed to communicate with the camera. An in-depth documentation of all new features in the `v4l2lepton3` library can be then found in chapter 4.

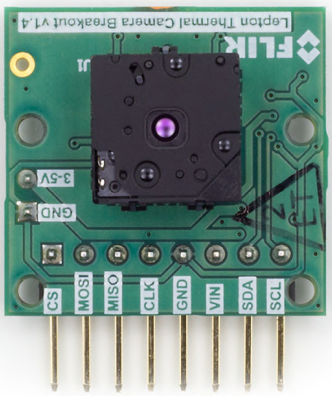


Figure 3.1: Lepton 3/3.5 with breakout board. (Source: [25].)

3.1.1 Lepton 3.5 specifications

In this project we use a Lepton 3.5 thermal camera module[3] made by the company FLIR[4] which is currently one of the leading manufactures of thermal camera solutions. The camera contains a sensor sensitive to long wave infrared (LWIR) light in range from 8 to 14 μm. The camera module is smaller than a dime and provides images with decent resolution of 160 by 120 pixels. The effective frame rate of the camera is only 8.7 Hz, however for our needs, it is sufficient. The camera only requires low voltage supply and has small power consumption of around 160 mW. See table 3.1 for more detailed specifications.

For better manipulation with the camera module we use a breakout board (figure 3.7) with a housing for the Lepton camera module. Lepton 3 and 3.5 both have the exact same dimensions and pinout, therefore both versions fit into the same breakout board. The breakout board provides better physical accessibility, improves heat dissipation and increases the input voltage supply range to 3-5 V using its built-in regulated power supply. This power supply provides the camera module with three necessary voltages: 1.2, 2.8 and 2.8-3.1 V. The breakout board also supplies the camera with master clock signal. [25] The camera uses two interfaces for communication:

- **SPI** for transferring video frames from the camera to the SPI master device.
- **I²C** for receiving control commands from the I²C 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 imaging market. The Lepton 3.5 thermal camera module can be considered low-cost when compared to other thermal camera devices available, as it costs around $230 (2020[5]). This could however be considerably more expensive when compared to other *non-thermal* solutions, for example when a simple infrared counting sensor is used.

Spectral range	8 to 14 μm
Array format	160 × 120 pixels
Pixel size	12 μm
Thermal sensitivity	<50 mK
Temperature range	−10 to +400°C
FOV horizontal	56°
FOV diagonal	71°
Depth of field	28 cm to ∞
Lens type	f/1.1 silicon doublet
Output format	16-bit Y16 raw temperature 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	160 mW operating, 5 mW shutdown mode, 800 mW shutter event
Dimensions	11.8 × 12.7 × 7.2 mm

Table 3.1: Lepton 3.5 camera module specifications. (Source [4].)

[3] FLIR Lepton homepage https://lepton.flir.com/
[4] FLIR homepage http://www.flir.eu/
[5] FLIR Lepton 3.5 supplier e-shop https://groupgets.com/manufacturers/flir/products/lepton-3-5

Table 3.1 summarizes the parameters of the new 3.5 version of the Lepton camera. From the version 3.0 it differs among others in temperature range and power consumption. The camera has increased temperature range and consumes a little bit more power during shutter events.

3.1.2 Lepton 3 vs 3.5 - false/true radiometry

The previously used version 3.0 is a non-radiometric camera or so-called *false radiometric*. It does have an option and a command for turning the radiometric feature on, however, this feature only ensures that the output values from the sensor stay the same when the internal temperature of the camera changes. This behavior is depicted in figure 3.2. This way one can be sure that for any given temperature of a pixel, there is a specific value on the output no matter the internal condition of the camera. It has false radiometry because it does not provide the function for converting incident flux of a pixel into the real temperature in Kelvin. This nonlinear function has to be supplied by the user.

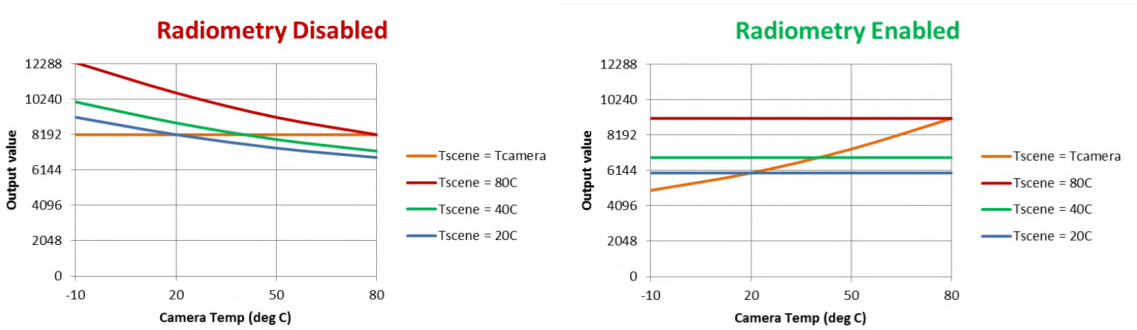


Figure 3.2: Hypothetical illustration of camera output vs camera temperature. (Source: [26-27].)

The mapping function is usually an output of a calibration process which involves an accurate spot thermometer. Mapping points between pixel values of the Lepton 3 module and actual temperatures taken by the thermometer are constructed over a wider temperature range and used for interpolation in order to obtain the calibrated mapping function. The documentation [4] mentions that the mapping function is close to linear but not completely.

The Lepton 3 camera module has been replaced by the newest version 3.5 mostly because of its *true radiometry* capability. The conversion function is provided internally and applied by the camera firmware, so that the output of the camera already contains the final temperature of each pixel in Kelvin.

The real temperature coming from Lepton 3.5 is scaled by a scalar, that can be set to either 100 or 10. To get the temperature from a pixel in Kelvin, the 16-bit integer pixel value needs to be divided by the scalar. See conversion examples in the table 3.2.

Note that the true radiometric mode is essential for the problem of people detection because knowing the exact temperature of a scene allows us to filter out regions of the scene that have temperatures outside the human-body-temperature range, which can simplify the computation. Alternatively, we can reject a potential detected object (person) after checking its temperature distribution.

16 bit pixel value	Scalar 10		Scalar 100	
	$[K]$	$[°C]$	$[K]$	$[°C]$
$16,000$	$1,600.0$	$1,326.85$	160.0	-113.15
$65,535$	$6,553.5$	$6,280.35$	655.35	328.2
$30,315$	$3,031.5$	$2,758.35$	303.15	30

Table 3.2: Lepton 3.5 16-bit pixel value conversion example using true radiometry for given resolutions (scalars).

3.1.3 Lepton 3.5 control protocol (CCI)

The *command and control interface* (CCI) is used to control the camera and is hosted on a two-wire interface (TWI), which is almost identical to I^2C. The protocol had been implemented in Python and supported essential commands for the Lepton 3 thermal camera.

For the Lepton 3.5 version, the protocol itself did not change, however, some new commands have become available – mostly connected with the new true radiometry feature. These commands had to be included into the control software. Both Lepton 3 and 3.5 have the same command `RAD_ENABLE` to enable or disable the *false* radiometric feature. The Lepton 3.5 comes with a set of three new commands setting the `TLINEAR` feature which corresponds with the true radiometry.

If the `TLINEAR` feature is left off while having radiometry enabled, the camera behaves like Lepton 3 with the false radiometry. The `TLINEAR` option enables the inner calibrated mapping function for converting the incident flux into real temperature in Kelvin, which the camera then returns for each pixel.

For Lepton 3.5, the following three essential commands have been added to the control software with respect to the true radiometry:

- `RAD_TLINEAR_ENABLE`: *tlinear* is the name of the feature that enables the incident flux to temperature conversion, in other words, makes *true* from *false* radiometry, as mentioned in subsection 3.1.2.
- `RAD_TLINEAR_SCALE`: this command sets or gets the scalar for the pixel values in Kelvin. The scalar may be set to 100 or 10.
- `RAD_TLINEAR_AUTO_SCALE`: turns on or off an automatic scalar selection between 100 and 10, based on the observed temperatures in the scene, which effectively increases or decreases resolution as well as measurement error.

An `OEM_REBOOT` command has also been included among others, as it turned out that issuing a *reboot* command over the I^2C interface deals with the camera seizing problem in the least invasive way. Though, it does not always work, as is mentioned later in section 3.2.

Furthermore, the previous implementation contained plenty of redundancy when adding a new command. Each command may have `GET`, `SET` or `RUN` method (in most cases at least two), plus for a `SET` command, several options are usually available. In the old implementation, with each newly added command, it was necessary to create a function for each available method and `SET` option. This redundancy and inelegance of the process of adding new commands have been addressed and as a part of the `v4l2lepton3` capture and control library. The camera control software comes reimplemented almost from scratch.

The implementation details with a listing of all currently available commands can be found in chapter 4.

3.1.4 Lepton 3.5 video transfer protocol (VoSPI)

The *Video Over SPI* (VoSPI) protocol did not change between version 3.0 and 3.5 of the Lepton camera. From this point of view, the code pulling frames out of the thermal camera should work without any change also with the version 3.5. This, however, was not the case, as there were major synchronization issues. Desynchronized camera is one of the biggest problems of Lepton modules reported by the community. The controlling computer or the code running on it is simply not fast enough to read out all packets from the camera in time, which causes the camera to reset. As a result, it is not possible to read even a single frame out of the camera. From the packet numbering, I calculated that in my case, the reading process would have to be **8 times** faster to run smoothly with the old implementation.

Also the new use case of the project requires collecting continuous video feeds from multiple cameras and processing them in a single place. This leads to a completely different architecture, therefore, the capture library had to be redesigned and sped up significantly. The new version of the `v4l2lepton3` library is described in detail in chapter 4.

3.2 Thermal camera control PCB

The Lepton 3 thermal camera module used to have one really bad habit – from time to time it would seize operation indefinitely. The camera stops sending frames over SPI and any attempt to read data from it results in a timeout. In such state, the camera also does not respond to any commands over I^2C claiming it is busy or not ready to receive any new command. Unfortunately, this problem also occurs in the 3.5 variant of the camera module, however not that often. In most cases, even though the camera does not return nor set any control registers and claims to be busy, the *reboot* command does reboot the camera completely fixing the broken state.

There are situations, however, when not even the reboot command works, and the camera gets in a state in which it simply can not be used anymore. This poses a real problem because the system is meant to be working remotely and physically unplugging and plugging the camera back in is not plausible. It was essential to provide a workaround in order to be able to manage the thermal capture system remotely. The camera used to be powered directly from the host computer. Being able to remotely disconnect the camera from the circuit would force a cold boot of the camera and would also come in handy when the camera is not needed. Turning the camera off when it is not needed would prolong its lifespan, save power and eliminate the clicking shutter noise.

In order to solve this problem, a custom power switch has been designed and built on a printed circuit board that allows for disconnecting the power from the camera and then reapplying it remotely. That effectively forces the camera to perform a full reboot.

3.2.1 The first design of the control circuit

The first design of the circuit contained an N-channel MOSFET transistor with a pull-down resistor on its gate that was controlled by a digital signal coming from a GPIO pin of the Raspberry Pi. The circuit also contained a couple of capacitors with values of 100 μF and

100 nF to improve power supply stability. The plan was to use Raspberry Pi's GPIO to apply 5 V to the camera through the transistor. The first design can be seen in figure 3.3.

The first design however proved itself not to be functioning properly because *log1* of 3.3 V coming from the Raspberry Pi's GPIO output would not saturate the N-channel MOSFET enough to turn the camera on, despite the typical gate-source threshold voltage of the IRFZ44N MOSFET [2] of 2-4 V.

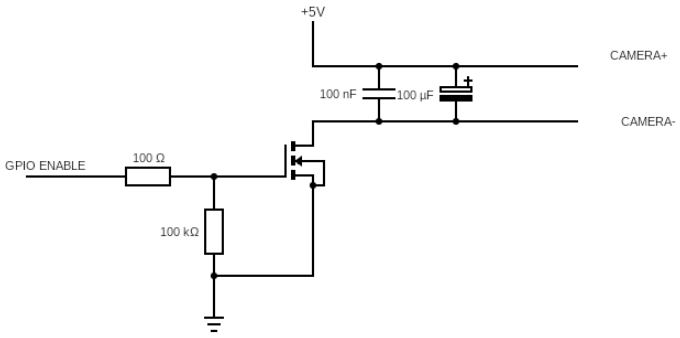


Figure 3.3: The first design of the MOSFET switch circuit.

3.2.2 The second design of the control circuit

The second design contained a standard power switch with the use of P-channel MOSFET and a second transistor – this time an NPN bipolar transistor BC337 [3] that is current-driven, and thus has no problem operating at the 3.3-V level. The bipolar transistor grounds the gate of the IRF9Z34N MOSFET [1], which then applies 5 V to the thermal camera supply rail. The second design was successful, and with its help, it is possible to turn the camera's power supply on or off remotely using a Raspberry Pi's GPIO pin. The second design can be seen in figure 3.4 and the physical custom PCB[6] in figure 3.5.

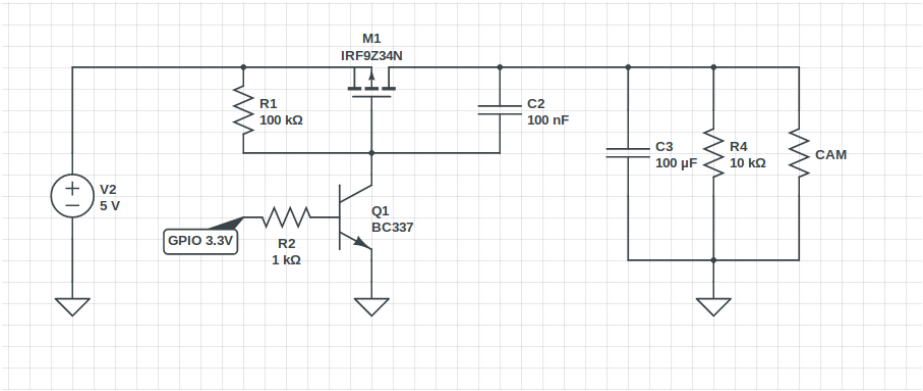


Figure 3.4: The second design of the MOSFET switch circuit.

[6]PCB – printed circuit board

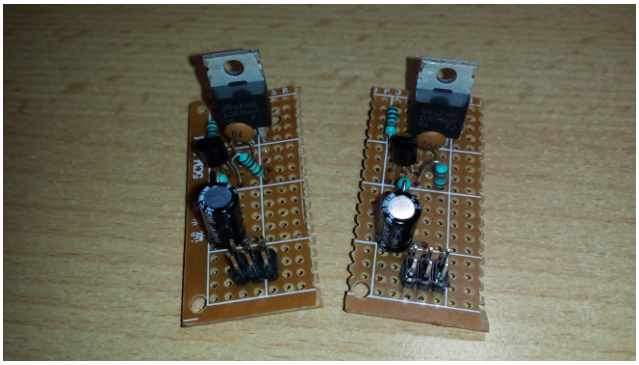


Figure 3.5: PCB of the second design of the MOSFET switch circuit.

3.2.3 GPIO control library *WiringPi*

For controlling GPIO pins of the Raspberry Pi computer, we are using the `wiringPi` library [28]. The standard Raspbian image has the library `wiringPi` preinstalled, however, on the Raspbian Lite operating system the library needs to be installed first using the following command:

```
$> sudo apt-get install wiringpi
```

This library provides a C API as well as an easy terminal access for controlling the GPIO pins of the Raspberry Pi computer. The API syntax is identical to the Arduino library having functions like `pinMode()` or `digitalWrite()`. An example C code of turning on and off the virtual GPIO pin 15 can be examined in listing 1.

GPIO pins can be controlled directly from the terminal using the `gpio` tool. The mode of pin 15 to `output` can be set using:

```
$> gpio mode 15 out    # 15=GPIO pin, modes=in/out/pwm/clock/up/down/tri
```

Writing *log1* to pin 15 can be done using:

```
$> gpio write 15 1    # 15=GPIO pin, values=1/0
```

```
1  #include <wiringPi.h>
2  #include "unistd.h"
3  #define PIN 15
4
5  int main(int argc, char **argv)
6  {
7          wiringPiSetup();
8          pinMode(PIN, OUTPUT);
9          while(true)
10         {
11                 digitalWrite(PIN, HIGH); sleep(1);
12                 digitalWrite(PIN, LOW);  sleep(1);
13         }
14         return 0;
15 }
```

Listing 1: One second blink example on the virtual GPIO pin 15.

Some GPIO pins are also connected to a special hardware modules of the processor like I^{2}C or SPI. In that case the mode of such pin is displayed as `ALTX` (alternative function X). Modes and current digital values of all available GPIO pins of the Raspberry Pi computer can be read using command `gpio readall` which is demonstrated in figure 3.6.

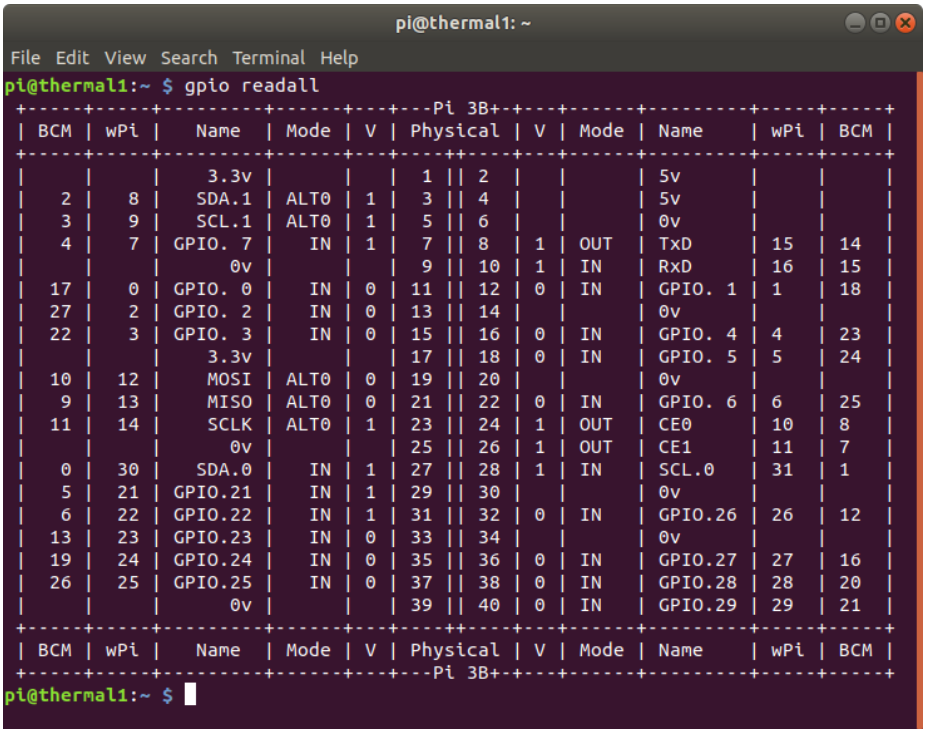


Figure 3.6: Exammple result of `gpio readall` command on a thermal unit.

The custom circuit described in subsection 3.2 allows us to remotely disconnect the camera from the power supply. In our case, the power switch is controlled with the `wPi` virtual GPIO pin 15 visible in figure 3.6. Unfortunately, if the camera is in the corrupted frozen state in which it does not send frames nor can be rebooted using I^{2}C control interface, cutting off its power supply does not do the trick and the camera does not reboot. Even though the switch transistor is fully off, there is still around 2 V on the camera power rail, which most likely causes the camera to sustain its internal broken state, and when 5 V is reapplied to the positive rail, the camera keeps not responding.

After the transistor turns off, the camera most likely starts stealing power from SPI or I^{2}C interfaces and its switching power supply keeps the camera's internal voltage from dropping all the way to 0 V. So in order to fully disconnect the camera, we must use the custom transistor switch, disable both interfaces and set all pins connected to the camera to *output log0*. This effectively pulls all pins of the camera to ground and discharges the camera completely leading to a full reset. After re-enabling data interfaces and applying power to the supply pin, the camera boots up normally and starts working as it is supposed to. This procedure needs to be done in around 5 % of the cases when the camera freezes. For the rest, issuing a simple `oem_reboot` command via ICC does the trick.

3.2.4 Segmentation fault on I²C kernel module unload

Another tricky part is disabling and re-enabling the data interfaces. Disabling these interfaces effectively means unloading kernel modules from the linux operating system. At this time, there is unfortunately a known issue[7] with disabling the I²C hardware interface. Around 25 % of attempts to unload the kernel module `i2c_bcm2835` fails on segmentation fault (`NULL` pointer dereferencing) inside the kernel which has catastrophic effect on the Raspberry Pi 3B+, as no other modules can be loaded or unloaded and the Raspbian operating system can not be rebooted because the command `sudo reboot [-f]` hangs indefinitely. The only way to revive the Raspberry Pi from this state is unplugging it and plugging it back in. (hard restart)

A snippet from the terminal output of the segmentation fault during unloading the kernel module using `rmmod` can be seen in listing 2. More information about the fault obtained from kernel message buffer using `dmesg`[8] is depicted in listing 3. The issue has been already fixed[9], however, it is going to take some time until the fix is merged into the official Raspbian kernel. Till then, the safer way is simply to reboot the whole Raspberry Pi straight away. During the reboot, all pins of the Raspberry Pi are, for a brief moment, set to *log0*, which cuts off the power to the camera through the custom transistor switch and the whole camera reboots.

```
pi@thermal3:~ $ sudo rmmod i2c-bcm2835
Message from syslogd@thermal3 at Oct 26 00:31:13 ...
 kernel:[  105.607954] Internal error: Oops: 17 [#1] SMP ARM
(...)
Message from syslogd@thermal3 at Oct 26 00:31:13 ...
 kernel:[  105.697356] Code: e8bd4000 e2504000 089da818 ebfff6c8 (e5943014)
Segmentation fault
```

Listing 2: Terminal snippet of the segmentation fault when unloading I²C kernel module using `rmmod`.

```
[  185.993328] Unable to handle kernel NULL pointer dereference at virtual
↪  address 00000012
[  185.998257] Internal error: Oops: 17 [#1] SMP ARM
[  185.999511] Modules linked in: i2c_bcm2835(-) (...)
[  186.009776] CPU: 1 PID: 790 Comm: rmmod Tainted: G C 4.19.80-v7+ #1274
[  186.013013] Hardware name: BCM2835
[  186.014681] PC is at clk_rate_exclusive_put+0x20/0x5c
[  186.029528] Process rmmod (pid: 790, stack limit = 0x7989f0fb)
```

Listing 3: `dmesg` snippet from after the kernel segmentation fault on unloading I²C kernel module.

[7] Report of the I²C kernel module segmentation fault https://www.raspberrypi.org/forums/viewtopic.php?f=28&t=255294

[8] `dmesg` (display message or driver message) – unix-like command printing the kernel message buffer

[9] [PATCH RFC] i2c: bcm2835: Store pointer to bus clock: Null pointer dereference fix commit https://marc.info/?l=linux-arm-kernel&m=157209334808763&w=2

3.3 Raspberry Pi 3B+

Compared with the bachelor's project [18], we have migrated from Orange Pi PC2 (Armbian) to Raspberry Pi 3B+ (Raspbian). The change has been made to improve portability, increase computational power and avoid some of the pitfalls connected with using cloned Chinese single-board computers with small community and many OS features still waiting to be completed.

It was necessary to reconfigure the whole image from the beginning. The thermal unit is currently using Raspbian Bustler Lite[10] which is a minimal operating image with only 435 MB in size. Since it is a minimal image, it is necessary to install all dependencies and libraries manually. The exact same steps would have to be performed on every thermal unit and repeated for every new unit in the future. Therefore, it only makes sense to use a tool to automatize the steps of preparing the environment on thermal units for running the detection and localization system.

This section describes the configuration and properties of the Raspberry Pi 3B+ computer, which is used to directly communicate with the thermal camera, and is therefore the brain of the thermal unit.

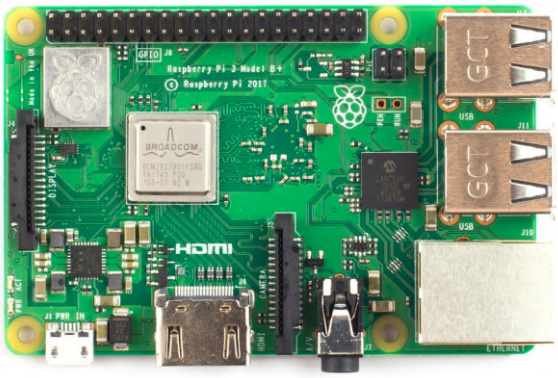


Figure 3.7: Raspberry Pi 3B+ single board computer. (Source: [45].)

3.3.1 Raspberry Pi 3B+ specifications

The Raspberry Pi 3B+ is the last revision of the third generation single-board computer. It is a low-cost, credit-card sized computer capable of performing everything one might expect from a regular desktop computer.

The Raspberry Pi runs a Debian-based operating system Raspbian Bustler Lite. The Raspberry Pi platform has a massive community base, and it is widely used by all sorts of hobbyists for their DIY projects. Its biggest upside is its size and hardware interfaces. The computer has a built-in hardware support for SPI, I^2C, UART, Bluetooth and Wi-Fi communication. Its general purpose input/output (GPIO) pins are also extremely important for interfacing with other electronic devices. Even for its size, the Raspberry Pi has nowadays a quite decent computation power. See table 3.3 with Raspberry Pi 3B+ parameters.

[10] Raspbian official operating system image download page https://www.raspberrypi.org/downloads/raspbian/

CPU	Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit @ 1.4 GHz × 4
RAM	1 GB LPDDR2 SDRAM
Wi-Fi	2.4 GHz and 5 GHz IEEE 802.11.b/g/n/ac
Bluetooth	4.2, BLE
Ethernet	Gigabit, but max 300 Mb/s, PoE support
GPIO	Extended 40-pin header
USB 2.0	4x
Power supply	5 V @ 2.5 A DC
Interfaces	CSI, DSI, SPI, I^2C, UART, stereo output, composite video output

Table 3.3: Raspberry Pi 3B+ specifications. (Source [45].)

3.3.2 Automation with Ansible

For the automatic deployment, it has been decided to use the tool Ansible[11]. It is an agentless tool, which temporarily connects to its targets via `ssh` to perform tasks specified in so-called Ansible *playbooks*. An Ansible playbook specifies actions that should be performed on the target machine. Some of the most common tasks in an Ansible playbook might include:

- managing users
- creating, delete files or directories
- copying a configuration file from local storage to the target
- making sure a dependency/library is installed on the target system
- cloning a git repository and compile a program from sources
- executing shell commands

and a lot more. The advantage of Ansible is in its flexibility. There does not have to be a master service running permanently that would be constantly checking the state of the target devices according to a given configuration like for example when using Puppet[12]. Ansible performs tasks from the playbook only when it is asked to and it works straight out of the box. The only requirement is that the target device is accessible through an `ssh` connection. Ansible will do the rest according to the given Ansible playbook.

3.3.3 Preparing the Raspbian image for the new thermal unit

After writing the Raspbian Bustler Lite image to an SD card using for example the *balenaEtcher* tool[13], the first thing to configure is the `ssh` daemon. The daemon is enabled by creating an empty file `/boot/ssh`.

In case the thermal unit should automatically connect to a wireless network, the `/boot/wpa_supplicant.conf` file shall be edited with the content from listing 4.

[11] Ansible – software provisioning, configuration management, and application-deployment tool https://www.ansible.com/

[12] Puppet – open-source configuration management tool https://puppet.com/

[13] balenaEtcher – free open-source utility for creating live SD cards and USB flash drives https://www.balena.io/etcher/

```
country=CZ
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1
network={
        ssid="the-internet-ssid"
        psk=069c6f3318349...eaf0077fa2f7
}
```

Listing 4: Contents of `/boot/wpa_supplicant.conf` file enabling auto-connection to a prespecified wireless network.

If the thermal unit will be connected to a wired network via an ethernet cable, the Wi-Fi (and Bluetooth) modules can be disabled by adding a lines from listing 5 to `/boot/config.txt` file.

```
dtoverlay=pi3-disable-wifi    # disable Wi-Fi
dtoverlay=pi3-disable-bt      # disable Bluetooth
```

Listing 5: Lines of `/boot/config.txt` file which disable Wi-Fi and Bluetooth modules on boot.

If there will be no DHCP server in the network, it is possible to specify static connection parameters in `/etc/dhcpcd.conf` file. An example code for setting a static IP address to the Raspberry Pi's interface `eth0` can be seen in listing 6.

```
# Example static IP configuration:
interface eth0
static ip_address=192.168.2.205/24
# static ip6_address=fd51:42f8:caae:d92e::ff/64
static routers=192.168.2.101
static domain_name_servers=8.8.8.8
```

Listing 6: Lines of `/etc/dhcpcd.conf` setting up a wired connection without a DHCP server.

At this stage, it is possible to copy an *ssh public key* into `/home/pi/.ssh/authorized_keys` file for easy access via `ssh` using asymmetric cryptography.

3.3.4 Ansible playbook

The knowledge used in this chapter comes from [30] and [26]. After taking care of the configuration on the SD card, it can be inserted into the Raspberry Pi. The computer

should boot up and connect to the network according to the configuration from the previous steps. After that, Ansible can take over. From the master computer, the thermal units are configured by running the Ansible playbook with the following command:

```
$> ansible-playbook -i hosts thermal_deploy.yml
```

The `hosts` file contains information about the target hosts – devices on which Ansible performs tasks specified in the playbook. In the example `hosts` file in listing 7, there is a group `thermal_clients` with 3 devices.

```
[thermal_clients]
th1 ansible_host=192.168.1.31 ansible_user=pi  # UNIT 1
th2 ansible_host=192.168.1.32 ansible_user=pi  # UNIT 2
th3 ansible_host=192.168.1.33 ansible_user=pi  # UNIT 3
```

Listing 7: Example `hosts` file for Ansible with 3 devices.

The playbook `thermal_deploy.yml` itself contains list of tasks to be performed on specified hosts. A snippet from this file can be seen in listing 8. Setting up the environment for thermal units can be, in our case, summarized into the following steps:

1. run `sudo apt update`, `sudo apt upgrade`
2. install the following `apt` packages: *git, python-dev, python3-dev, python-pip, python3-pip, wiringpi, python-opencv, python3-opencv, libboost-all-dev, v4l-utils, v4l2loopback-dkms, v4l2loopback-utils*
3. install python/python3 packages using `pip`/`pip3`: *smbus2, SPI-Py*[14]
4. clone the `v4l2lepton3` library and and compile it
5. run `sudo apt autoclean`, `sudo apt autoremove`
6. run `sudo raspi-config` with arguments to enable SPI and I^2C interfaces

After executing all tasks from the playbook, the thermal unit is ready to operate. If any changes need to be done globally to all deployed thermal units, they can be stated in the Ansible playbook and Ansible will make sure they are applied to all the devices.

[14]SPI-Py library https://github.com/lthiery/SPI-Py.git

```
---
- hosts: thermal_clients
  tasks:
  - name: Running apt upgrade
    become: true
    apt:
      upgrade: yes
      update_cache: yes
      cache_valid_time: 86400

  - name: Install apt packages.
    become: true
    apt:
      name: "{{item}}"
      state: present
    with_items:
        - git
        - python-dev
        - python3-dev
```

Listing 8: Snippet from an Ansible playbook.

3.3.5 Enabling SPI and I²C hardware interfaces

In order to communicate with the camera using SPI and I²C hardware modules on the Raspberry Pi, they need to enabled on the kernel level. On the previously used Orange Pi board, this was a very tedious process that led to decompiling the device tree overlay, manually specifying pins where the hardware modules are connected, recompiling it and ensuring the overlay gets loaded during boot up.

On the Raspberry Pi computer all of this can be done using the `raspi-config` utility, which allows to enable and disable specific modules of the processor using graphical interface or simply by executing the script with arguments as in listing 9. Note that this shell command can also be specified as an Ansible task in a playbook.

```
1 sudo raspi-config nonint do_spi 0  # enable SPI
2 sudo raspi-config nonint do_i2c 0  # enable I2C
3
4 sudo raspi-config nonint do_spi 1  # disable SPI
5 sudo raspi-config nonint do_i2c 1  # disable I2C
```

Listing 9: Enabling SPI and I²C interfaces using a shell command.

As a result, there are corresponding character devices permanently visible in the `/dev` directory and they will also be present after reboot. These devices can be manipulated using `ioctl` system calls. Whether are these devices available in the operating system can be checked using commands in listing 10.

```
pi@raspberrypi:~ $ ls /dev/spi*
/dev/spidev0.0  /dev/spidev0.1
pi@raspberrypi:~ $ ls /dev/i2c*
/dev/i2c-1
```

Listing 10: Demonstration of character devices in the `/dev` directory.

3.4 Thermal unit enclosure

The Lepton camera sends video frames over the SPI interface on 20 MHz which implies that the length of wires connecting the Lepton camera to the Raspberry Pi needs to be as short as possible – maximum of 20 cm in order to provide a stable connection without interference and transmission errors. This condition enforces the Lepton camera and the Raspberry Pi to be physically close to each other. Together, they form a thermal unit.

In order to make the whole thermal unit transferable, protected and professionally looking for quick demonstrations or real life deployment, an enclosure has been designed to fit and mount all of its components—the Raspberry Pi 3B+, the custom camera switch circuit board and the Lepton 3.5 camera. The thermal unit case is composed of two parts – an enclosure box for the Raspberry Pi with the power switch and a camera chassis that is mounted to the top of the first part with a bit of slack that allows the camera chassis to be moved along the horizontal axis. This way it is possible to adjust the viewing angle of the camera using a screw and a rubber band that applies a back force to the camera case against the screw. The unit's enclosure box has hexagonal holes from top and bottom to provide air flow to cool down the Raspberry Pi, as it gets quite hot even with a rather small load.

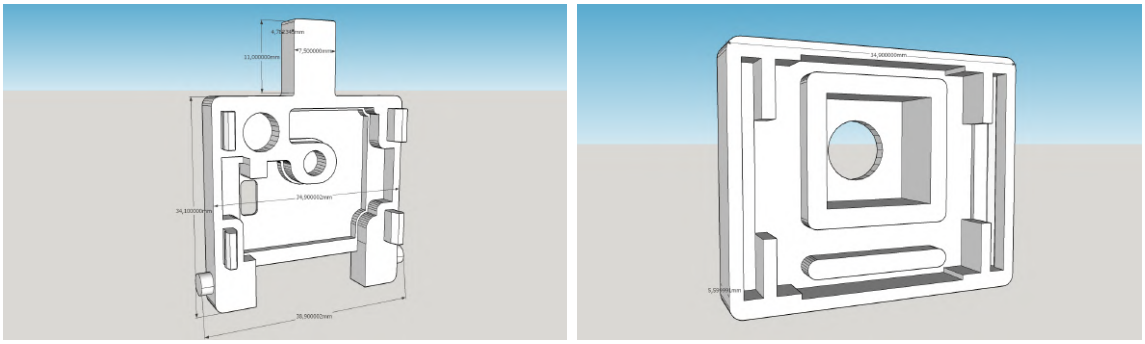


Figure 3.8: Lepton Breakout Case `.stl` model.

The enclosure box was designed in the Sketchup[15] software and exported into the `.stl` format for 3D-printing. The Lepton 3.5 camera chassis model, created by the official Lepton

[15]Sketchup – Trimble design software https://www.sketchup.com/

Breakout Board manufacturer GroupGets[16], has been taken from the portal Thingiverse[17]. The model can be seen in figure 3.8. Two pins were added in the lower part of the back side of the chassis to serve as a pivot points around which the camera could move. The 3D-printed chassis with the Lepton 3.5 camera in a breakout board can be seen in figure 3.9.

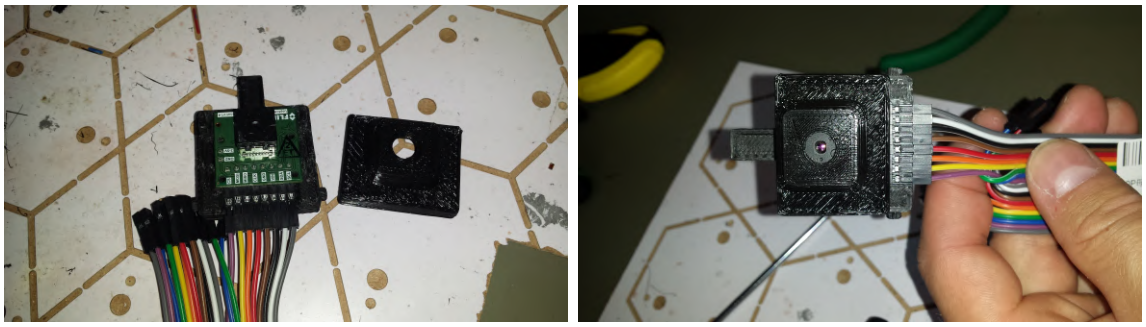


Figure 3.9: 3D-printed case for the Lepton 3.5 in a breakout board.

The unit's enclosure comes from a Raspberry Pi 3B+ `.stl` model[18] also published at Thingiverse. The model has been edited and enlarged to fit the Raspberry Pi, the custom PCB with the transistor switch, the moving camera chassis and all wiring needed to connect all components together. The `.stl` model of the bottom piece of the thermal unit enclosure can be seen in figure 3.10 and the top piece in figure 3.11.

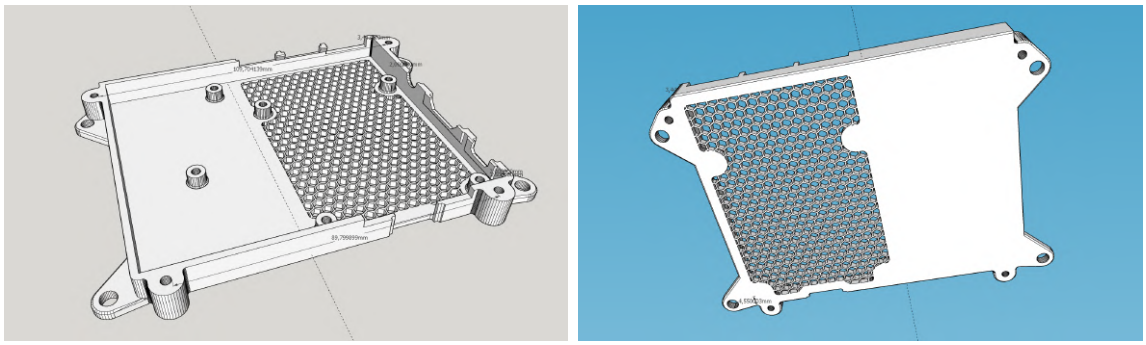


Figure 3.10: Thermal unit enclosure model – bottom.

[16]Lepton Breakout Case `.stl` model https://www.thingiverse.com/thing:1563825

[17]Thingiverse – a website dedicated to the sharing of user-created digital design files https://www.thingiverse.com/

[18]Raspberry Pi 3B+ Case `.stl` model http://www.thingiverse.com/thing:3361218

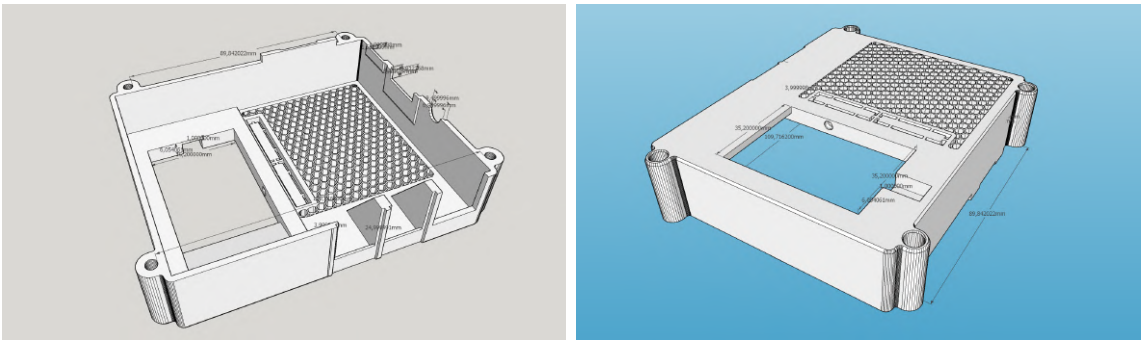


Figure 3.11: Thermal unit enclosure model – top.

The camera chassis with the Lepton camera gets inserted into the hole on the right side of the top piece of the enclosure. The SPI and I^2C interfaces are connected directly to the Raspberry Pi via approximately 10 cm long jumper wires. Two power wires are connected to the custom PCB with the switch, which is mounted in the enclosure underneath the camera, right next to the Raspberry Pi. From the custom PCB, there are three wires going to the Raspberry Pi directly – to *GND*, *5 V* and a virtual *GPIO-15* pin. An assembled thermal unit can be seen in figure 3.12 on the right side, and on the left side, there is a connected thermal unit just without the top cover. Total of four thermal units were constructed for testing purposes.

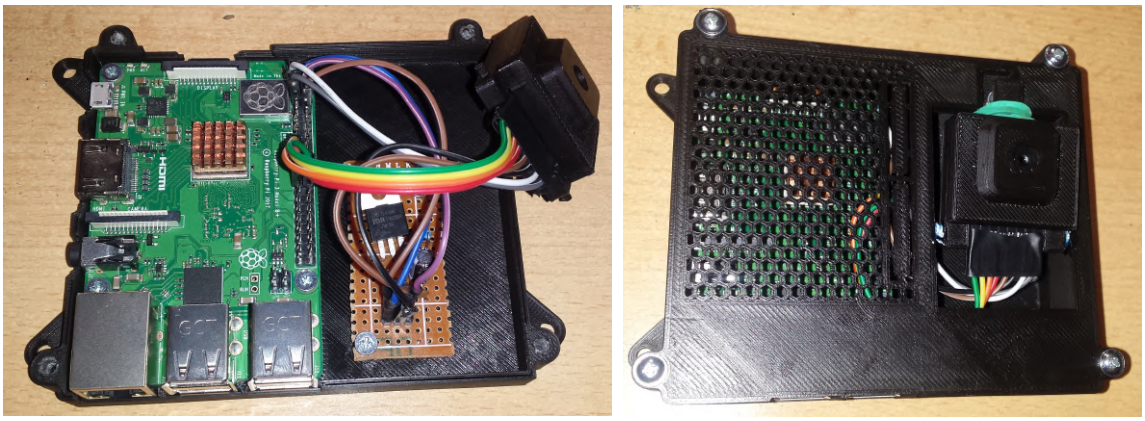


Figure 3.12: A finished thermal unit with Raspberry Pi 3B+, custom power switch and a Lepton 3.5 thermal camera in a chassis.

Chapter 4

v4l2lepton3 capture and control library

The v4l2lepton3 library is the main software part of the project. It is a library that takes care of controlling the camera and retrieving thermal video feed from it. It has been initially presented as a part of the bachelor's thesis [18], however, because of the issues connected with the previous implementation described in subsections 3.1.3 and 3.1.4, it has been redesigned and reimplemented almost from scratch. The library consists of two parts: the C++ application for thermal video manipulation and a Python3 package for camera control and single frame manipulation.

This chapter goes in detail through the capture and control parts of the new version of the library. The old implementation that still relates to the bachelor's thesis can be found in a separate branch[19] of the v4l2lepton3 repository. The new implementation is in the master branch[20] of the same repository.

4.1 Controlling the camera over CCI

The Lepton camera provides a *command and control interface* (CCI) via a two-wire interface almost identical to I^2C with the only difference being that all transactions must be 16 bits in length. Lepton's I^2C address is 0x2A and during communication behaves as a slave device. All Lepton's registers are 16 bits wide. Lepton camera offers 4 control registers and 16 data registers which are all 16-bit wide and are used by the host (master) device to issue commands to the camera. A command is issued by writing and reading particular registers in the camera via I^2C. The exact process is described in the CCI documentation [7]. The protocol has been implemented in the bachelor's project and analyzed in detail in the bachelor's thesis [18] [subsection 3.1.4].

The CCI protocol has not changed from Lepton version 3 to 3.5, however, the implementation of the control library has been redone to remove redundancy and inelegance mentioned in subsection 3.1.3. Furthermore, significantly more commands were added including essential commands for the true radiometry feature. This section explains differences between the old and new implementation, how the redundancy has been removed and which commands the control library currently supports.

[19] v4l2lepton3 git repository – old implementation https://gitlab.com/CharvN/v4l2lepton3/-/tree/bp

[20] v4l2lepton3 git repository – new implementation https://gitlab.com/CharvN/v4l2lepton3

4.1.1 Old implementation of the CCI

As mentioned in subsection 3.1.3, the previous implementation contained plenty of redundancy when adding a new command. An example may be the `VID_PCOLOR_LUT` command, which controls the false color palette used for artificial coloring of the thermal image. The command has two methods – `GET` and `SET`. The value of the property, which the commands sets or gets, ranges from 0 to 8 – each representing a predefined color palette. In the old implementation, this led to having 9 functions for the `SET` method of the command and one other for the `GET` method. The `GET` function also returned a raw value of the currently set palette, which had to be looked up in the Lepton IDD manual [7]. The actual values that can be set or get with the `VID_PCOLOR_LUT` command are listed in table 4.1, and an example of the command functions in the old implementation is depicted in listing 11.

Color palette	Option values in list of bytes
wheel6	[0x00, 0x00]
fusion	[0x00, 0x01]
rainbow	[0x00, 0x02]
glowbow	[0x00, 0x03]
sepia	[0x00, 0x04]
color	[0x00, 0x05]
icefire	[0x00, 0x06]
rain	[0x00, 0x07]
user defined	[0x00, 0x08]

Table 4.1: Options with their respective values in lists of bytes for the `VID_PCOLOR_LUT` command setting false color palette for the Lepton 3/3.5 thermal camera.

```
    def vid_pcolor_lut_get(self):
        return self._command(Lepton3Control.VID, Lepton3Control.VID_PCOLOR_LUT,
        ↪  Lepton3Control.GET, 2)

    def vid_pcolor_lut_set_fusion(self):
        return self._command(Lepton3Control.VID, Lepton3Control.VID_PCOLOR_LUT,
        ↪  Lepton3Control.SET, [0x00, 0x01])

    def vid_pcolor_lut_set_rainbow(self):
        return self._command(Lepton3Control.VID, Lepton3Control.VID_PCOLOR_LUT,
        ↪  Lepton3Control.SET, [0x00, 0x02])

    def vid_pcolor_lut_set_icefire(self):
        return self._command(Lepton3Control.VID, Lepton3Control.VID_PCOLOR_LUT,
        ↪  Lepton3Control.SET, [0x00, 0x06])

    (...)
```

Listing 11: Example of Lepton 3/3.5 color palette command functions for `GET` and `SET` methods with 3 out of 10 options in the old implementation.

This redundancy and inelegance of the process of adding new commands has been addressed, and as a part of the `v4l2lepton3` library, the camera control software comes reimplemented almost from scratch.

4.1.2 New implementation of the CCI

In the new implementation, each command has exactly one definition, which automatically generates allowed methods and contains a translation map for each option that the command can set.

If we take for example the previously mentioned command `VID_PCOLOR_LUT`, in the new implementation, those 10 functions covering this command shrink all the way to 1. The function created according to the command description in the CCI documentation [7] and illustrated in figure 4.1 is depicted in listing 12.

SDK Module ID: VID 0x0300

SDK Command ID: Base 0x04
With Get 0x04
With Set 0x05

SDK Data Length: Get 2 size on an **Enum** data type on a 32-bit machine
Set 2 size on an **Enum** data type on a 32-bit machine

Compatibility	C-SDK Commands	Description
All Lepton Configurations	`LEP_GetVidPcolorLut()`	Updates **`vidPcolorLutPtr`** with the Camera's current video pseudo-color LUT selection.
All Lepton Configurations	`LEP_SetVidPcolorLut()`	Sets Camera's current video pseudo-color LUT selection to **`vidPcolorLut`**

C SDK Interface:

```
LEP_RESULT LEP_GetVidPcolorLut(LEP_CAMERA_PORT_DESC_T_PTR portDescPtr,
                               LEP_PCOLOR_LUT_E_PTR vidPcolorLutPtr)

LEP_RESULT LEP_SetVidPcolorLut(LEP_CAMERA_PORT_DESC_T_PTR portDescPtr,
                               LEP_PCOLOR_LUT_E vidPcolorLut)
/* Video Pseudo-Color LUT Enum
*/
 typedef enum LEP_PCOLOR_LUT_E_TAG
 {
   LEP_VID_WHEEL6_LUT=0,
   LEP_VID_FUSION_LUT,
   LEP_VID_RAINBOW_LUT,
   LEP_VID_GLOBOW_LUT,
   LEP_VID_SEPIA_LUT,
   LEP_VID_COLOR_LUT,
   LEP_VID_ICE_FIRE_LUT,
   LEP_VID_RAIN_LUT,
   LEP_VID_USER_LUT,
   LEP_VID_END_PCOLOR_LUT

 }LEP_PCOLOR_LUT_E, *LEP_PCOLOR_LUT_E_PTR;
```

Figure 4.1: `VID_PCOLOR_LUT` command description in the CCI documentation [7] (page 72).

Constructing the command object from the CCI documentation

From the part of the documentation in figure 4.1, the following information for the `VID_PCOLOR_LUT` command is extracted:

- The command addresses the `VID` hardware module of the camera with the 16-bit address of `0x0300`.
- The command ID is `0x04` (base).
- The command has two available methods: `GET` and `SET`.
- There are 9 values ranging from 0 to 8, which can be set using the command.

Using the extracted information, a single object representing the command with automatically generated underlying methods is constructed. The object is depicted in listing 12.

```
class Lepton3Control(object):
    (...)
    COMMANDS: Dict[str, Lepton3Command] = {
        'vid_pcolor_lut': Lepton3GetSetCommand(Lepton3Command.Module.VID, 0x04,
            {
                'wheel6':  [0x00, 0x00],
                'fusion':  [0x00, 0x01],
                'rainbow': [0x00, 0x02],
                'glowbow': [0x00, 0x03],
                'sepia':   [0x00, 0x04],
                'color':   [0x00, 0x05],
                'icefire': [0x00, 0x06],
                'rain':    [0x00, 0x07],
                'user_defined':    [0x00, 0x08]
            }),
        (...)
    }
```

Listing 12: Definition of the `VID_PCOLOR_LUT` command in the new implementation of the `v4l2lepton3` library.

CCI implementation in v4l2lepton3 Python3 package

The control software is implemented in the `v4l2lepton3.control` Python3 module as a part of the `v4l2lepton3` Python3 package available in the git repository[21]. The module contains `Lepton3Control class`, which can be imported to a custom project or inherited from and extended with more commands. The module also includes addresses for I^2C registers and command hardware modules as well as base classes for general commands, which eases their definition. The following list summarizes all important entities in the `v4l2lepton3.control` module, which is the control part of the `v4l2lepton3` library. They may be imported and utilized in any other project that requires controlling of the Lepton 3/3.5 thermal camera.

[21] `v4l2lepton3` control and capture library git repository https://gitlab.com/CharvN/v4l2lepton3

- `Lepton3Command` – base class defining a Lepton command. Holds supported methods, command id, command module (address of the hardware module that the command is issued to), length in bytes of the command's parameter and parameter mappings.
- `Lepton3GetCommand` – derived class from `Lepton3Command`, supports only the `GET` method.
- `Lepton3GetSetCommand` – derived class from `Lepton3GetCommand`, this class adds the `SET` method.
- `Lepton3RunCommand` – similarly, class inheriting from `Lepton3Command` supporting the `RUN` method.
- `Lepton3EnableDisableCommand` – helper class, which inherits from `Lepton3GetSetCommand` and automatically supplies the `enabled/disabled` command parameters as `1/0` values.
- `Lepton3Control` – class containing the implementation of the CCI protocol having instantiated commands with their methods and parameters, routines for writing and reading any register via I^2C and also for executing a command.
- `Lepton3Command.Module` – enumeration class containing addresses of all Lepton 3 and 3.5 hardware modules, which can be addressed by a command.
- `Lepton3Command.Method` – enumeration class with `GET, SET and RUN` command methods.
- `Lepton3Control.Register` – enumeration class containing addresses of the most commonly used registers in Lepton thermal cameras.

In the repository, there is the `lepton3control.py` script that can be executed using a Python3 interpreter. This allows to quickly get, set or run any supported command with predefined options from the console. Listing 13 contains a complete list of all currently supported commands with their respective methods.

An interaction sample with getting and setting the false color lookup table (using the command `VID_PCOLOR_LUT` mentioned earlier) is shown in listing 14. Getting a currently set value for the color palette is in the new implementation translated back to the name of the option. The previous implementation would return a list of bytes and its meaning would have to be looked up in the CCI documentation.

```
Use lepton3control.py <i2c_number> <command> <method> [<data>]
        <i2c_number> -- for /dev/i2c-1 use 1
        <command> -- one of the commands below
        <method> -- one of the get, set, run - see available methods next to
        ↪  commands below
        <data> -- parameter only for the set method, leave empty to see
        ↪  available options
Commands:
        agc_calc_enable -- ['GET', 'SET']
        agc_enable -- ['GET', 'SET']
        agc_policy -- ['GET', 'SET']
        oem_bad_pixel_replacement_enable -- ['GET', 'SET']
        oem_calc_status -- ['GET']
        oem_customer_part_number -- ['GET']
        oem_flir_part_number -- ['GET']
        oem_output_format -- ['GET', 'SET']
        oem_reboot -- ['RUN']
        oem_sw_revision -- ['GET']
        oem_temporal_filter_enable -- ['GET', 'SET']
        oem_thermal_shutdown_enable -- ['GET', 'SET']
        oem_video_out_enable -- ['GET', 'SET']
        rad_enable -- ['GET', 'SET']
        rad_ffc_run -- ['RUN']
        rad_tlinear_auto_scale -- ['GET', 'SET']
        rad_tlinear_enable -- ['GET', 'SET']
        rad_tlinear_scale -- ['GET', 'SET']
        rad_tshutter_mode -- ['GET', 'SET']
        sys_aux_temp_k -- ['GET']
        sys_camera_up_time -- ['GET']
        sys_customer_serial_number -- ['GET']
        sys_ffc_run -- ['RUN']
        sys_ffc_status -- ['GET']
        sys_flir_serial_number -- ['GET']
        sys_fpa_temp_k -- ['GET']
        sys_frames_to_average -- ['GET', 'SET']
        sys_frames_to_average_run -- ['RUN']
        sys_gain_mode -- ['GET', 'SET']
        sys_ping -- ['RUN']
        sys_scene_statistics -- ['GET']
        sys_shutter_position -- ['GET', 'SET']
        sys_telemetry_enable -- ['GET', 'SET']
        sys_telemetry_location -- ['GET', 'SET']
        vid_focus_calc_enable -- ['GET', 'SET']
        vid_freeze_enable -- ['GET', 'SET']
        vid_low_gain_pcolor_lut -- ['GET', 'SET']
        vid_output_format -- ['GET', 'SET']
        vid_pcolor_lut -- ['GET', 'SET']
```

Listing 13: List of all currently supported commands with their methods in the `v4l2lepton3` library.

```
pi@th4:~/py/v4l2lepton3 $ python3 lepton3control.py 1 vid_pcolor_lut get
Opening i2c device: 1
Booted: True Ready: True ErrorCode: 0
Error code: 0
Result: wheel6

pi@th4:~/py/v4l2lepton3 $ python3 lepton3control.py 1 vid_pcolor_lut set
Opening i2c device: 1
Booted: True Ready: True ErrorCode: 0
Error: For SET method for command: vid_pcolor_lut you must specify one of this
↪  parameters: {'wheel6': [0, 0], 'fusion': [0, 1], 'rainbow': [0, 2],
↪  'glowbow': [0, 3], 'sepia': [0, 4], 'color': [0, 5], 'icefire': [0, 6],
↪  'rain': [0, 7], 'user': [0, 8]}

pi@th4:~/py/v4l2lepton3 $ python3 lepton3control.py 1 vid_pcolor_lut set fusion
Opening i2c device: 1
Booted: True Ready: True ErrorCode: 0
Error code: 0
Done.

pi@th4:~/py/v4l2lepton3 $ python3 lepton3control.py 1 vid_pcolor_lut get
Opening i2c device: 1
Booted: True Ready: True ErrorCode: 0
Error code: 0
Result: fusion
```

Listing 14: Demonstration of getting and setting the false color lookup table using the new implementation of the `VID_PCOLOR_LUT` command from the `lepton3control.py` file.

4.2 Capturing thermal frames: server-client model

As mentioned in the subsection 3.1.4, there were several issues with the previous implementation of the capture software. Even though the *video over SPI* (VoSPI) protocol has not changed from version 3 to 3.5, the old implementation was showing major desynchronization issues with the new camera. Furthermore, previously, the capture software in C++ was implemented with the idea that the frames would go into a local v4l2 virtual video device where they could be manipulated by common linux tools like ffmpeg, vlc or gstreamer. This would include sending the thermal video over a network using for example an RTP stream.

First of all, this method was quite slow. It required the capture application to pull frames from the camera and copy it over to the v4l2 virtual video device. Moreover, there had to be an ffmpeg (or similar) application running, which would copy the frames out of the virtual video device, compress them and send them over the network. The whole process copies each frame two times more than necessary. Additionally, in order to have a smooth fluent capture, the master computer (Raspberry Pi) has to be dedicated to operating the I^2C only. Any delay, caused by a different process using process time, might result in a camera desynchronization.

Secondly, the idea of sending the thermal video over the network using ffmpeg-like tool works only for false color RGB888 format. I was unable to find out a combination of format and lossless codec to transmit the raw Y16 thermal feed, whose each pixel is a

16-bit value representing real temperatures. Either the combination of codec and format was not supported by the tool or the codec fell back to a lossy one and the video came out with artifacts on the receive side. The RGB888 format comes already nonlinearly normalized from the camera for displaying purposes, and thus, is unusable for reconstructing temperatures of the scene.

The new project's requirement was to collect thermal feed from multiple cameras, detect and locate people and merge them into a single scene. Real temperatures are needed for the thermal detector, therefore, the whole capture part of the library had to be redesigned to be able to reliably transmit raw 16-bit thermal images over the network. On top of that, the whole capture process had to be sped up about 8 times to avoid losing synchronization with the camera.

The new implementation of the capture software follows the **server-client** model. It is written in C++ and forms a single process that runs on the Raspberry Pi computer to achieve maximum speed. The thermal unit behaves like a synchronous server. Its server process listens on a port and waits for a client to connect. Once a client is connected, it initializes the SPI interface and starts pulling frames from the camera and sending them over the open socket. When a connection is lost to the client, the thermal unit stops communicating with the Lepton camera and starts listening for another client.

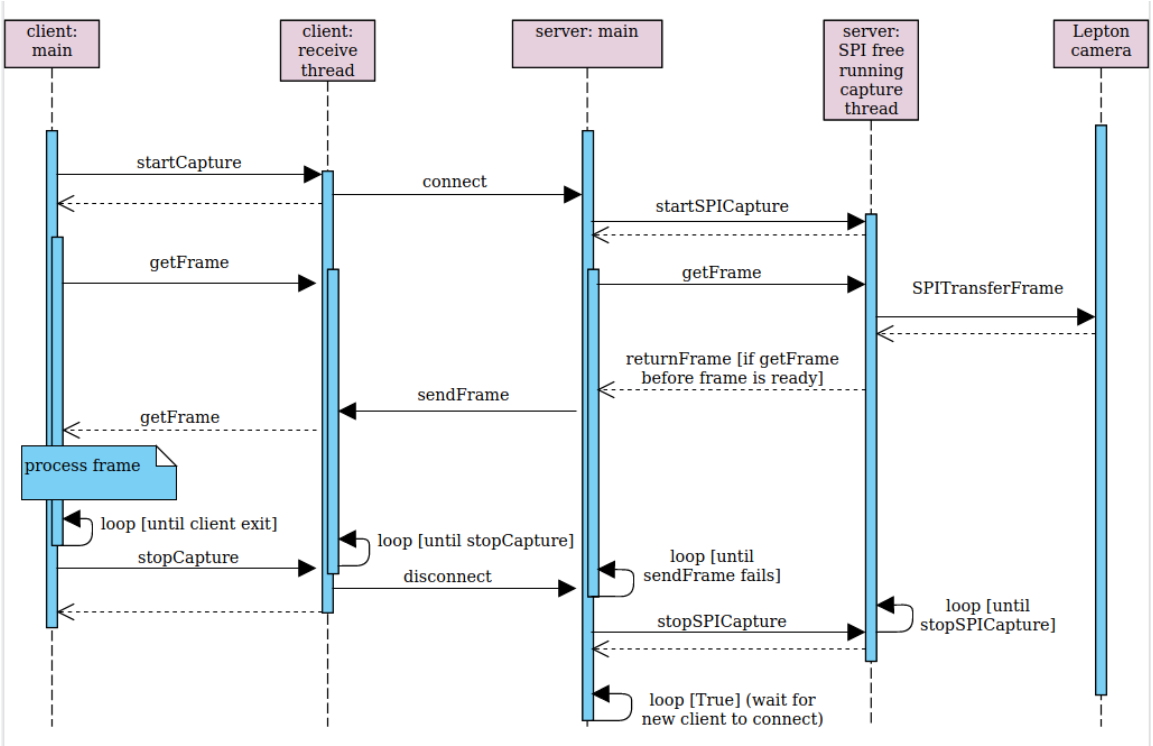


Figure 4.2: Sequential graph of the server-client model.

The central station for aggregating all thermal data acts in this case as a client. It connects to the port of the server and immediately starts receiving frames in the raw Y16 format. The communication between the client and the server is shown in figure 4.2. The `server:capture` and `client:receive` threads are free running, which means that they are continuously receiving frames from the Lepton camera or server respectively, and they hand over a frame to the main thread of the server or client process only when requested

using the `getFrame` call. This way it is ensured that the Lepton camera does not lose synchronization and that the client always gets the most recent frame. In case there is a new frame available before it has been requested by the main thread, the frame gets simply discarded. Discarding frames lowers the effective frame rate, which is being logged by the process and optionally by the client as well.

The connection between the server and the client is realized using a TCP connection. The TCP transport protocol has been chosen because it ensures in-order delivery of every packet. If packets got lost or arrived out of order, it would not be possible to assure proper reconstruction of each frame. The server is sending frame by frame, pixel by pixel. The client keeps receiving bytes until $160 \times 120 \times 2$ bytes are obtained. From this data, the client reconstructs the thermal frame in its raw format (Y16). Since the stream may be compressed by the `zlib` stream compressor, it is important to receive every frame and then decompress it.

The following subsections describe implementation, features and usage of both sides of the client-server implementation of the capture library.

4.3 Server side

Both server and client sides are available in the `v4l2lepton3` git repository[22]. In order to get the server up and running on the thermal unit manually without the use of Ansible, the following steps must be performed:

- Clone the `v4l2lepton3` repository into the Raspberry Pi.
- Install the server from sources using the following commands:

```
cd v4l2lepton3 && mkdir build && cd build
cmake ..
make server
```

  In order to compile the server, its dependencies must be installed first. The library depends on `libboost C++`, `CMake >= 3.6` and `zlib`.

- Run the server with custom arguments using the following command:

```
./server [-p <port>] [-s <spi_device>] [-t <timeout_ms>] [-h] [-c]
```

  - `-h --help`: shows help
  - `-p --port <port>`: sets server TCP port, default `2222`
  - `-s --spi <spi_device>`: SPI device, default `/dev/spidev0.0`
  - `-t --timeout <timeout_ms>`: frame acquisition timeout in ms, default `5000`
  - `-c --compress`: turns on zlib compression

The server by default starts listening on port `2222`. The SPI device may also be changed from the default `/dev/spidev0.0`. There is a timeout in the server set to 5,000 ms, which prevents the server from freezing up when it is impossible to obtain a frame from the camera.

[22] `v4l2lepton3` control and capture library git repository https://gitlab.com/CharvN/v4l2lepton3

If for some reason the frame is not read from the camera in time (set by the timeout), the server assumes that there has been a critical problem with connection to the camera and shuts down with an error code `2`. The connection problem may arise by having a loose wire or by the camera freezing up. This error code indicates that there has been a problem with the connection to the camera and a supervisor (a system service for example) may decide to reboot the camera using the CCI `OEM_REBOOT` command or by disconnecting it completely through the custom power switch.

When the `-c` argument is used, the server turns on `zlib` stream compression, which reduces the amount of transmitted data from 360 KB/s to 130-160 KB/s. Turning the compression on has a negative side effect because it puts strain on the CPU. With the compression on, the server drops the frame rate a little bit from 8.7 FPS to 7.8 FPS and sometimes causes a desynchronization event. The desynchronization is however temporary and the server quickly recovers, reinitializes the camera and starts transmitting frames once again.

The server is logging events into the `server.log.N` file. There can be up to 5 log files, and when the log file is full, the server rotates the log files so that `server.log.0` always contains the most recent log messages. An example of a single session captured in the log can be seen in listing 15. The server logs: starting of the server, waiting for a client, receiving a connection, an average frame rate of the transmitted thermal video, the event of a client disconnecting or an exit signal.

```
[2020-06-03 01:39:48.321926 [info] Waiting for a client on port: 2222
[2020-06-03 01:40:02.662328] [info] Client 192.168.1.2:47070 connected.
[2020-06-03 01:40:02.662926] [trace] Starting SPI capture thread.
[2020-06-03 01:42:23.939730] [trace] FPS: 8.77543
[2020-06-03 01:42:25.535755] [trace] Lost connection with the client: write:
↪  Broken pipe
[2020-06-03 01:42:25.536103] [info] Client disconnected.
[2020-06-03 01:42:25.536243] [trace] Stopping SPI capture thread.
[2020-06-03 01:42:25.536383] [trace] Closing SPI.
[2020-06-03 01:42:25.536793] [info] Waiting for a client on port: 2222
[2020-06-03 01:42:27.983937] [info] Received SIGINT. Exiting.
```

Listing 15: Session example from a `v4l2lepton3` server log.

The communication between the server and the client in this case goes only one way. Consequently, the client may disconnect at any point without notifying the server (as might happen unintentionally when there is a network connection problem). In order to handle such situation, the server is expecting possible errors when sending each frame to the client. When any error occurs, the server logs it and assesses the situation as the client has disconnected, stops its dedicated thread for SPI communication, resets its internal state and starts waiting for a new client.

In order to read a single frame from the camera, the capture process must read out 4 segments of 60 packets of 64 bytes (for raw Y16 format without telemetry). During this process, several problems may arise as follows:

- **SPI error**: When there is an SPI problem with either opening the device or transceiving the data, the server tries to reopen the SPI device and reinitialize the camera every 2 seconds until the per-frame-timeout limit is reached. After that, the server exits

with error code `2`, which may trigger some outer events, for example a forced restart of the thermal unit.

- **Invalid packet**: Lower 4 bits of the first byte having the value of `0xF` signifies an invalid packet. The camera sends invalid packets when it is in not in the ready state and can not send data. This may happen after startup, during resynchronization event (for example when the host fails to read the whole frame in time before the next one is ready) or when the camera is calibrating itself– during the *flat field correction* (FFC) event. When the capture process detects an invalid packet, it sleeps for 1 μs and tries reading a packet again. A valid packet usually appears after few attempts.

- **Lost synchronization**: An invalid packet may also be received in the middle of a segment transaction. This means that the host has not been able to read all segments of a frame in time (most likely because the host has been too slow) and the camera is in an unknown state. In this situation, the server forces resynchronization, which translates into idling the SPI communication for 200 milliseconds. This assures that the camera is reset and ready to start the transmission over. After a few invalid packets, the server starts receiving a new frame from the beginning.

- **Unexpected segment**: A frame consists of 4 segments. Each segment has a segment number stored in the header of the 20^{th} packet. Segments must be received in order from segment number 1 to 4. If a segment number is 0, the whole segment is invalid and should be discarded. The server keeps receiving segments until it gets segments with numbers from 1 to 4 in order. If the server receives an unexpected segment, it logs the accident and tries pulling another one. After all 4 segments are received in order, they get assembled into a frame and sent over to the client.

The server can handle well each issue that might occur during the thermal video feed transmission. If desynchronization happens, the server quickly recovers and starts sending frames soon after so that it is not even observable on the client side. The only time, when a client side experiences an interruption, is when the camera performs the automatic flat field correction. During that time, the camera is not sending any frames, it closes its shutter and briefly exposes the camera's sensor to a uniform thermal scene allowing itself to recalibrate, and thus, produce highly uniform images. The FFC process takes about 1 second.

Optimizations

The server C++ implementation had to be optimized and sped up in order to avoid constant desynchronizations with the camera. The following changes have been made when compared to the original code of the capture software:

- **Dual segment buffering**: There is a dedicated thread that is in charge of capturing a whole segment (60 packets of 164 bytes). After the segment is received, segment buffers are swapped and a new capture is started straight away. The caller thread meanwhile processes the received segment, strips the packet header and places the segment into the frame.

- **Dual frame buffering**: When the frame is complete, frame buffers are swapped and a new frame is being constructed asynchronously in the new buffer while the old buffer is used for sending the frame to the client.

- **Reduced OS calls**: Every SPI operation translates in an `ioctl` system call. The old implementation used to transceive one packet at a time (164 bytes) and then checked its header to see if the packet was valid. This naturally adds plenty of overhead and delays. In the new implementation, the server pulls a single packet at a time until the packet is valid (starting packet with number `0`). Then the rest of the segment (59 packets) is pulled at once using a single large transaction. The SPI transaction is $59 \times 164 = 9676$ bytes long.

  On a standard Raspberry Pi, an SPI transaction this large can not be performed straight away because the system SPI device buffer is set to only 4 kB. The transaction fails or hangs indefinitely. The size of the SPI buffer can be obtained by running the following command:

```
$> cat /sys/module/spidev/parameters/bufsiz
4096
```

  The size of the system SPI buffer can be increased up to 64 kB by placing

```
spidev.bufsiz=65536
```

  parameter to the end of the `/boot/cmdline.txt` file and rebooting. After increasing the buffer size, the large SPI transaction succeeds and gains valuable time.

These optimizations helped to speed up the server capture process to run smoothly without any desynchronizations.

4.4 Client side

On the client side, the following two implementations were created:

- **Python client**: The Python implementation is more generic and agile. The client is placed in the `v4l2lepton3.client` Python3 module, which can be easily used in other projects. In our case, we are going to be using this client implementation in the detection and localization process.
- **C++ client**: The C++ implementation is designed for a more specific use case. The C++ client works similarly as in the bachelor's project. It connects to the server and pushes every frame it receives into a local virtual video device, so that the thermal stream may be locally manipulated by generic video processing tools like the `ffmpeg, vlc, gstreamer` and so on.

The following subsection describes specific features, design and usage of these two client implementations.

4.4.1 Client implementation: Python

The python implementation can be found in the `Lepton3Client` class in the `v4l2lepton3.client` module. Its constructor only takes two arguments: `ip` and `port` and has one essential function `get_frame()`, which returns the most recent frame from the camera as a numpy `np.ndarray` object with 160×120 16-bit unsigned integers. The typical usage of this class is demonstrated in listing 16.

The Python package containing the client, control and single frame manipulation functionality can by installed into the system by running:

```
python3 -m pip install .
```

in the `v4l2lepton3` directory. The library is then used by the rest of the system to interact with the camera directly.

In the repository, there is the `lepton3client.py` file that imports from the module and can be directly executed. When interpreted with the Python3 interpreter, it is expecting two positional parameters `ip` and `port`. The script instantiates the Lepton client and starts receiving frames in an infinite loop. Each frame is normalized using OpenCV to increase its dynamic range and rendered in an OpenCV window. Every 2 seconds, the script prints average frame rate of processed thermal images, and using the *spacebar* key, the current frame can be saved.

The Python client automatically recognizes whether the server is using compression or not. At first, it tries to use the `zlib` decompression and if it fails, the client continues to receive frames with decompression disabled.

```
1 from v4l2lepton3.client import Lepton3Client
2 with Lepton3Client('192.168.1.51', 2222) as client:
3     while True:
4         frame = client.get_frame()
5         # process, normalize, print etc
```

Listing 16: Example of a typical usage of the Python client implementation.

4.4.2 Client implementation: C++

The C++ client is designed in more of a standalone manner. It is used by simply running it in the background and the result is having a thermal video stream available in a virtual video device in the local system. The client connects to the remote server and pushes every frame it receives into the video device.

Before compiling the client, there must be `c++ libboost`, `zlib` and `cmake` installed on the client system as well as a virtual video kernel module `v4l2loopback`. The `v4l2loopback` repository[23] contains all necessary installation steps.

After loading the kernel module with

```
1 $> sudo modprobe v4l2loopback
```

there is going to be a video device visible in the linux device tree, for example `/dev/video0`.

After cloning the `v4l2lepton3` capture and thermal library, the C++ capture client can be compiled using the following commands:

```
1 cd v4l2lepton3 && mkdir build && cd build
2 cmake ..
3 make client
```

[23] `v4l2loopback` virtual video kernel module – git repository https://github.com/umlaeute/v4l2loopback

The typical usage of the C++ client is demonstrated below:

```
./client -i <ipv4> [-p <port>] [-v <video_dev>] [-c]
```

- `-h --help`: shows help
- `-i --ip <ipv4>`: IP address of the server
- `-p --port <port>`: TCP port of the server, default `2222`
- `-v --video <video_dev>`: name of the video loopback device, default `/dev/video0`
- `-c --compress`: turns on `zlib` frame decompression

The C++ client initializes the virtual video loopback device to accept `RAW` color space and `Y16` (a single 16-bit intensity value per pixel) format. It connects to the server and every frame it receives pushes into the virtual video device. For the C++ client, it is necessary to configure the compression the same way as for the server.

Chapter 5

Person detection

This chapter describes the development and evolution of the person detector from the version in the bachelor's project to the new detector that is used now. The first section summarizes the method used in the old legacy detector and pinpoints its defects and drawbacks. The next section goes theoretically through the current general methods that can be used for person detection purposes. The sections that follow from then on describe the newly chosen method for person detection and the process connected with getting the method to work for our use case, its usage and results.

5.1 Legacy thermal detector

The person detector used in the bachelor's project was a simple one based on filtering temperatures outside the human-body-temperature range and image processing. The camera used in the old project was the Lepton 3, which does not have the true radiometry feature, so it was necessary to provide the mapping function between incident flux and real temperature. This function has been only an approximation with relatively large error.

5.1.1 Detector method

The old detector was designed to work in the following steps:

1. Convert pixel values to real temperatures using our approximate mapping function.
2. Filter out pixels that have temperatures outside the human-body-temperature range.
3. Increase the dynamic range of the thermal image using linear normalization.
4. Apply adaptive binary mean thresholding to obtain a binary image with white areas containing potentially detected objects.
5. Reduce noise using morphological transformations (opening and closing).
6. Find contours of compact white areas in the binary image.
7. Create border boxes around found contours.
8. Filter out boxes with insufficient area.
9. Draw found boxed into the original image.

A more detailed description of the legacy thermal detector can be looked up in the bachelor's thesis [18] [chapter 5]. An example of the detector in work is depicted in figure 5.1.

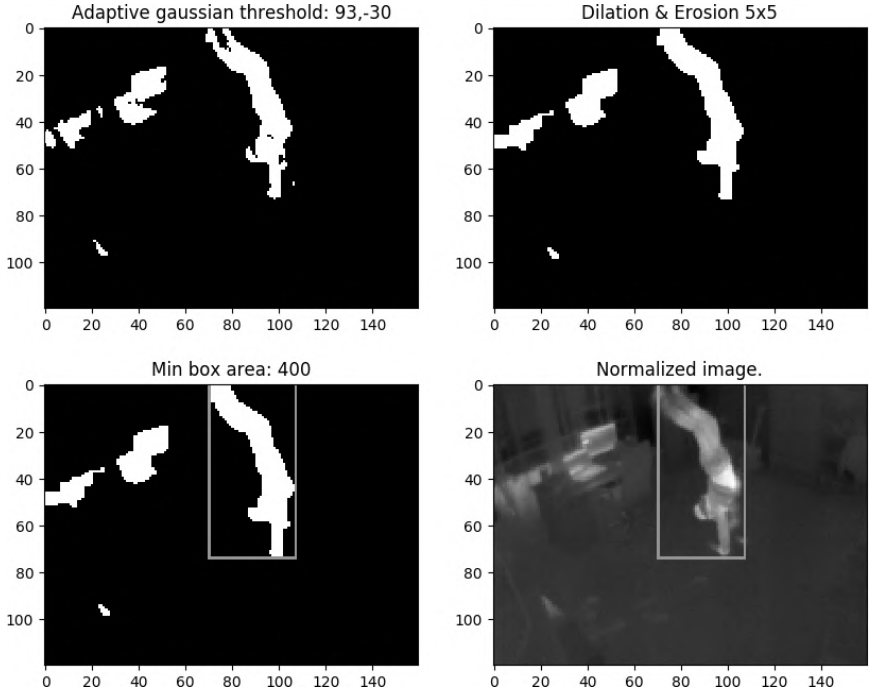


Figure 5.1: Example of the legacy thermal detector at work.

5.1.2 Detector hyper parameter calibrator

The legacy detector has various parameters and opportunities for adjustments. In the bachelor's project, it has been empirically calibrated for the particular dataset captured in quite a small room usually with a few people in it using the *false radiometric* Lepton 3 thermal camera. Since the detector was showing very good results in such environments and was extremely fast, it was planned to continue using this detection technique also in the new project, in which we are trying to create a large system that covers wide open areas with complex borders and multiple thermal cameras. In order to use the detector with the new Lepton 3.5 thermal camera in a larger area and achieve the best results possible, it was necessary to adjust the hyperparameters of the detector.

The parameters of the legacy thermal detector are the following:

- temperature filtering `MIN`, `MAX`
- adaptive thresholding `BLOCK SIZE`, `CONSTANT`
- kernel sizes for `OPENING` and `CLOSING`
- minimal detected object bounding box `AREA` and `TEMPERATURE`

In order to calibrate the hyperparameters of the detector, a calibration script for visual adjustments was created. The usage of the calibration script can be seen in listing 17. In order to run the calibrator, the `--directory` parameter has to be specified. The calibrator goes through all `.tiff` raw thermal images from the directory and displays results of the

thermal detector with the current hyperparameters. Detector parameters can be adjusted by pressing keys specified in listing 18. Every parameter change takes immediate visual effect.

```
python3 calibrate_thermo_detection_params.py -d DIR [-h] [-s SCENE] [-c CAMERA]

Arguments:
  -h,         --help           # Show this help message and exit.
  -d DIR,     --dir DIR        # Directory containing .tiff raw thermal images.
  -s SCENE,   --scene SCENE    # Scene model in .JSON format.
  -c CAMERA, --camera CAMERA   # Name of a calibrated camera in the loaded scene.
```

Listing 17: Arguments of the thermal detector hyperparameter calibrator.

Optionally, by running the script with `--scene` and `--camera` arguments, it is possible to load a scene and project detected objects into it using the specified camera in the later argument. The scene abstraction and its implementation is described later in this thesis, specifically in chapter 6. For now, the only important part is how well the detector detects and encloses detected objects in bounding boxes.

```
Controls +/-:
        - UP/DOWN - adaptive temp MAX
        - LEFT/RIGHT - adaptive temp MIN
        - Q/A - thresholding BLOCK SIZE
        - W/S - thresholding CONSTANT
        - E/D - kernel size OPEN
        - R/F - kernel size CLOSE
        - T/G - min box area
        - U/J - min BOX TEMP threshold
        - SPACE - next image
        - ESC - exit and print parameters
```

Listing 18: List of active control keys for the detector hyperparameter calibrator.

By pressing `SPACE` the script loads a next thermal image and displays detected objects and optionally their position in world space. By pressing `ESC`, the calibrator exits and prints out the current detector hyperparameters. An example of visual output of the calibrator can be seen in figure 5.2 and text console output in listing 19.

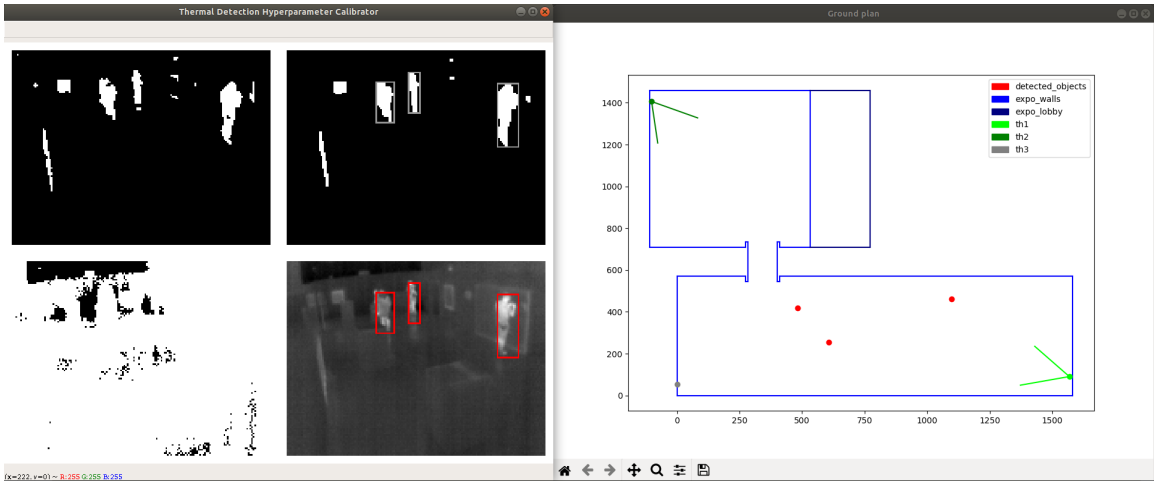


Figure 5.2: Visual example output of the detector parameter calibrator.

```
$> calibrate_thermo_detection_params.py -d ../DATA/th1 -s nm_scene.json -c th1
File: 20191213_1554_05.raw
Min: 19.27000000000004 °C  Max: 29.430000000000007 °C
Normalizing: min 29242  max: 30258
->[ESC]
Parameters: DETECTOR:
        - TEMP: min: 21.0 °C  max 37.0 °C
        - KERNEL SIZE: open: 2  close: 5
        - THRESHOLDING: block size: 95  constant: -30
        - BOX AREA: 54
        - MIN BOX TEMP: 25.0 °C
```

Listing 19: Text example output of the detector parameter calibrator.

5.1.3 Detector issues

As mentioned previously, the detector did a really good job on detecting simple and more importantly smaller scenes. Additionally, the detector is extremely fast, thus, it would be perfect for a real-time operation in a larger, complex environment with multiple cameras.

Unfortunately, after some testing in a large museum hall with higher amount of people with all sorts of occlusions, the detector started failing. The hall was 15.8 × 5.7 meters which is quite a large area for a camera with the resolution of only 160 × 120 pixels. When there were multiple people standing further away from the camera, their heat signature started creating a single interconnected object, which confused the detector into classifying a whole group of people as a single person. This problem is well visible in every part of figure 5.5.

Another problems are occlusions in general, if a person is standing in front of another, any software or even a person is going to have a hard time telling them apart. This is especially true for thermal imagery, where boundaries between similarly warm objects are not distinct enough. An example of such difficult thermal image can be seen in subfigures a), b) and c) of figure 5.5.

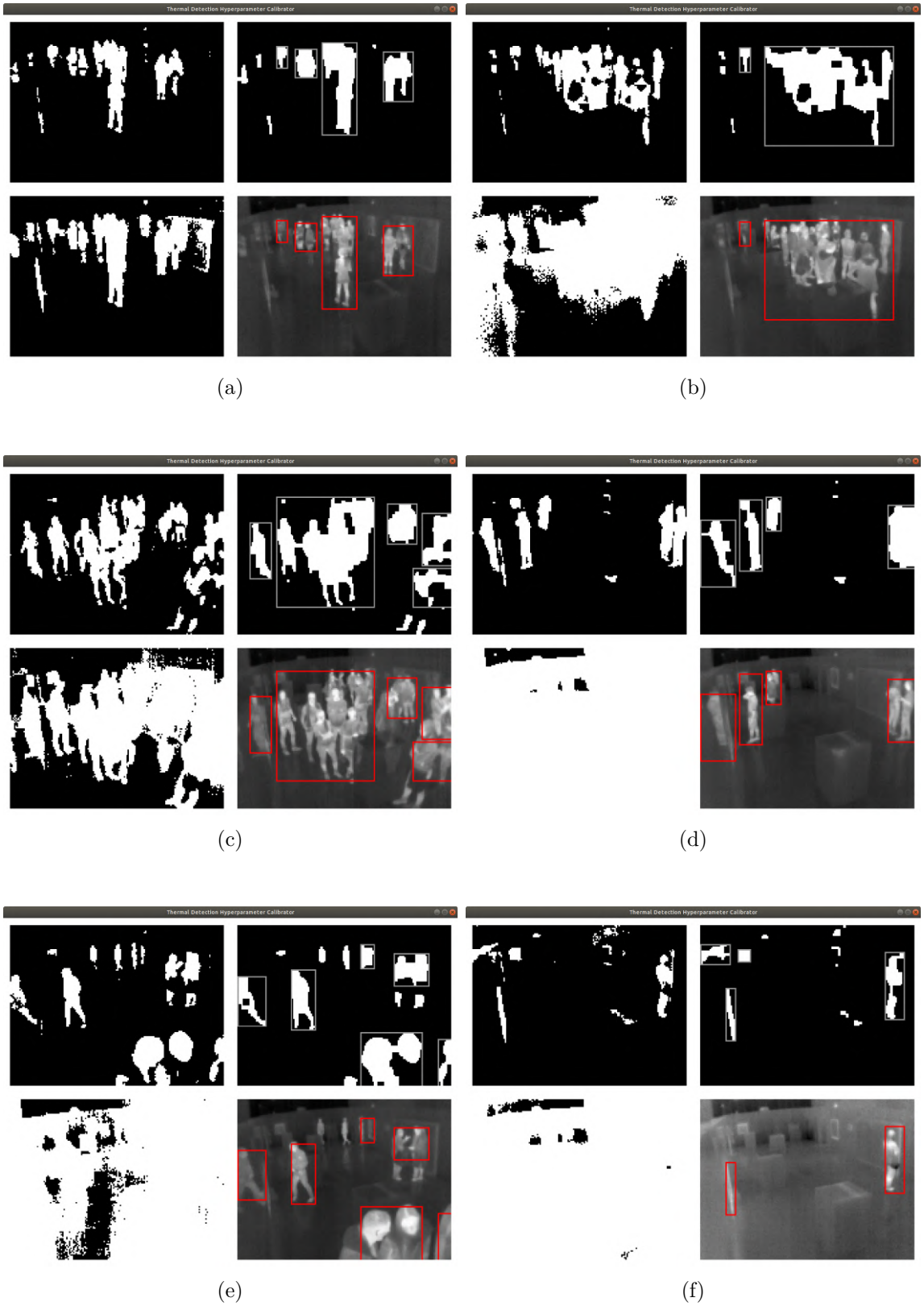


Figure 5.5: Examples of the failing legacy thermal detector in a more challenging environment.

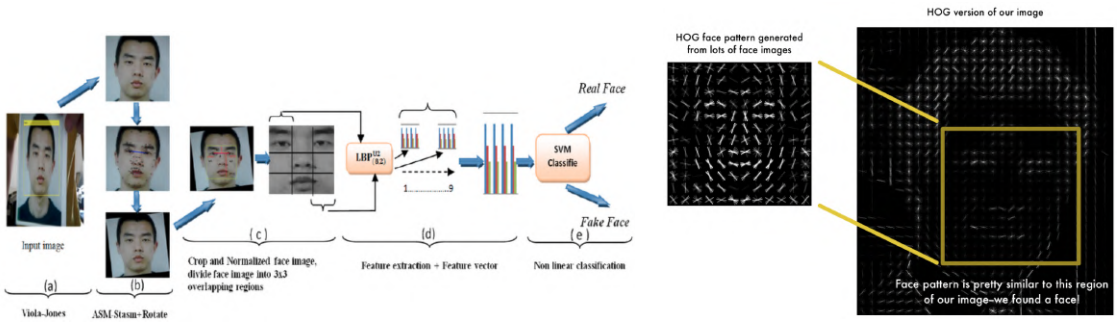
Moreover, the detector is prone to make false positive detections. If there is an object in the scene with about the same temperature as a human – like for example a TV, an information board with backlight or a radiator – the detector often classifies it as a person. It basically considers any large enough object in the allowed temperature range as a person. This issue can be noticed in subfigure f) of figure 5.5.

There is one more issue, which is more connected to the thermal imagery than the legacy detector itself, and that is thermal reflections. As a standard camera and object detector would classify a mirror image of a person as a real person, the same thing happens also in thermal imagery. Only it is a bit more unpredictable for humans because we can not see which surface happens to be a good thermal mirror. Usually flat and shiny surfaces, or for example a polished floor, reflect heat (as a long wave infrared radiation) very well. This kind of problem is noticable on the left side of subfigures c), d) and e) of figure 5.5.

These issues make the simple legacy thermal detector unusable for the new project, as one of the requirements is being able to detect individual persons in a larger area with more than a few people, for example in a museum exhibition hall. Therefore, a new detector based on a completely different method has been chosen, designed and implemented to suit the needs of the new project. The following sections focus on image detection methods in general, and later, on the method chosen for this project.

5.2 Object detection

Object detection is in simple words a task of placing boxes around objects in an image and saying what those objects are. Humans are extremely good in object detection, where on the other hand, computers struggled quite a lot historically. Even though object detection as a problem has been around since the sixties, the first actually good facial detector was released less than 20 years ago. The **Viola-Jones** algorithm [60], released in 2001, was using hand-coded features that were fed into an support vector machine (SVM) classifier. The hand-coded features for facial detection would be positions of eyes, nose, mouth and their relation with respect to each other. The algorithm performed well in detecting faces matching with the hand-coded features, however, struggled with detecting rotated faces or faces in any other orientation. [46]



(a) Viola-Jones algorithm

(b) Histograms of Oriented Gradients (HOG)

Figure 5.6: Illustration of the first facial detectors using hand-coded features and histograms of oriented gradients. (Source [46].)

In 2005, a new HOG-based detection algorithm was released. The histogram of oriented gradients (HOG) [19] was used as a feature descriptor, where each pixel of the image was replaced by a gradient showing the direction of decreasing pixel intensity with respect to surrounding pixels. The image was then divided into squares and all gradients inside each square merged into a single most dominant gradient. During this process, an image was replaced by a simple representation of the essence of the image using gradients. The algorithm would then use a similarity metric to determine how close an image is to the object we are looking for by comparing their gradient patterns. These two techniques are illustrated in figure 5.6. [46]

The biggest breakthrough came in 2012, when the *deep learning era* began. The **CNN** based **AlexNet** [33] from Alex Krizhevsky outperformed every other solution at that time and won multiple image classification contests. Convolutional neural networks had been known since the nineties, however it has been only recently with the immense increase of processing power and amount of data when the neural networks began to show their potential. The convolutional neural network in essence learns the feature descriptors on its own during the training process as opposed to the previous two methods in which they had to be crafted by hand.

These three algorithms, as described, correspond more to object classifiers, meaning, that they can tell what the object in an image is if there is nothing but one object in it. It can not detect and classify multiple objects in a single image. This has however been proven to be possible by repurposing any image classifier. The classifier can sequentially classify every part of an image through a sliding window, and detections with the highest confidence score, in that case, represent the output of the detector. This approach is however extremely computationally expensive.

In 2014, the popular **R-CNN** object detector [23] was released followed by **Fast R-CNN** [22] and **Faster R-CNN** [51] (2015), which were using the *selective search* technique instead of a sliding window to create sets of bounding boxes that were later fed into the classifier to cut down the number of operations. Even though the R-CNN was the most accurate detector on the market at the time, it had to look from thousand up to hundred thousand times at a single image to perform the detection, and that was still very far from real-time. The R-CNN model is illustrated in figure 5.7 and speed comparison with other models can be seen in table 5.1.

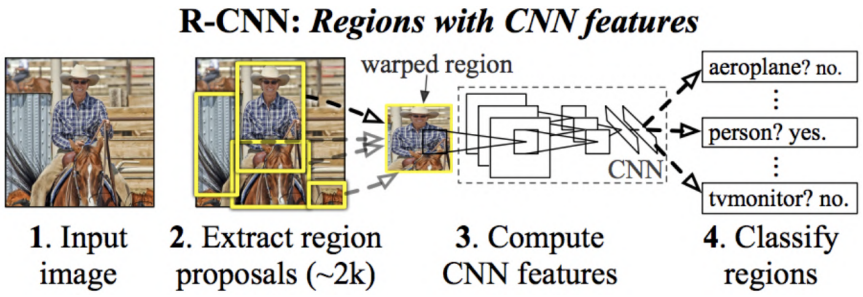


Figure 5.7: Illustration of the R-CNN detector pipeline. (Source [23].)

In 2015, there was a revolution in the field of object detection with the new **YOLO** algorithm which took a completely different approach and outperformed R-CNN and all of its variants.

Detector	Pascal dataset mAP[24] %	Speed
DPMv5[25]	33.7	14 s/image
R-CNN	66.0	20 s/image
Fast R-CNN	70.0	2 s/image
Faster R-CNN	73.2	7 image/s
YOLOv1	69.0	45 image/s

Table 5.1: Speed and accuracy comparison of relevant object detectors in 2016. (Source [47].)

5.3 YOLO – You Only Look Once

You Only Look Once (YOLO) is the state-of-the-art real-time object detection system originally presented by Joseph Redmon, Santosh Divvala, Ross Girshick and Ali Farhadi in 2015. When compared to all other detectors at the time, it took a completely different approach. Instead of repurposing an image classifier and using it to classify different regions in the image, it uses a neural network that takes an image as an input and in a single pass outputs regions with detected classes and confidence scores for each region as well as class.

The new approach required redefining parametrization of object detection. Every image is split into a grid, where each grid cell is responsible for predicting several bounding boxes and confidence scores for each bounding box, saying, how sure the detector is that a certain bounding box actually contains an object, and if there is an object, what kind of object it is.

If the image is split into 7×7 grid, each cell is responsible for predicting 2 bounding boxes and the detector should detect 20 classes, then the output of the neural network has

$$7 \times 7 \times (2 \times 5 + 20) = 1{,}470 \text{ values},$$

where 5 is the total number of parameters for each bounding box – namely `x`, `y`, `width`, `height` and `confidence`. This particular example shows how the YOLO version 1 is constructed.

To summarize, the YOLOv1 detector is a neural network trained to predict tensors with 1,470 values, which contain detected bounding boxes with corresponding class probabilities and detection confidence scores all at once. With this approach, the detector pipeline can be as quick as a classification pipeline. Moreover, since the whole image is processed at once, the model can incorporate global context of the image, which increases its accuracy. The detection process of the first version of YOLO is illustrated in figure 5.8.

[24] **mAP** – mean average precision, popular metric for measuring the accuracy of object detectors in %

[25] DPMv5 (Deformable Parts Model version 5) – one of the base legacy detection models using the sliding window approach

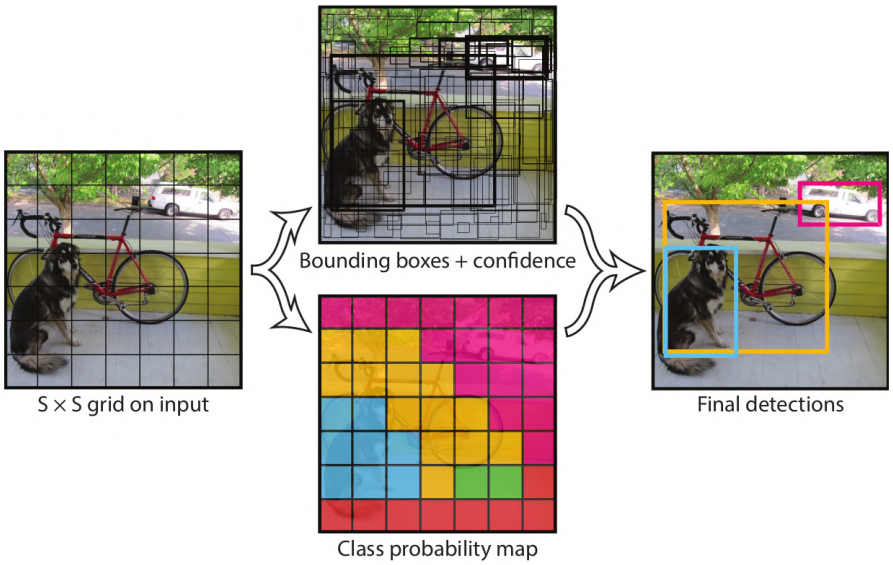


Figure 5.8: Illustration of the YOLOv1 detection process. (Source [48].)

In 2016, the second version of YOLO was released [49] featuring important improvements that increase its overall accuracy. The detector finished training on images with higher resolution and changed the way of representing bounding boxes. The YOLOv2 uses *dimension clusters* to represent bounding boxes. Using unsupervised learning, the creators extracted 5 most common shapes and sizes of bounding boxes occuring in the VOC 2007 image dataset[26], and used them as templates for bounding boxes that each cell in the YOLOv2 detector can detect. The new representation makes the problem easier to learn. The YOLOv2 also uses multi-scale training, meaning, that the input image size is not fixed throughout the training process but changes on the fly resulting in a more robust detector, as it works better on differently sized images.

The YOLOv3 [50] brought even more improvements in 2018. One of them is the support for multiple labels, as not every time are two different classes exclusive – like a pedestrian and a child. More importantly, the YOLOv3 is using a new backbone (or feature extractor part of the network) Darknet-53. The network has 53 convolutional layers with short-cut connections, allowing for extraction of finer-grained information from the image, which significantly improves detection accuracy of small objects. Unlike the previous versions, YOLOv3 makes bounding box predictions at 3 different scales further improving accuracy of the detector.

In February 2020, the original author of YOLO Joseph Redmon announced that he would stop his research in the field of computer vision. On his Twitter account he wrote:

> *"I stopped doing CV research because I saw the impact my work was having. I loved the work but the military applications and privacy concerns eventually became impossible to ignore."* — Joseph Redmon, 20. February 2020[27]

This post raised a lot of questions about what we are actually doing here and what consequences it might have. It also proves just how good the YOLO model is. Fortunately,

[26] PASCAL Visual Object Classes 2007 image dataset http://host.robots.ox.ac.uk/pascal/VOC/voc2007/

[27] Twitter post from J. Redmon https://twitter.com/pjreddie/status/1230524770350817280?lang=en

our system uses thermal cameras just so that facial or person recognition is not possible, therefore, privacy is not in danger in this case.

By the end of April 2020, the fourth version of YOLO [15] was released by Alexey Bochkovskiy and his team promising even better accuracy and speed, effectively dominating every other solution in the field of real-time object detection. The creators performed an ablation study[28] to test and select the most effective training optimization methods, which can lead to huge improvements in accuracy, yet, with no or minimal additional computational cost.

The tested methods were mostly data augmentation techniques that could potentially increase the descriptive power of the feature extracting part of the network. Some of the data augmentation methods are the following: *edge map, flip, rotate, detexture, cutmix, mosaic, dropblock regularization* and so on. A new activation function *mish* has been tested as well as other specialized techniques like *cross stage partial connections* or *multi-input weighted residual connections*. The optimizations also covered selecting the optimal hyper-parameters of the model like *number of training steps, batch size, learning rate, momentum, weight decay* or *minibatch size*. As a result, the YOLOv4 is superior to all other object detectors in terms of both speed and accuracy, which can be observed in figure 5.9.

For the new person detector used in this project, the YOLO object detection model has been chosen. More information about the specific YOLO implementation used in our project is located in section 5.5 of this chapter.

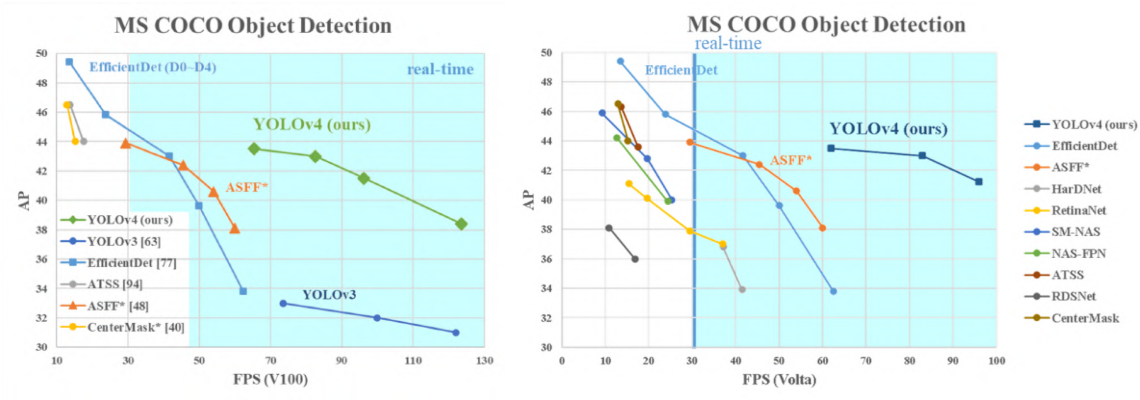


Figure 5.9: Speed and accuracy comparison of different object detectors. (Source [15].)

5.4 Training on a thermal dataset

Every object detector requires a set of labeled images on which the detector can be tested, and most of the time, even trained if the detector belongs to the category of supervised machine learning. Our chosen detector requires the labeled dataset for both training and testing. A labeled dataset contains images with information about objects we want to detect. This information is called *labels* or *annotations*, and there are different ways to store this information. A label stores information about: in which image the object is, in which region of the image the object is and what the object is (which class).

[28] *Ablation study* has its roots in the field of experimental neuropsychology of the 1960s and 1970s, where parts of animals' brains were removed to study the effect that this had on their behaviour. (Source: https://stats.stackexchange.com/questions/380040/what-is-an-ablation-study-and-is-there-a-systematic-way-to-perform-it)

5.4.1 Object annotation types

There are several different labeling formats. Every labeling format supports different types of objects, can be used with different image processing technique and is stored in a different way. Some formats support more types of objects, some are simpler and designed for a single purpose only. Some annotation formats may also be used in different disciplines than object detection. This subsection has been inspired by [24], [43].

(a) bounding boxes

(b) lines

(c) key-points or landmarks

(d) pixel-wise segments

Figure 5.10: Object annotation type examples. (Source [43].)

Some of the most commonly used object types are the following:

- **Bounding boxes**: simplest and most commonly used object type. For each object in an image, an enclosing border box is recorded. This object type is usually used for standard object detection.
- **Polynomial segments**: objects do not always have to be enclosed in a rectangular box. This object type allows the object to be circumscribed by a more complex shape made of polygons.
- **Pixel-wise segments**: an annotated object is described by a set of pixels that belong to it. This type of label is usually used for semantic segmentation of an image, where the segmentation algorithm is trying to classify each pixel into a particular class of objects. When compared with the previous techniques, the pixel-wise segmentation has the highest resolution.
- **Key-points**: this kind of annotation stores several interconnected points creating a skeleton of the object. This kind of label is often used for facial expression detection, human body parts, poses and similar.

- **Lines and curves:** often used for training autonomous vehicle systems for detection of road lanes.

Examples of these object annotation types are demonstrated in figure 5.10.

5.4.2 Annotation formats

Object annotations may be stored using several different formats. The most common labeling formats are `COCO`, `PascalVOC` and `YOLO`. Each format supports different object types and has its specific way to store them.

- **COCO Dataset Format**: The COCO format has been developed for the large, publicly available annotated image set COCO[29]. It supports several different object types and is usable for the broadest variety of digital image processing tasks including object detection, image segmentation, keypoint detection, image captioning and others. Annotations are all in a single `JSON` file with the following structure:

```
{
    "info": {
        "description": "COCO 2017 Dataset", "year": 2017
    },
    "licenses": [...],
    "images": [{
        "file_name": "img1.jpg",
        "height": 427, "width": 640, "id": 397333
    }],
    "annotations": [{
        "category_id": 1,
        "area": 702.1057499999998,
        "id": 154235421, "image_id": 397333,
        "bbox": [473.07,395.93,38.65,28.67]
    }],
    "categories": [
        {"supercategory": "person","id": 1,"name": "person"}
    ]
}
```

- **PascalVOC**: Another commonly used annotation format is PascalVOC. This format is very popular for object detection. Unlike COCO, the PascalVOC format uses `XML` encoding, and annotations are not stored in a single file, as instead, there is a dedicated annotation file with labeled objects for each image. Another difference is the way how border boxes are represented. A bounding box in the COCO dataset format is represented by `x`, `y` of the top-left corner plus `width` and `height`, while the PascalVOC format uses `x`, `y` of the top-left and bottom-right corners. The PascalVOC annotation file has the following structure:

```
<annotation>
        <filename>img1.jpg</filename> <segmented>0</segmented>
        <size>
```

[29] COCO image dataset http://cocodataset.org/#home

```
                <width>640</width>
                <height>427</height>
                <depth>3</depth>
        </size>
        <object>
                <name>397333</name> <pose>Frontal</pose>
                <occluded>0</occluded>
                <difficult>0</difficult> <truncated>0</truncated>
                <bndbox>
                        <xmin>473</xmin>
                        <xmax>511</xmax>
                        <ymin>395</ymin>
                        <ymax>423</ymax>
                </bndbox>
        </object>
</annotation>
```

- **YOLO**: The YOLO annotation format is by far the most simple and easy to use, but it can be used to annotate only rectangular objects. A regular `.txt` plain annotation file must be created for every image in the dataset with the same file name as the image it stores annotations for. Each line of the annotation file represents one object and has the following pattern:

```
<class_id> <x> <y> <width> <height>
```

  Coordinates `x` and `y` are referring to the top-left corner of the bounding box, so from this point of view, the notation is similar to the COCO dataset format. YOLO, however, has all 4 coordinates in relative units with respect to the image size. This means that `x`, `y`, `width` and `height` are all floating point numbers between 0 and 1. `class_id` is an index of a user-defined class starting at 0. Class translations are then placed in a separate file and utilised by the specific image processing algorithm. The YOLO implementation described later in this chapter requires a `.names` text file, where each line contains a class name. For example, the class name on the first line corresponds with `class_id` 0, the second line with `class_id` 1 and so on.

5.4.3 Custom thermal dataset

There are currently no pretrained YOLO models for thermal data. The only way how to utilize the YOLO object detector to detect people in thermal images is to create a custom, annotated, thermal dataset and use it for retraining the YOLO detection model, which has been originally trained on the large COCO image dataset. In other words, an extensive thermal image set had to be captured and manually annotated.

The YOLOv4 documentation [14] suggests to have at least 2,000 images in the custom dataset for each class we want to detect. In order to construct the custom thermal dataset, two different thermal datasets were combined. The first one is the only publicly available thermal dataset from FILR and the second one was created using previously constructed thermal units with Lepton 3.5 thermal cameras.

FLIR's thermal dataset

FLIR, as the world's largest company specializing in the design and production of thermal imaging cameras, offers a free thermal dataset[30] for algorithm training usage. The dataset contains thermal images and video from the FLIR's high-end TAU 2[31] thermal camera mounted to the front of a car. The thermal images in the FLIR's dataset come from a significantly better device costing over 200 times more than the Lepton 3.5. This means that the data is of higher resolution and significantly different characteristics, and using only the FLIR's dataset would presumably yield poor results.

Since the trained model would be used exclusively with the low-cost Lepton 3.5 camera module, it makes sense to combine the FLIR's thermal dataset with one captured using the Lepton 3.5 camera. This way it is possible to harvest features specific to the Lepton 3.5 and also essential characteristics of people on generic high resolution thermal images.

The FLIR's thermal dataset includes over 14,000 thermal images with 5 annotated classes in raw Y16 and normalized grayscale RGB888 formats. Since we are specifically interested in people, only thermal images containing people were selected. That sums up to **7,044** thermal images with **28,151** annotated person objects. An example from the FLIR's thermal dataset is visible in figure 5.11.

The FLIR's dataset is using the COCO annotation format to store object annotations for images. This format is not supported by the YOLO implementation that we are using in this project, and therefore, the annotations had to be converted into the YOLO format using a custom script.

Figure 5.11: Examples from the FLIR's thermal datset. (Source [8].)

Lepton 3.5 custom dataset

The total of 3 thermal units were deployed in the Czech National Museum[32] in Prague. The cameras were capturing thermal images in two larger halls (15.8 × 5.7 meters) every minute during opening hours for over several months. As a result, an extensive database of raw thermal images has been created.

[30] FREE FLIR Thermal Dataset for Algorithm Training https://www.flir.com/oem/adas/adas-dataset-form/

[31] FLIR's TAU 2 thermal camera – product page https://www.flir.com/products/tau-2/

[32] Czech National Museum in Prague https://www.nm.cz/en

During this process, the thermal units were behaving as independent units executing a script that was programmed to capture and save a single thermal frame at a time. The script is located in the `scripts` directory of the `v4l2lepton3` library under the name of `capture_periodical.py`. When the script is interpreted by the Python3 interpreter, it opens the preconfigured SPI device and reads out one frame in the raw Y16 format. The thermal image is then saved under the current timestamp name in both – `.tiff` format (raw) and `.png` format after normalization.

The Python interpreter is very slow on its own, and when combined with many system calls to interact with the SPI device, it is unusable for continuous capturing of the thermal feed. Capturing a single frame is however possible. The script has the total of 45 seconds to pull and save each thermal image. If it fails to do so before the timeout runs out, the script logs an error and quits.

The script is designed for a single run at a time, and therefore, has its own log file where it logs successfully saved frames and any errors that might have occurred, including timeouts which often mean that a camera froze and has to be rebooted.

The script has been scheduled to run every minute during opening hours of the Museum, which is from 10 to 18 hours. The ideal way to schedule such task is to use the built-in linux tool `cron`[33]. `Cron` is a linux daemon to execute scheduled commands. Scheduling a command in linux can be done by running

```
crontab -e
```

and inserting a scheduling configuration as a new line. In order to run the sequential capture script every minute every day from 10 to 18 hours, the following configuration has been inserted:

```
* 10-18 * * * python3 /home/pi/v4l2lepton3/scripts/capture_periodical.py
```

Scheduled commands can be listed using the `crontab -l` command.

From the captured thermal dataset, the total of **6,372** images were selected for annotation. Rejected images were often either empty or overcrowded scenes, which were not suitable for manual annotation process. On these 6,372 selected images, I annotated **25,477** person objects by hand. The annotating process took about a man-week of work.

For labeling the images, the `labelImg` tool has been used. `LabelImg` is a free graphical image annotation tool supporting PascalVOC and YOLO annotation formats. Graphical interface of this tool is shown in figure 5.12.

[33] cron(8) - linux man page https://linux.die.net/man/8/cron

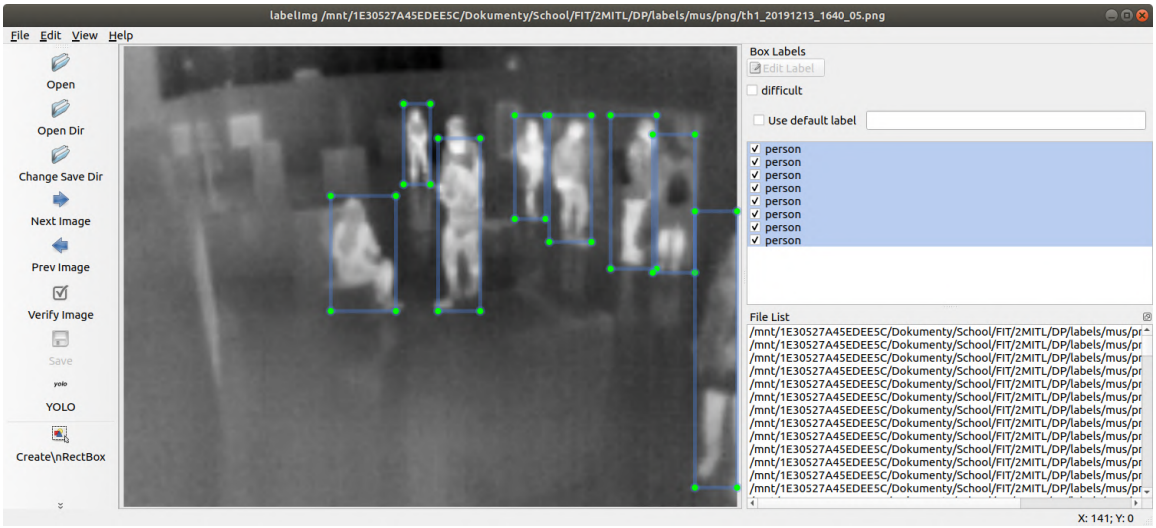


Figure 5.12: Example of the `LabelImg` graphical interface.

Merging the FLIR's and the custom thermal dataset resulted in having **13,416** thermal image files with **53,628** annotated person objects. This amount is significantly more than the minimum of 2,000 suggested by the YOLOv4 manual, which is a good predisposition for training a rigid and reliable object detector.

5.5 Darknet implementation of YOLO

For training the YOLO object detection models, the official YOLOv4 **Darknet** [14] implementation has been chosen. It is written in C and CUDA and has been forked from the original YOLO branch[34] created by Joseph Redmon and Ali Farhadi. The fourth version of the YOLO real-time object detection system is however implemented and maintained by Alexey Bochkovskiy in his forked repository [14]. By the time I was choosing an object detector, the YOLOv4 detector had been released for only 2 weeks and was already raising a wave of enthusiasm. Also at that time, the only YOLOv4 implementation was the Darknet one, which was perfectly fine, as it is well documented, remarkably fast, allows for GPU acceleration and contains a step-by-step official manual on how to train YOLO models on custom datasets.

5.5.1 Transfer learning

The Darknet framework comes with pretrained YOLO models, which have been trained on the COCO image dataset to detect up to 80 classes from regular color images. A YOLO model can not be used for thermal data straight away for obvious reasons. On the other hand, it is also not necessary to train the YOLO model from scratch, which would require a huge dataset, computational resources and plenty of time.

In this case, the pretrained model can be retrained and repurposed for our specific task – that is to detect person objects in thermal images. The retraining process is often called **transfer learning**. Repurposing pretrained models is a common process, as it saves a large portion of computation and time when compared with training from scratch.

[34] Darknet: Open Source Neural Networks in C `https://pjreddie.com/darknet/`

Most detector models based on convolutional neural networks can be simplified into two sections. The first section, which receives the input image, creates and extracts feature maps from the image, which are then used for the detection itself in the second section. These feature maps are pieces of information about the visual world – like individual interconnected shapes, color changes, contrast transitions, context of an object and so on. Analogically, a human seeing a round object of the skin color, with two evenly spaced eyes and wavy shaped red-colored lips underneath classifies the object as a human head. It is those little details and their context (distance, ratios, colors), which determine the classification result. These details correspond to the feature maps extracted in the first „section" of detection or classification models.

The idea behind transfer learning is that a pretrained network trained on a large dataset has built-in itself a generic model of the essence of the visual world. Retraining takes advantage of these learned feature maps and continues training from the point after the first section has converged. This significantly reduces training time, necessary amount of training images and improves overall accuracy. By transfer learning, we effectively train only the last part of the model. As the model already understands the essence of an image, the training focuses only on the custom task, which is in our case person detection on thermal images.

The Darknet repository contains a step-by-step manual[35] on how to perform transfer learning and retrain pretrained YOLO models for detection of custom objects. This manual has been used to prepare the custom training dataset for transfer learning.

5.5.2 Preparing custom dataset for training in Darknet

From the previous step, there are 13,416 thermal images and 13,416 annotation files in the same folder. For training the detector, the RGB888 normalized grayscale thermal image format has been selected.

Image format

The Darknet framework is able to work with single channel images, however, this feature must be enabled in the source code and the framework must be recompiled. We decided not to go down this path, as the community had been reporting issues, bugs and interestingly loss of accuracy when using a single channel input data. Even though the YOLO pretrained model has been trained on a color dataset, it has been proven that it can well handle gray scale images as well. In our case, the model is being fed by gray scale images with three channels, meaning, that each channel of every pixel has the same 8-bit value.

As a consequence to this decision, normalized 8-bit gray scale images have been selected from the FLIR's thermal dataset. From the custom Lepton 3.5 thermal dataset, the raw Y16 images had to be normalized and converted to 8-bit gray scale `.jpg`. Since it is desired to utilize the feature map from the pretrained YOLO model as is, it only makes sense to provide the model with images visually as close as possible to human readable images – images on which a human could perform the detection task quickly and accurately.

If we were to input raw Y16 thermal images, the model would probably first have to adjust the feature map and develop the image normalization function in order to start „seeing" the image as it was trained to with the regular color dataset. In order to prevent

[35] How to train (to detect your custom objects) using yolov4 manual https://github.com/AlexeyAB/darknet#how-to-train-to-detect-your-custom-objects

this, and therefore speed up the convergence and increase accuracy, the images fed into the model are preprocessed by a custom normalization function.

Thermal image normalization functions

Raw thermal images in the Y16 format have the theoretical range of pixel values from 0 to 655.35 K. For a common room temperature scene, this range is too wide and only a small portion of the available pixel values are used. When displaying such raw image, the scene appears uniform with minor color differences. For displaying or feeding the image into the detection model, it is better to increase its dynamic range by first clipping temperatures outside some reasonable range that are not of interest, and then applying a normalization algorithm. Some of the most common normalization algorithms are listed bellow:

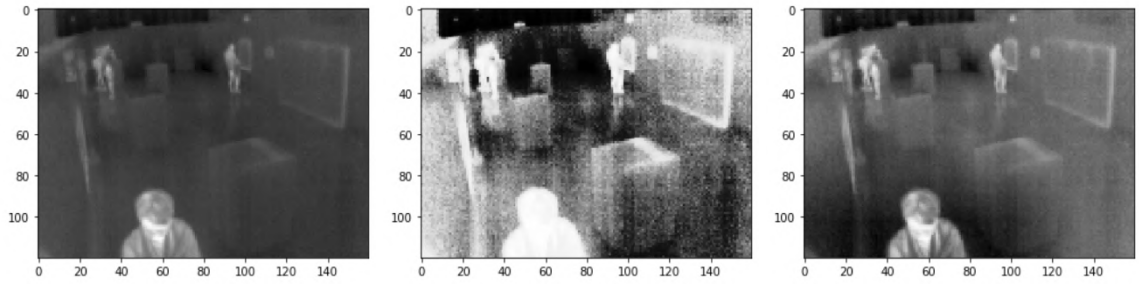
- **Linear normalization**: Linear normalization is the simplest algorithm for dynamic range expansion. It takes the minimum and maximum pixel value from the image and linearly spreads all values in between into the new range. If we aim for 8-bit images, the new pixel values are between 0 and 255.
- **Histogram equalization (HEQ)**: Histogram equalization is a bit more complex normalization technique. It solves one problem of the linear normalization. If there is a relatively cold scene with a small very hot object, all pixel values corresponding with temperatures between the cold scene and the hot object are left unused. This results in a small white object on a monolithic black background.

  Histogram normalization is one of the best methods for image enhancement without the loss of information. It spreads out more frequent pixel values, flattens the histogram. This results in areas of lower local contrast to gain higher contrast. It is a nonlinear normalization yielding unrealistic images, however, it is great for object detection as it highlights outlines of objects.
- **Contrast limited adaptive histogram equalization (CLAHE)**: CLAHE is an advanced method that is not using a single global histogram but normalizes several local histograms throughout the image. In an image with two large parts – one cold and other hot – the adaptive algorithm increases the contrast in both areas separately revealing the finest details.

  The regular adaptive histogram equalization algorithm tends to overamplify regions with near constant pixel values. The contrast limited variant reduces this noise amplification problem. The CLAHE algorithm limits the contrast amplification by clipping the histogram above a predefined threshold and distributing the values above equally into every bin of the histogram. This method yields the most desirable results and is also implemented in the Lepton camera itself under the *automatic gain control* (AGC) feature name. This feature can however be used only for RGB false color camera output.

All these three normalization algorithms are implemented using `numpy` and `OpenCV` in the `ThermoDetection/ThermoHelper.py` file in the second project's repository[36] containing the detection system. The comparison between normalization algorithms is shown in figure 5.13.

[36] `thermo-person-detection` – detection system git repository https://gitlab.com/CharvN/thermo-person-detection



(a) Linear normalization (b) Histogram equalization (c) CLAHE

Figure 5.13: Comparison of normalization algorithms for increasing dynamic range.

5.5.3 Configuring a custom YOLO model

This subsection summarizes steps required to prepare the custom thermal dataset for transfer learning according to the Darknet YOLOv4 manual[35].

1. Clone and install the Darknet framework from repository [14].
2. Create a copy of `cfg/yolov4-custom.cfg` model configuration file into the `cfg/yolov4-thermal.cfg`.
3. Alter the `cfg/yolov4-thermal.cfg` configuration file for the custom dataset as follows:
   (a) Set `batch=64`.
   (b) Set `subdivisions=32`. The manual suggests 16, however, with 6 GB of GPU RAM, the graphics card was running out of memory.
   (c) Set `max_batches=4000`. It is recommended to set the number of iterations to 2,000 × the number of classes, but not less than 6,000 or number of images for training. Since we have around 13,000 images in total, the parameter has been set to 12,000 at first, but after a few experimental runs, 4,000 iterations were well enough in this case. More on that later.
   (d) Set `steps=3200,3600` which correspond to 80 and 90 % of `max_batches` parameter.
   (e) Set the network size to `width=320 height=320`. The dimension must be a multiple of 32, images from the Lepton 3.5 have resolution of 160 × 120 but FLIR's thermal dataset contains images of resolution 640 × 512 pixels. A reasonable size of the network in between has been chosen, so that resources are not wasted on a large network, yet, detailed information from the FLIR's dataset are not lost.
   (f) Set `classes=1` for each of three `[yolo]` layers in the config.
   (g) Set `filters=18` for each of three `[convolutional]` layers before every `[yolo]` layer. The recommended value is (the number of classes + 5) × 3.
4. Create a `data/thermal.names` plain text file containing class names. Each line contains a single class name, the first line corresponds to a `class_id` 0. In our case, the file only contains the word `person` on the first line.

5. Create a `cfg/thermal.data` plain text file with the following content:

```
classes = 1
train   = data/thermal.train.txt
valid   = data/thermal.test.txt
names   = data/thermal.names
backup  = backup/
```

   `backup` is the directory where the training checkpoints of the model will be stored. `train` and `valid` parameters point to text files with image locations intended for training and testing respectively.

6. Place all images and annotation files from the merged thermal custom dataset into the `data/thermal` directory.

7. Create `data/thermal.train.txt` and `data/thermal.test.txt` files containing links to all image files in the `data/thermal` directory split by a reasonable ratio. We have chosen 80 % training and 20 % testing.

8. Download pretrained weights for the specific YOLO model according to the configuration file. For YOLOv4 template, the `yolov4.conv.137` weights file is suggested.

For purposes of this project, steps 1, 2 and 8 were repeated to prepare other YOLO models for comparison. In total, `yolov4`, `yolov3-spp` and `yolov3-tiny` models were prepared for transfer learning with the custom thermal dataset. After the configuration process, the Darknet directory contains new files as demonstrated in listing 20.

```
darknet
|-- backup
|-- cfg
|   |-- thermal.data
|   |-- yolov3-spp-thermal.cfg
|   |-- yolov3-tiny-thermal.cfg
|   |-- yolov4-thermal.cfg
|-- data
|   |-- thermal.names
|   |-- thermal_all_test.txt
|   |-- thermal_all_train.txt
|   |-- thermal
|       |-- img1.jpg
|       |-- img1.txt
|       |-- img2.jpg
|       |-- img2.txt
|       |-- (...)
|-- yolov4.conv.137
|-- darknet53.conv.74
|-- yolov3-tiny.conv.15
```

Listing 20: Tree of new files created during the configuration of the custom YOLO model in the `darknet` directory.

5.5.4 Training custom YOLO models on thermal images

The training of the YOLO detection models has be performed on a laptop with specifications listed in table 5.2.

CPU	Intel® Core™ i7-9750H CPU @ 2.60 GHz 12×
RAM	16 GB DDR4 @ 2,666 MHz
Storage	1 TB SSD NVMe
Graphics	1× NVIDIA GeForce GTX 1660 Ti
Graphical memory	6 GB GDDR6
NVIDIA Driver version	435.21
NVIDIA CUDA version	10.1
NVIDIA CUDA compute capability	7.5 (Turing)
NVIDIA cuDNN version	7.6.5

Table 5.2: Specifications of a laptop used for training custom YOLO models.

In order to properly utilize the power of this machine, it was important to edit the Darknet's `Makefile` before compilation. The modified parameters in the `Makefile` are shown in listing 21.

```
GPU=1
CUDNN=1
CUDNN_HALF=1
OPENCV=1
AVX=1
OPENMP=1

# GeForce RTX 2080 Ti, RTX 2080, RTX 2070, Quadro RTX 8000, Quadro RTX 6000,
↪  Quadro RTX 5000, Tesla T4, XNOR Tensor Cores
ARCH= -gencode arch=compute_75,code=[sm_75,compute_75]
```

Listing 21: GPU-specific changes in the Darknet's `Makefile`.

Setting the graphics specific *compute capability* using the `ARCH` parameter to 7.5 made the largest impact on performance. Setting the computation capability higher than 7 makes the `nvcc` CUDA compiler utilize mixed-precision on tensor cores of the GPU, which can improve performance more than 3 times. During some testing, the GPU version of Darknet was measured to be up to **6 times faster** than the CPU version.

After the configuration of the custom thermal dataset and YOLO models, the training is started by running the following command:

```
./darknet detector train cfg/thermal.data cfg/yolov4-thermal.cfg yolov4.conv.137
```

Current information about the training process is displayed in real time in a dedicated window. The course of training is represented by a graph of **loss** for current iteration. The information panel also contains: *current iteration, maximum iterations* and *estimated remaining time*. Additionally, the *mean average precision* **mAP** metric can be turn on and displayed in real time by using the `-map` argument. A loss function represents the

current error of the model on the training data and should decrease with iterations. The mean average precision is a widely used representative metric for measuring wellness and accuracy of the model on testing data – here, the higher the better. The training process saves weights every 1,000 iterations into the `backup` folder. After the training is completed, the most recently updated loss graph is stored as a detailed look on the training process.

After training is complete, one model should be selected from all the checkpoints along the way – the most accurate yet general. The Darknet's manual recommends to select model weights from a so called *early stopping point* as demonstrated in figure 5.14. This way we avoid selecting an overfitted model.

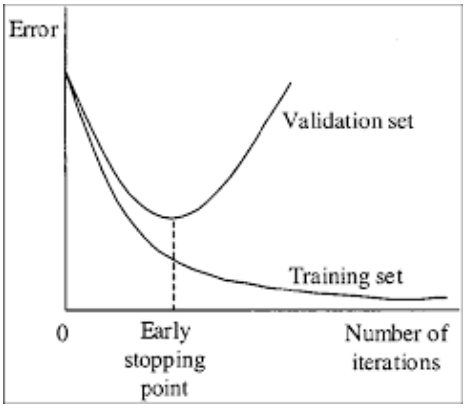


Figure 5.14: Loss development graphs on training and testing data throughout iterations demonstrating the problem of overfitting the model. (Source [14].)

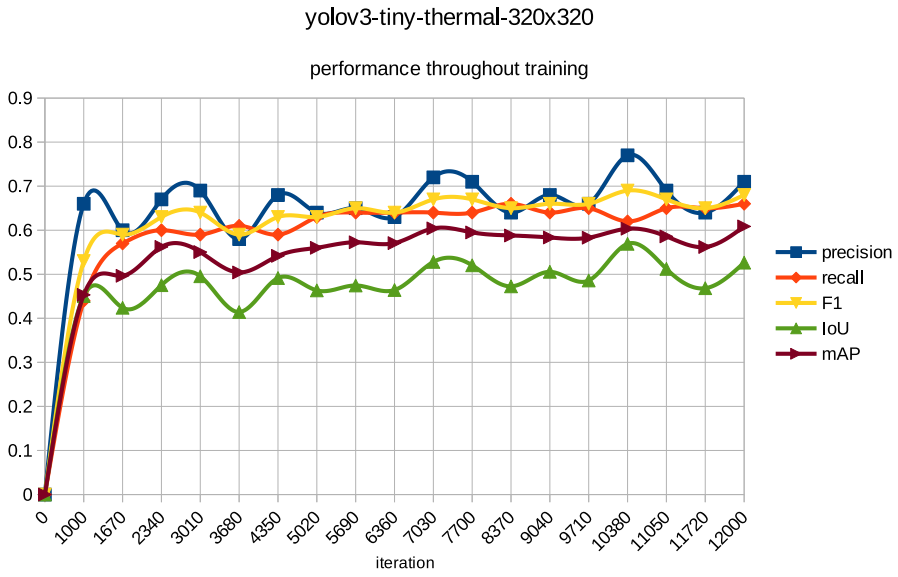


Figure 5.15: Graph of performance of the `yolov3-tiny-thermal` model on testing data throughout iterations.

The first training session was performed on the `yolov3-tiny-thermal` with the size of 320 × 320 and 12,000 recommended iterations. The *tiny* model was chosen because it was designed to run on embedded devices and is very small, thus, really fast to train. Its training chart is visible in figure 5.16 as well as its accuracy development throughout iterations in figure 5.15.

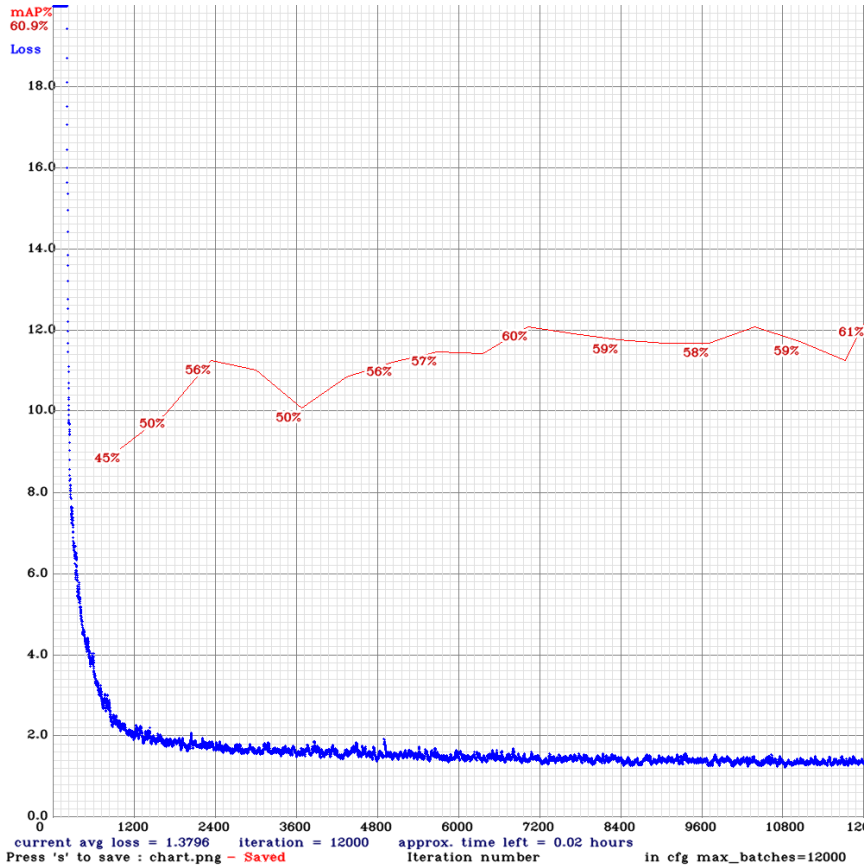
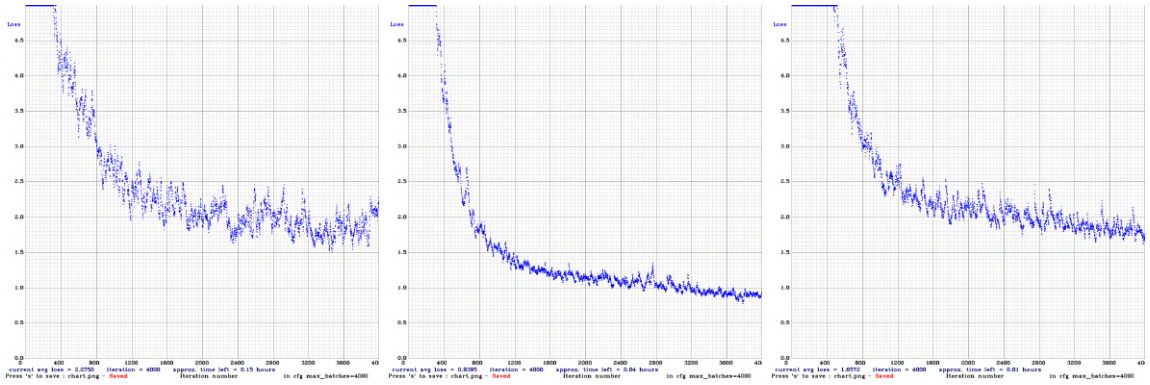


Figure 5.16: Loss and mAP development graphs throughout training of the `yolov3-tiny-thermal` model.

From the training graphs, it is noticeable that from iteration 2,000 onward the loss function stops decreasing and the mean average precision metric stops rising. In fact, it does not get significantly higher even after twice as much iterations. With this being said, the number of iterations can be safely reduced to around 4,000 iterations. In this training session, the model weights from iteration 2,000 (or 3,000) would be used as output if we wanted to follow the *early stopping point* strategy in order to achieve the highest level of generalization (avoid overfitting).

The number of iterations required for the full training of a YOLO model most likely correlates with the dataset being used, or the fact that all used YOLO models behave quite alike, which is observable in figure 5.17. The loss graphs of the other models – `yolov4-thermal` and `yolov3-spp-thermal` – show similar behavior, therefore, it is safe to assume that 4,000 iterations is enough for every one of them. This way we can set the number of iterations to 4,000 and experiment with the size of the models in order to find the best performing one. Firstly, all three different types of the YOLO model has been trained having the same size. From these three models, `yolov4` performs the best, so experiments

with larger model size have been done only for the YOLOv4 object detector. The maximum size of the YOLO model trainable using the reference computer is 512, which requires just under 6 GB of graphics memory.



(a) `yolov4-thermal` (b) `yolov3-spp-thermal` (c) `yolov3-tiny-thermal`

Figure 5.17: Loss development graphs of custom YOLO detection models during the first 4,000 iterations.

Training of the neural networks was surprisingly fast, most likely because the detector was trained only for one class and because of the GPU acceleration with the *compute capability* 7.5.

Complete characteristics of each custom YOLO model trained on the thermal dataset are summarized in tables 5.3 and 5.4. From these three detectors, the one based on `yolov4`, is the fastest and yields the best results. The `yolov4-thermal-320` object detector has therefore been chosen as the flagship for person detection in this project. The model with size 320 × 320, even though has lower mAP, has higher precision and seems more reliable and accurate on thermal images from the Lepton 3.5 camera, as the larger models would sometimes tend to incorrectly detect small objects, for example an arm of a person can sometimes get detected as a person far away from the camera.

Feature	**yolov3-tiny**	**yolov3-spp**	**yolov3-tiny**	**yolov3-spp**
Size	160 × 160	160 × 160	320 × 320	320 × 320
Stopping iteration	2,000	2,000	3,000	2,000
Detection FPS	8.70	6.70	8.70	6.62
Training time [hours]	0.25	2.1	2.5	6.3
Precision	0.40	0.69	0.67	0.80
Recall	0.38	0.61	0.60	0.76
F1-score	0.39	0.65	0.63	0.78
Average IoU	27.09 %	49.69 %	47.53 %	58.33 %
Mean average precision	24.38 %	57.74 %	56.21 %	77.51 %

Table 5.3: Results and characteristics of each custom YOLOv3 detection model trained for 4,000 iterations on the custom thermal dataset.

Feature	yolov4	yolov4	yolov4
Size	320 × 320	416 × 416	512 × 512
Stopping iteration	2,000	2,000	2,000
Detection FPS	8.70	8.70	8.70
Training time [hours]	5.2	5.9	7.8
Precision	0.87	0.85	0.84
Recall	0.80	0.87	0.90
F1-score	0.83	0.86	0.87
Average IoU	66.30 %	66.19 %	64.58 %
Mean average precision	85.66 %	89.98 %	91.50 %

Table 5.4: Results and characteristics of each custom YOLOv4 detection model trained for 4,000 iterations on the custom thermal dataset.

5.6 Usage of the trained YOLO detector

The result of the transfer learning is a model trained to detect persons on thermal images. From the three trained models, the best one has been chosen to be used in this project. The model is represented by two files: `yolov4-thermal.cfg` and `yolov4-thermal.weights`. The `yolov4-thermal.weights` file is obtained from the `backup` directory, where all intermediate checkpoints created during training are stored.

These two files can be easily used for detection using the `DNN` module of the OpenCV library. The DNN module implements forward pass (inferencing) with deep networks, pretrained using deep learning frameworks like Caffe, TensorFlow, Torch or Darknet. The DNN module has been supporting YOLO models trained by the Darknet framework for a long time, however, the YOLOv4 was added to the list of supported networks only 1 day before I considered using OpenCV for the detection itself. Consequently, the OpenCV library had to be compiled from sources in order to have the most recent updates including the YOLOv4 support.

The Python code in listing 22 demonstrates the simple usage of the retrained custom YOLOv4 thermal detector using the DNN module of the OpenCV library.

```
1 import cv2
2
3 yolo = cv2.dnn_DetectionModel(yolov4-thermal.cfg, yolov4-thermal.weights)
4 yolo.setInputSize(width = 160, height = 160)
5 yolo.setInputScale(1.0 / 255)
6
7 classes, confidences, boxes = yolo.detect(img, confThreshold=0.1,
  ↪  nmsThreshold=0.4)
```

Listing 22: Demonstration of the DNN module of OpenCV library performing detection on an image using the custom YOLOv4 thermal detector.

The DNN module and the way it operates has been abstracted into the `YoloDetector` class in the `ThermoDetection/YoloDetector.py` file of the `thermo-person-detection`

repository[37]. The class automatically loads the custom YOLOv4 model and can be just imported and used for the detection.

The `confThreshold` parameter sets a confidence threshold. Based on this threshold, the detector filters out potential objects with the confidence score less than `0.1`. The `nmsThreshold` (non-maximum suppression threshold) parameter is used to reduce the number of proposed objects that are overlapping up to a certain level. The detector often proposes multiple objects of the same class that are just slightly misaligned, which creates duplicate detections. All of these proposed detections are in reality a single object. The non-maximum suppression algorithm calculates the *intersection over union* (IoU) of every two proposed objects, then, if the calculated IoU is larger than a specific threshold, the proposal with smaller confidence score is removed. [16] An exaple of the non-maximum suppression algorithm in action can be seen in figure 5.18.

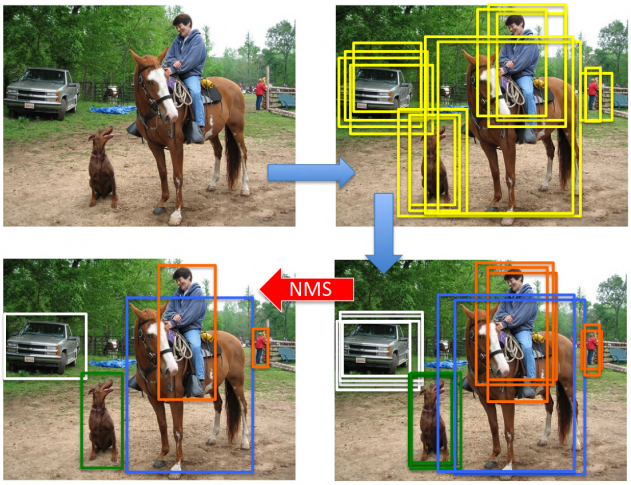


Figure 5.18: Example of the non-maximum suppression (NMS) algorithm in action. (Source [16].)

The Darknet library can be compiled into a shared library with a python binding, however, using it inside the Lepton 3.5 client is not as neat and clean as using the DNN module of the OpenCV library. The Darknet approach is definitely a way to go for high performance use cases. The Lepton 3.5 camera however has an effective frame rate of only 8.7 FPS, so detection speed is not crucial in this case. The CPU version of OpenCV used to slow down the live stream with the detection about 1 FPS, however, the most recent OpenCV update with YOLOv4 optimizations solved the throttling completely, and the live thermal feed with the YOLOv4 detector is at full speed.

Interestingly, the OpenCV library compiled to run on a GPU is in this case slower and gives out only about 6.3 frames per second. I attach the speed reduction to the fact that the OpenCV has to frequently copy a small amount of data back and forth between the graphics and main memory.

The `YoloDetector` class has been used in an upgraded `lepton3client` implemented in the `lepton3client_detect.py` file. The script connects to a remote Lepton 3.5 server and every frame it receives runs through the custom YOLO detector. Detected objects are highlighted in the image shown to the user inside an OpenCV window in real time.

[37] `thermo-person-detection` – detection system git repository https://gitlab.com/CharvN/thermo-person-detection

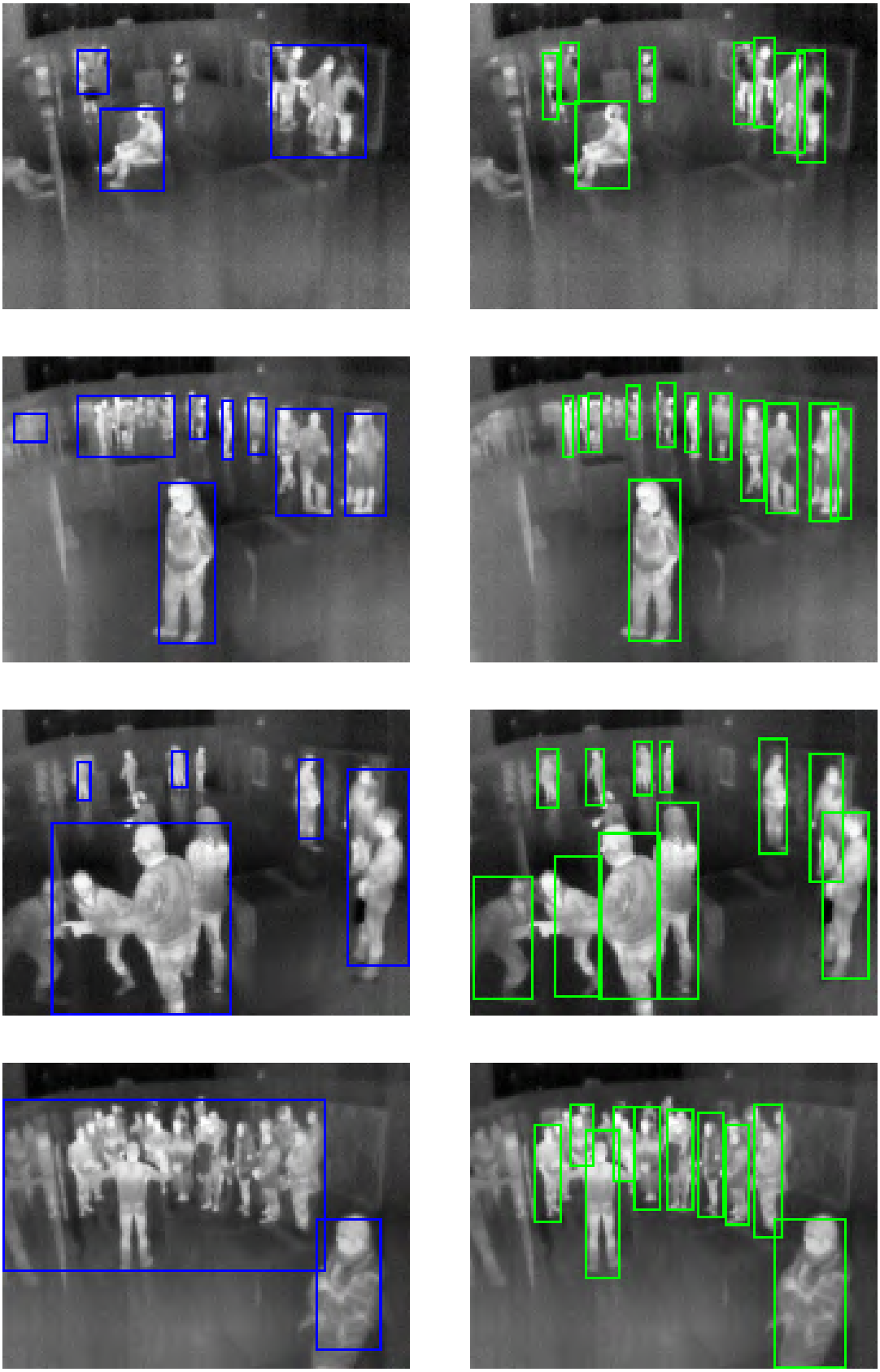


Figure 5.22: Comparison between the legacy and the new YOLOv4 detector.

The detector proved itself very well. When compared with object detection models, the YOLOv4 is the state-of-the-art real-time detector. Our custom YOLO thermal detector is fast, has a remarkable accuracy and solves almost all problems that the legacy thermal detector based on trivial image processing had. It deals well with larger groups of people, occlusions and eliminates false detections of warm objects like radiators, TVs and so on. The thermal reflections are however still problematic and there is no easy fix for that. The YOLOv4 detector does a surprisingly good job even for a very complex scenes that poses a real challenge even to humans. A side by side comparison with the legacy detector can be seen in figure 5.22.

Chapter 6

Scene reconstruction

The detection process yields bounding boxes around detected people represented by image coordinates. The next step is to create an abstraction of the environment monitored by the camera, and then, translate each detected object into an approximate location in the model of the environment. This chapter describes the implemented scene abstraction and explains the solution to the problem of how to programmatically compute correspondence between image coordinates of a bounding box and 3D coordinates in the scene model.

Theoretical parts of this chapter are almost identical to those in the bachelor's thesis [18] (section 5.3) and have been taken over. The implementation part has been however revised completely with the exception of the mathematics behind the theory. Changes to the implementation have been forced by the fact that the output of this project should be a system capable of utilizing multiple thermal cameras to compute somewhat accurate locations of people in a complex environment. The old implementation was capable to project only into a simple rectangular plane from a single camera.

6.1 Projecting objects from image to 3D scene model

In order to approximate coordinates of an image object in world space, it is necessary to understand the camera location and orientation in world space. Knowing the pose of the 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) – it is possible to cast rays from the camera origin into world space using pixel coordinates of the detected bounding boxes, and therefore, approximate their location in the monitored environment.

The knowledge used to write this section comes from [27], [36], [56], [57], [39], [20], [38], [41], [56], [59] and [17].

6.1.1 Perspective-n-point problem

The perspective-n-point problem (PnP) is a problem of estimating the pose of a calibrated camera. By pose, we understand the camera 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 those mapping points, estimating the pose is a

matter of solving a system of linear equations. At least 4 pairs of points are required to find a solution. The perspective-n-point problem can be expressed by equation 6.1, which comes from the perspective projection (**world to screen** or **world to image** transformation).

$$\mathbf{P_i} = \mathbf{K} \left[\begin{array}{c|c} \mathbf{R} & \mathbf{t} \end{array} \right] \mathbf{P_w} \tag{6.1}$$

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 6.1 can be found in equation 6.2.

$$\begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & \gamma & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \left[\begin{array}{ccc|c} r_{00} & r_{01} & r_{02} & t_x \\ r_{10} & r_{11} & r_{12} & t_y \\ r_{20} & r_{21} & r_{22} & t_z \end{array} \right] \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} \tag{6.2}$$

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 6.3). 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.

$$\left[\begin{array}{c|c} \mathbf{R} & \mathbf{t} \end{array} \right] = \begin{bmatrix} r_{00} & r_{01} & r_{02} & t_x \\ r_{10} & r_{11} & r_{12} & t_y \\ r_{20} & r_{21} & r_{22} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.3}$$

The matrix of intrinsic camera parameters $\mathbf{K}$ represents the transformation of a point from camera to screen (or alternatively 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 6.2, it is possible to compute rotation and translation vectors, and therefore construct the **world to camera** or **view** matrix (6.3) that can be used to transform points from world into camera space.

6.1.2 Solving PnP problem using OpenCV

By solving the perspective-n-point problem, it is possible to determine the rotation and position of the camera in world space based on pairs of 3D world and 2D corresponding image coordinates. The pose of the camera can be calculated using the function `cv2.solvePnP()` or `cv2.solvePnPRansac()` in the OpenCV library. [38] The function takes, along with the mentioned 2D and 3D mapping points, also the intrinsic camera matrix $\mathbf{K}$ and distortion coefficients.

$$\mathbf{K} = \begin{bmatrix} f_x & \gamma & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{6.4}$$

The intrinsic camera matrix (equation 6.4) can be constructed with focal length in each axis and usually camera resolution – width x and height y. In this case, the focal lengths

f_x, f_y are calculated from the horizontal and vertical field-of-view (FOV), which are known parameters of the Lepton 3.5 camera.

For the intrinsic camera parameter 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 6.5, the center point is calculated using the camera image resolution x and y.

$$c_x = \frac{x}{2} \tag{6.5}$$
$$c_y = \frac{y}{2}$$

$$f_x = \frac{c_x}{\tan \frac{h_{fov}}{2}} \tag{6.6}$$
$$f_y = \frac{c_y}{\tan \frac{v_{fov}}{2}}$$

The variable γ represents the *axis skew* causing shear distortion in the projected image. For 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 the distortion coefficients equal to 0.

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

From the rotation vector, it is possible to construct a rotation matrix using the *Rodrigues'* algorithm[38], and after that, express the whole transformation from world to camera space using a single matrix – the **world to camera** matrix. The Rodrigues' algorithm is also available in the OpenCV library as `cv2.Rodrigues()`.

6.1.3 Reversing world to screen transformation

The **world to camera** matrix, when combined with the intrinsic parameter matrix, represent the complete world to screen transformation. This transformation can be reversed, which allows for points to be projected from the image back to world space. To reverse the transformation, equation 6.1 (or 6.2) needs to be rearranged. The rearranged form can be seen in equation 6.7.

$$\begin{bmatrix} x_w \\ y_w \\ z_w \\ w_w \end{bmatrix} = \left[\begin{array}{c|c} \mathbf{R}^{-1} & -\mathbf{R}^{-1} \cdot \mathbf{t} \\ \hline 0 & 1 \end{array}\right] \mathbf{K}^{-1} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} \tag{6.7}$$

where $-\mathbf{R}^{-1} \cdot \mathbf{t}$ can be expressed as in equation 6.8.

$$-\mathbf{R}^{-1} \cdot \mathbf{t} = - \begin{bmatrix} r_{00} & r_{01} & r_{02} \\ r_{10} & r_{11} & r_{12} \\ r_{20} & r_{21} & r_{22} \end{bmatrix}^{-1} \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} \tag{6.8}$$

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

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 6.7 describes the process of a point transformation from image space back to world space. The intrinsic camera matrix inverse $\mathbf{K^{-1}}$ represents the transformation from screen (image) space into camera space. Again, a single 4×4 matrix can be assembled to simplify the transformation from the camera coordinate system to the world coordinate system. This matrix would be called **camera to world** matrix and is the key to reverse-projecting points from the image to the world coordinate system.

It is important to note that projecting points from world space (3D) to screen space (2D) always causes a loss of one dimension (depth). When reverse-projecting a point from image space of a single camera, it is impossible to determine the original world point, as infinite number of world points with varying depth gets projected into that single image point. In other words, for any image point, a line exists in the scene model.

For our use case, it would be helpful to obtain a single point in world space instead of a line. For any image point that we want to reverse project we can cast a ray in world space from the camera origin and look for interesting intersection points in the scene. The ray is effectively a line in the world coordinate system whose all points get projected into the same one image point. If we find an intersection of this ray with an interesting other object in world space – for example the ground plane with the third coordinate equal to 0 – such intersection point may be declared as the result of the reverse transformation. This means that for any image point we are able to assign a single point in world space. This is particularly useful for estimating location of objects from a single camera. This is of course only possible provided that we have an additional information about those objects – for example that those objects are people and they stand on the floor.

6.1.4 Reverse projection process of an image point

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

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 effectively 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 $\mathbf{C_w}$. (This step can be done only once for each camera.)

4. Calculate euclidean vector $\mathbf{R}$ (ray direction) pointing the direction from camera origin $\mathbf{C_w}$ to point $\mathbf{P_w}$ – both in world space. This vector effectively describes 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 as in equation 6.9

$$\mathbf{R} = \mathbf{P_w} - \mathbf{C_w} \tag{6.9}$$

5. By scaling vector $\mathbf{R}$ using scalar s and adding it to the camera origin in world space, it is possible to find single point $\mathbf{P}$ on the ray in world space (equation 6.10).

$$\mathbf{P} = \mathbf{C_w} + s \cdot \mathbf{R} \tag{6.10}$$

Scalar s can be adjusted so that the result point meets our specific requirements. For example, scalar s can be set in such way that the z coordinate of the point on the ray has a specific value z. For the value of $z = 0$, the computed point in world space both gets projected into the initial image point and coincides with the ground plane of the scene. The scalar is computed in equation 6.11.

$$s = \frac{z - z_{C_w}}{z_R} \tag{6.11}$$

6.1.5 Placing a detected person into the scene

The previous subsection described how to reverse-project an image point into a line (camera ray) in the scene coordinate system. The ray can be collapsed using an additional information about the object placement. In this case, the detected object is a person, therefore, we may select a ray that intersects 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 in order to collapse the ray into a single point in world space. In other words, we are explicitly searching for a point on the ray in world space with a particular z coordinate – height from the ground. For any detected object in the thermal image enclosed by a bounding box, we can either assume its lower edge is touching the ground or its upper edge is located the average-human-height above the ground.

6.2 Scene abstraction software

One of the outputs of the bachelor's project [18] was a scene abstraction script allowing to model a rectangular scene with a single camera and reverse-project detected people into it. In order to model a multi-room exposition or any larger complex environment with multiple cameras, the scene abstraction had to be rewritten almost from scratch and improved significantly.

In the new implementation, the scene model is stored in a `JSON` configuration file, which contains among others positions of cameras and a list of boundaries with their respective names and displayed colors. The boundaries are stored as a list of vertices that are connected one by one. A vertex is a tuple of `x` and `y` coordinates of the integer type. The scene abstraction is programmed in the `ThermoDetection/Scene.py`. An example of the scene configuration file is depicted in listing 23. Each camera has three additional attributes: `ip`, `port` and `mapping_points`. The first two are used for enabling a live connection during calibration or for production usage, and the `mapping_points` attribute contains mapping points from the image (2D) to the world (3D) coordinate system that are used for calibrating each camera in the scene every time it is loaded by solving the perspective-n-point problem.

These mapping points can be either added manually into the configuration file or inserted automatically using a visual scene calibrator implemented in the `ThermoDetection/SceneCalibrator.py` file.

```
{
  "scene": {
    "boundaries":[{
        "name": "room1",
        "color": "blue",
        "vertices": [[0, 0], [1582, 0], [1582, 570], [0, 570], [0, 0]]
    }],
    "cameras":[{
        "name": "unit1",
        "color": "green",
        "ip": "192.168.1.111",
        "port": 2222,
        "position": [1582, 17, 250],
        "mapping_points": {
          "image_points":  [[81, 28], [19, 36], [147, 57], [102, 15], [81, 6],
          ↪  [17, 12], [25, 117]],
          "world_points": [[0, 570, 0], [5, 5, 0], [1059, 570, 0], [529, 570,
          ↪  170], [0, 570, 250], [0, 0, 250], [1250, 0, 0]]
        }
    }]
  }
}
```

Listing 23: Example of a scene `JSON` configuration file.

By calibrating a scene, we understand creating mapping points between the image and the world for each camera in the scene model. The mapping points are then used to calculate the screen to world transformation matrix for every camera when the scene is loaded. The transformation matrix is then used to project detected objects into the scene model.

A scene can be visually calibrated by running the `calibrate_scene.py` script, which utilizes both static and dynamic scene calibrator. The usage of the calibration script is depicted in listing 24. The calibration script has to be always provided with a path to the scene configuration file. Optionally, if the `--camera` argument is supplied, the calibrator connects to the camera and starts the calibration process using a live feed from the camera. In that case, the connection parameters have to be stored in the configuration file for the particular camera. Alternatively, the `--file` argument can be used for calibrating the camera using a static raw `.tiff` image.

```
python3 calibrate_scene.py [-h] -s SCENE [-c CAMERA] [-f FILE]

Arguments:
  -h,         --help          # show this help message and exit
  -s SCENE,   --scene SCENE   # Path to a scene .JSON config file.
  -c CAMERA, --camera CAMERA # Camara name to be calibrated.
  -f FILE,    --file FILE     # Calibration .TIFF file for static calibration.
```

Listing 24: Arguments of the scene calibrator from `calibrate_scene.py`.

When launching the calibrator, the `.json` configuration file must be specified using the `--scene` parameter. After either connecting to the live camera feed or loading the static thermal image from a file, the calibration process begins. The calibrator shows a view from the camera (static or live) and a ground plan of the scene model with all cameras in their respective locations. The calibration process consists of clicking points in the window with the camera view and entering 3D corresponding world coordinates.

The typical step by step process of calibrating a new scene is the following:

1. Create a `scene.json` configuration file with boundaries and cameras with their associated colors, names and vertices in the world coordinate system. Optionally add ip and port for each camera for live calibration.
2. Run the scene calibrator without any arguments other than `--scene` to confirm the configuration corresponds to a valid scene. The calibrator will only display the ground plane of the scene.
3. Run the scene calibrator to calibrate one camera from the scene using the `--camera` parameter. Use static or dynamic calibration. Example in figure 6.1 shows the calibrator in action calibrating the `th1` camera from the static `th1.tiff` image executed by

```
./calibrate_scene.py -s scene.json -c th1 -f th1.tiff
```

4. Adjust brightness and contrast of the image using the arrow keys.
5. Click an image point on the camera view and enter `x,y,z` corresponding coordinates in world space.
6. Repeat step **5** at least 4 times. A previously added pair of mapping points can be removed by pressing `BACKSPACE`.
7. Close the calibrator using the `ESC` key and press any key to overwrite the `scene.json` configuration file. During this step, all mapping points created in step **5** will be stored in the configuration file, and next time the scene is loaded, the camera will be automatically calibrated. (The screen to world projection matrix will be computed.)
8. Repeat from step 3 for every camera in the scene.

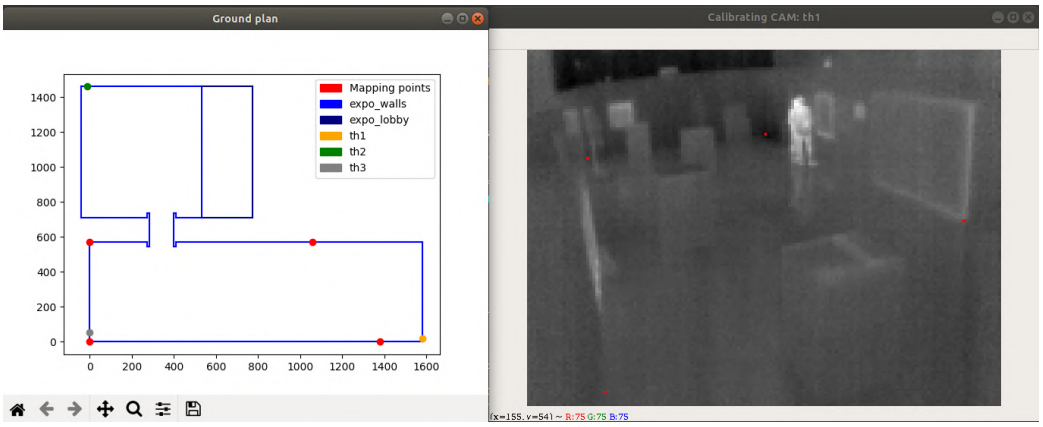


Figure 6.1: Example of the visual scene calibrator calibrating camera `th1`.

The result of the scene calibration is a configuration file of a scene model in a single coordinate system with cameras with known projection matrices. A calibrated camera shows its field of view using visible arms on the ground plan. An example of a scene with calibrated cameras can be seen in figure 6.2. With the screen to world projection matrices, it is possible to assign a line in the same 3D world coordinate system to each image pixel of every camera. These lines are then used for localizing detected people and placing them into the scene.

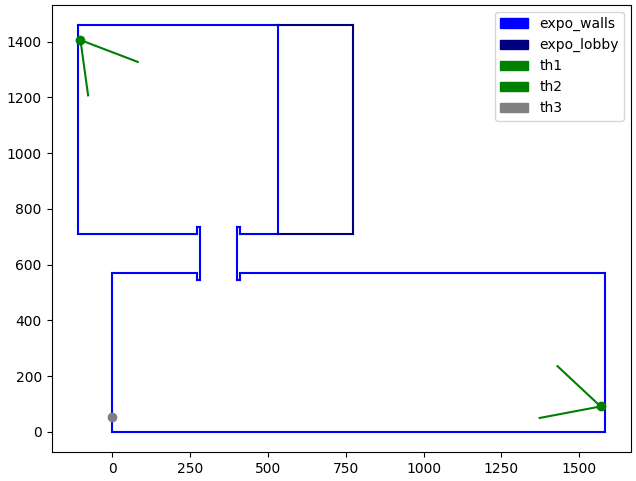


Figure 6.2: Example of a scene with calibrated cameras displaying their field of view.

Chapter 7

Real use-case deployment

This chapter summarizes and demonstrates real-world application possibilities of the new system for detection and localization of people. The first section goes step by step through the installation process of the detection system in a new environment and the second section describes an ongoing cooperation with the Czech National Museum in Prague, which had expressed interest in using the system for movement analysis of its visitors.

7.1 Deployment process

This section describes the steps of a typical installation of the detection system in a new environment.

1. Based on the size and complexity of the monitored area, choose the number of thermal units. For larger areas, it is important to be aware that the Lepton 3.5 thermal camera has resolution of 160 × 120, which means that localization accuracy will suffer. If a thermal unit is placed 2 meters above ground on one side of a 16-meter-long hall, a single image pixel difference can result in almost 1 meter difference on the opposite side of the hall. In this particular example, it would be worth considering placing a second camera to the opposite side of the hall to improve accuracy.

2. Obtain or assemble the thermal units.

3. For every thermal unit, download and burn a Raspbian image into an SD card. Having the SD card connected to a computer, configure a network connection for the thermal unit and install an `ssh` key for easy and secure access over the network.

4. Install thermal units at their stable positions in the monitored area and connect them to the network and power.

5. Try to `ssh` into each thermal unit to see if everything works properly. While connected to the thermal unit, it is advised to change the default Raspberry Pi password and the name of the device.

6. Clone `v4l2lepton3`[39] and `thermo-person-detection`[40] repositories.

[39] `v4l2lepton3` control and capture library git repository https://gitlab.com/CharvN/v4l2lepton3

[40] `thermo-person-detection` – detection system git repository https://gitlab.com/CharvN/thermo-person-detection

7. Change the Ansible `hosts` file to match the number of deployed thermal units with their respective IP addresses for `ssh` connection under the `[thermal_units]` section.

8. Run the Ansible playbook to install all required dependencies and configure the thermal units with the following command:

```
$> ansible-playbook -i hosts thermal_deploy.yml -v --forks 4
```

   The `forks` parameter sets the number of concurrent connections, so it should be set to the number of thermal units. This step automatically clones the `v4l2lepton3`[41] repository into the `/home/pi` directory and installs the `v4l2lepton3` server in the `/home/pi/v4l2lepton3/build` directory.

9. Turn the camera on by running the `./on` script from the `v4l2lepton3` directory in the thermal unit. The script enables SPI and I^2C interfaces and sets the virtual GPIO pin 15 to high, which activates the custom power switch that applies 5 V to the camera. The camera can be turned off using the `./off` script.

10. The Lepton 3.5 should be shipped with the default options set from the factory that do not require any modifications. However, it does not hurt to check that all camera parameters are the way they should using the `get` command method. To set everything correctly, these commands can be executed from the thermal unit's `/home/pi/v4l2lepton3/` directory:

```
$> ./lepton3control.py 1 vid_output_format set raw14
$> ./lepton3control.py 1 oem_output_format set raw14
$> ./lepton3control.py 1 sys_gain_mode set low
$> ./lepton3control.py 1 sys_telemetry_enable set off
$> ./lepton3control.py 1 agc_enable set off
$> ./lepton3control.py 1 rad_enable set on
$> ./lepton3control.py 1 rad_tlinear_enable set on
$> ./lepton3control.py 1 rad_tlinear_scale set 100
$> ./lepton3control.py 1 rad_tlinear_auto_scale set off
```

11. Turn on the `v4l2lepton3` server by running the `/home/pi/v4l2lepton3/build/server` in the background.

12. On the configurating computer, create a `scene.json` file with the borders of the monitored area, camera locations, colors, names, connection data, according to the description in section 6.2.

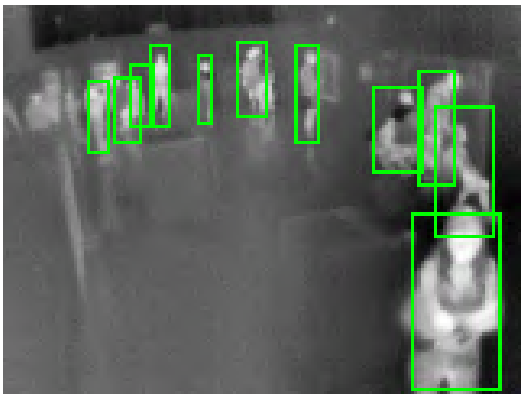
13. Run the visual scene calibration script first only with the `--scene` argument to verify that the scene configuration file is valid. Then, for each camera in the scene, run the calibration script as follows:

```
$> ./thermo-person-detection/calibrate_scene.py -s scene.json
$> ./thermo-person-detection/calibrate_scene.py -s scene.json -c cam1
$> ./thermo-person-detection/calibrate_scene.py -s scene.json -c camX
```

[41] `v4l2lepton3` control and capture library git repository https://gitlab.com/CharvN/v4l2lepton3

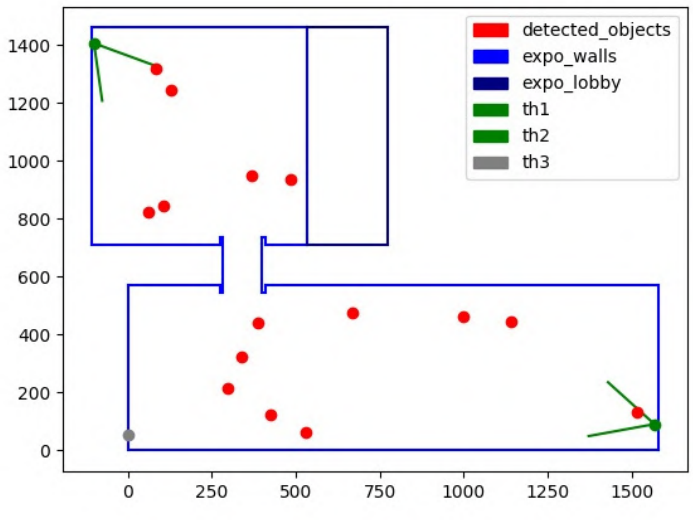
After each run, save and overwrite the configuration `.json` file. The calibrator saves the mapping points for each camera, which are used for constructing its screen to world projection matrix. A calibrated camera can be spotted by its two arms showing its field of view on the scene ground plan.

14. At this point, everything is set up and ready. It is possible to connect to any camera and view its thermal feed in real time using the C++ `v4l2lepton3` client in a `v4l2loopback` virtual video device. Alternatively, the `lepton3client.py` Python implementation can display the feed in an OpenCV window. The extended client `lepton3client_detect.py` can do the same thing with the addition of loading the YOLO detector and highlighting people detected in the feed. Another possibility is to setup a `cron` scheduled task, capture a single frame at a time and save it locally or remotely using the `v4l2lepton3/scripts/capture_periodical.py` script.



(a) th1

(b) th2



(c) ground plan

Figure 7.1: Demonstration of the live detector and locator tool in action.

15. For the actual production usage and real-time monitoring with detection and localization, the control computer may run the `detect_live.py` live detector and localizator from the `thermo-person-detection` directory with the `scene.json` calibration file:

```
$> ./detect_live.py -s scene.json
```

The script builds the scene abstraction according to the configuration file supplied, calibrates all cameras with defined mapping points, connects to their live feed, displays the view from each camera in a separate window with the detector running and plots every detected object into the ground plan. The ground plan is updated with every frame, again, in a separate window. The live detector and locator in action with two active cameras and merged locations of detected people is demonstrated in figure 7.1.

7.2 People flow analysis in the Czech National Museum

There is an ongoing project between the STRaDe[42] research group at the university and the Czech National Museum in Prague regarding people detection, localization, heatmap generation, group detection and other. It is in the museum's interest to learn hidden information about the flow of people through its premises.

Using a thermal system for monitoring people has two great advantages. Firstly, the system performs exactly the same way regardless of light conditions during the day and during the night, which makes it also usable as a security system that can trigger an alarm during the night when there is a person detected while the system is armed. The second advantage is that the resolution and characteristics of thermal imagery make the system unable to perform person recognition as a matter of principle. Consequently, it is safe to use the system on daily basis without potential accusations of spying or personal privacy breach. This however also means that it will not be able to create any valuable evidence when used as a security system.

Figure 7.2: Illustration of the thermal unit installation process in the Czech National Museum in Prague.

Within the scope of this project, the total of three thermal units have been assembled and deployed in two large exhibition halls. The thermal units were installed in the National

[42]STRaDe Research group - part of Department of Intelligent Systems at Faculty of Information Technology of Brno University of Technology https://strade.fit.vutbr.cz/en/

Museum in Prague to overwatch an ongoing exposition with the aim of extracting information about the visitor's movement. More specifically, the outcome of utilizing thermal units at this stage was to extract heatmaps corresponding to visitors common whereabouts. The heatmap can potentially yield hidden information about visitor's interest in specific parts of the exposition, which is a very valuable information for the management of the museum. Figure 7.2 demonstrates the installation process in the Czech National Museum in Prague.

The cooperation is mutually beneficial because in exchange we were allowed to capture an extensive database of thermal images, and therefore, construct the custom thermal dataset for Lepton 3.5, which led to the creation of the new enhanced detector. The new detection and localization system has been used to construct a heatmap of the exposition revealing the most favorite areas. The heatmap is depicted in figure 7.3.

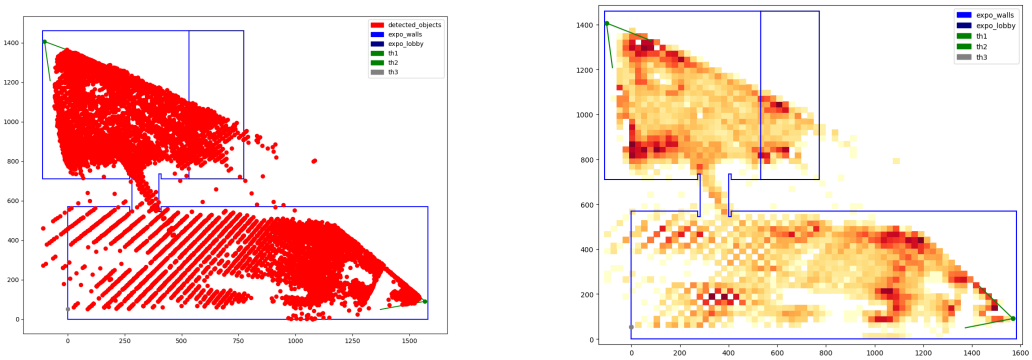


Figure 7.3: Heatmap illustration constructed with the new custom YOLO thermal detector.

The heatmap reveals interesting information about the common whereabouts of people. The dark red color shows spots in the exposition that were occupied by people more often. These hotspots can be associated with specific parts of the exhibition that were particularly interesting to the visitors. The white spots in the middle of the larger hall correspond with no or very small amount of detections, and in fact, these areas were obstructed by show-cases and panels. It is however noticeable that there are larger gaps between detected positions further away from the camera in the lower hall. This has to do with the small resolution of the camera as mentioned previously. One pixel change in the camera view can translate up to one meter large step in the scene, which introduces artifacts into the heatmap. No person can ever be detected in the white regions of the checkerboard pattern visible on the bottom left side of the heatmap in figure 7.3, as they correspond with positions in between two pixels in the thermal image. There was a third camera installed on the opposite side of the exposition hall (`th2` marked gray) that would eliminate this problem, however, not long after the deployment, the camera went offline and could not be revived before the end of the exhibition. The system was supposed to be reinstalled in a new exhibition, unfortunately, this was prevented by the COVID-19 outbreak forcing museums to close down completely.

The museum represents the ultimate challenging environment for the detection and localization system that uses a Lepton camera with relatively small resolution. The modeled scene is large and contains many people. If we considered a smaller room with less people and therefore not so many occlusions, with the new YOLO detector, the new system would not have any problems and yield reasonably accurate positions at all times.

Chapter 8

Conclusion

The goal of the whole project was to utilize small low-cost thermal imaging cameras to create a system solving the problem of detecting and locating people, which shall find its usage in many areas like hazardous area guarding, security systems, people flow analysis for marketing purposes and others. The advantage of using a system based on thermal imaging is that lighting conditions are irrelevant for correct functioning and that the system may be deployed to locations where privacy plays an important role because the camera resolution and thermal imagery in principle prevent facial recognition, yet, allow for detecting of people.

The master's project was a continuation of the bachelor's project documented in [18], and its goal was to eliminate flaws from the final solution of the bachelor's project, implement suggested improvements and extend it into an easily deployable system supporting large complex scenes with multiple thermal cameras.

In the master's project, the Lepton 3 thermal camera has been upgraded to the version 3.5, which supports true radiometry, meaning, that the user does not have to supply a temperature conversion function anymore, which used to be very error-prone. The Orange Pi PC2 computer used in the previous project has been replaced with the more traditional and well maintained Raspberry Pi 3B+. In order to provide a way to manage the camera remotely, a custom power switch circuit has been designed allowing to turn the camera on or off remotely using a GPIO pin on the Raspberry Pi. The printed circuit board with the control circuit placed in between the camera and the Raspberry Pi enables the camera to be turned on and off remotely forcing a full reboot that solves the problem with freezing up. The Lepton 3.5 together with the Raspberry Pi 3B+ and the custom printed circuit board were placed and encapsulated in a custom 3D-printed enclosure box, which represents a single thermal unit. The thermal unit is now robust and can be safely transported, presented or deployed to a new environment.

Since it is expected that more thermal units will be assembled in the future, the process of configuring each thermal unit has been automated using the Ansible tool. A custom Ansible playbook has been created taking care of all installation steps, libraries and dependencies. The Ansible playbook is used to prepare each new thermal unit.

With the upgrade of the camera, new commands had to be implemented in the control software in order to be able to utilize the new features of the camera. At this occasion, the control script has been rewritten almost from scratch, which helped removing some code redundancies when defining new commands, especially commands with multiple methods and various available options. The total of 38 commands with all methods have been added to the control tool.

The old capture library could not be utilized anymore because it was not fast enough to maintain synchronization with the new Lepton 3.5, and most importantly, transferring the raw thermal video from multiple cameras into a single computer for processing turned out to be impossible using the old approach. The capture library has been therefore redesigned according to the client-server model with built-in data transfer via TCP sockets. The thermal unit now runs a server implementation written in C++ and sped up by dual segment and dual frame buffering. When a client connects to the server, it starts pulling raw thermal frames from the camera and sends them directly through the open TCP connection, or optionally, through a `zlib` compressor beforehand. The server does not have synchronization issues, not even with the new Lepton 3.5 thermal camera, recovers from any unexpected event and does its own logging.

As a part of the new `v4l2lepton3` capture library, two clients have been implemented. The first one in C++ preserves the original usage of the virtual video device allowing for a remote thermal unit to be connected as a local video device. The second client is implemented in Python, can be used on its own to display a live thermal feed, but more importantly, it is used in the rest of the detection system as it is very simple, easy to include, yet sufficiently fast.

As a part of an ongoing cooperation with the Czech National Museum in Prague, three thermal units were assembled and deployed to one of their expositions. Using the included single-thermal-frame capture script and `cron` scheduling, an extensive custom thermal database from a complex real-world scene has been acquired over the period of several months. The data has proven that the old simple thermal detector can not be used in such complex environment because of its poor performance. Therefore, it had to be upgraded.

The custom thermal dataset has been manually annotated and merged with the FLIR's official thermal database creating the total of **13,416** thermal images with **53,628** annotated person objects in it. The merged thermal dataset has been used to train several different YOLO object detectors with different network sizes including the recently released state-of-the-art YOLOv4 real-time object detector based on a deep neural network. From the trained models, the YOLOv4-320 has been chosen to be used in the final detection system as it performed the best. On testing thermal images and also on a real-time thermal video, the detector performs incomparably better than the old simple one, and is able to reliably detect a dozen people regardless of their pose in a complex scene with partial occlusions. Everything the detector needs is a single 160×120 thermal image enhanced by temperature filtering and *contrast limited adaptive histogram equalization* (CLAHE). Even though the new YOLOv4 detector is significantly slower than the old detector, it is still faster than any other currently available object detector based on a neural network and fast enough to process the thermal video at full speed (at 8.7 Hz) without decreasing the frame rate.

The mathematics behind reverse-projecting image points into a 3D scene model stayed the same, however, the scene abstraction software has been redone from scratch. The scene is stored in a `JSON` configuration file, supports multiple cameras and multiple polygonal boundaries with different colors and names. The scene abstraction software also includes a visual camera calibrator tool, which simplifies the calibration process of every newly installed camera. The tool connects to a camera, displays its thermal feed in real time, then, the user selects significant points in the thermal image and enters corresponding real-world coordinates of those points. The mapping coordinates are then stored in the configuration file of the scene and used every time the scene is loaded for computing the projection matrix of each configured camera.

The project also contains plenty of supporting scripts that can be used for testing or troubleshooting the system. This includes a script for disconnecting the camera completely, for turning it back on, the Python `v4l2lepton3` client implementation with the detection performed on every frame with the option to save the frame, scripts for single-frame capturing (used with scheduling locally and remotely), a script for running the detector on raw thermal images from a specified directory an so on. The complete system is implemented in the `detect_live.py` file, which loads the preconfigured and calibrated scene, connects to every calibrated camera in the scene, shows their live feed in separate windows and draws locations of detected people in the ground plan of the observed scene.

Within the scope of the cooperation with the Czech National Museum in Prague, the detector has been applied to the collected thermal dataset with the aim of constructing a heatmap of visitor's frequent occurrence within the exposition. The heatmap reveals hidden information about which particular sections of the exposition interest the visitors the most.

To summarize

A new thermal unit has been designed with a custom enclosure box, a new upgraded camera, a custom control circuit and an upgraded host computer. The process of configuring a new thermal unit has been automated using Ansible. The `v4l2lepton3` control and capture library has been redesigned from scratch. The control part supports many more commands with all methods and translated options. The capture part has been split into two parts – a server and a client. The C++ multithreaded implementation of the server has been sped up by double segment buffering, double frame buffering and reduced number of system calls. It does not lose synchronization with the camera, can recover from any kind of error and allows for `zlib` compression. There are two client implementations available. The C++ one uses a virtual video device to bring the remote thermal feed into the local machine for generic processing, the Python implementation is simple and easy to use or include in other projects. It is used in the detection software, but can also be used for quick previews. The scene abstraction software has been redesigned so that now a scene is abstracted in a `JSON` configuration file, supports multiple polygonal boundaries and multiple cameras, which can be calibrated visually using a visual calibrator tool. Finally, the old thermal detector showing poor results in larger scenes with more people has been replaced by the new state-of-the-art YOLOv4 real-time object detector trained on custom thermal dataset that has been created by merging the FLIR's public thermal dataset and a custom one created in the Czech National Museum within the scope of an ongoing cooperation. The new detector is far superior to the old detector and can reliably detect people even in some of the most challenging situations. The final detection system loads a preconfigured scene, connects to all cameras, displays their real-time thermal feeds, and after the detection has been performed, the detected persons are marked in the ground plan representation of the scene. The new detector has been applied to the captured data from the Czech National Museum in Prague with the aim of constructing a heatmap of visitor's movements. The built heatmap proves the capabilities of the detection system and may be beneficial to the management of the museum.

Drawbacks and potential improvements

By having the new detector, the process of estimating locations of detected objects from their bounding boxes becomes the largest area for possible improvements. The accuracy

of an estimated location rapidly decreases with distance because of the low resolution of the thermal camera. About 16 meters away from the camera, a difference of 1 image pixel can easily translate into a 1-meter difference in the scene model. If a bounding box around a person is moved even by a few image pixels, its estimated location can change drastically.

In order to reverse-project an image point into a single point in the scene model, it is necessary to provide some additional information – for example the z coordinate of the searched point. For reverse-projecting feet of a detected person, the z coordinate would be set to 0, alternatively, 170 (cm) would be used for the head position (to represent an average person height). Using the head position is usually less accurate than feet, as the height of people varies naturally. The position of feet works well when the camera is located high above ground or there are not many people in the scene. If neither condition is met, there is a higher possibility that a person would have its feet occluded by a different object. In that case, the system would assume that the person is further away and misplace him completely. When a bounding box is touching the bottom of an image, the system expects that the feet of the detected person is not visible and uses the head position instead. This however does not solve the issue with occlusions.

One solution might be to train the detector to detect two classes – a torso and a whole person. That would require reannotating the whole dataset and longer training with unsure results because the detector would then detect both the torso and the whole person and the system would have to identify that those two detections belong to the same person, which adds more room for error.

Both problems could be solved by adding another camera to observe the same scene from a different angle. The camera would have a priority to localize objects closer to it and both cameras could agree on the same objects. That additional coordinate required for placing the detected object into the scene would be provided from the two cameras using stereo vision. This feature was planned to be included in this project, however after the COVID-19 outbreak, the Czech National Museum was closed down along with our testing environment with several thermal units, therefore, this feature could not be properly implemented nor tested. This feature shall remain on the top of the list of future upgrades.

Another possible improvement could be implementing *allowed* and *blocked area* concept for localization. In the current implementation, there are no rules saying which section of the scene is marked for possible occurrence of people and there is no way to tell which part of the scene is actually observable from which camera. By being able to determine which area of the scene is observable, it would become possible to exclude incorrect locations of detected people that lie outside the observable part of the scene. These outliers are often caused by thermal reflections, large occlusions or the small resolution of the camera.

The future improvements might also tackle with lens distortion of Lepton cameras, as it becomes apparent for some particular modules. Another interesting feature to implement in the system could be person tracking. Each detected person would obtain an ID and his movement through the scene would be stored in a database. This kind of data could be used for a more specific type of people flow analysis where we could, for example, calculate the most typical direction of the movement of people. The candidate technology for object tracking could be the new DeepSORT [63].

Bibliography

[1] *HEXFET® Power P-MOSFET IRF9Z34N Datasheet*. International Rectifier, 1997. Datasheet PD - 9.1485B [Accessed: 20-06-2020]. Available at: https://cdn.instructables.com/ORIG/FNM/2CI9/I5ZZY6BI/FNM2CI9I5ZZY6BI.pdf.

[2] *HEXFET® Power N-MOSFET IRFZ44N Datasheet*. International Rectifier, 2001. Datasheet PD - 94053 [Accessed: 20-06-2020]. Available at: pdf.datasheetcatalog.com/datasheet/irf/irfz44n.pdf.

[3] *ON Semiconductor® NPN amplifier transistor BC337*. Semiconductor Components Industries, LLC, 2013. Publication order number: BC337/D Rev. 8 [Accessed: 20-06-2020]. Available at: https://www.onsemi.com/pub/Collateral/BC337-D.PDF.

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

[5] *FLIR LEPTON® 3.5 Long Wave Infrared (LWIR) Datasheet*. FLIR Commercial Systems, Inc., 2018. Document number: 500-0659-00-09 Rev: 203.

[6] *FLIR LEPTON® Lepton 3 Lepton 3.5 Application Note*. FLIR Commercial Systems, Inc., 2018.

[7] *FLIR LEPTON® Software Interface Description Document (IDD)*. FLIR Commercial Systems, Inc., 2018. Document number: 110-0144-04 Rev: 303.

[8] *Free Thermal Dataset for Algorithm Training*. FLIR Commercial Systems, Inc., 2020. Product page. Available at: https://www.flir.com/oem/adas/adas-dataset-form/.

[9] Airport Suppliers, Acorel SAS. *Automatic High Accuracy Passenger Counting Systems*. 2016. [Online; Accessed: 21-06-2020]. Available at: https://www.airport-suppliers.com/supplier/acorel/.

[10] Albawi, S., Mohammed, T. A. and Al-Zawi, S. Understanding of a convolutional neural network. In: *2017 International Conference on Engineering and Technology (ICET)*. 2017, p. 1–6.

[11] Alhamoud, A., Nair, A. A., Gottron, C., Böhnstedt, D. and Steinmetz, R. Presence detection, identification and tracking in smart homes utilizing bluetooth enabled smartphones. In: *39th Annual IEEE Conference on Local Computer Networks Workshops*. Sep 2014, p. 784–789. DOI: 10.1109/LCNW.2014.6927735.

[12] Ačkar, H., Almisreb, A. and Saleh, M. A Review on Image Enhancement Techniques. *Southeast Europe Journal of Soft Computing*. april 2019, vol. 8. DOI: 10.21533/scjournal.v8i1.175.

[13] Bansal, M., Southall, B., Matei, B., Eledath, J. and Sawhney, H. LIDAR-based Door and Stair Detection from a Mobile Robot. *Proceedings of SPIE - The International Society for Optical Engineering*. april 2010, p. 2–. DOI: 10.1117/12.849926.

[14] Bochkovskiy, A. *YOLOv4 - Neural Networks for Object Detection (Windows and Linux version of Darknet)*. 2020. Github repository. Available at: https://github.com/AlexeyAB/darknet.

[15] Bochkovskiy, A., Wang, C.-Y. and Liao, H.-Y. M. *YOLOv4: Optimal Speed and Accuracy of Object Detection*. 2020. Available at: https://arxiv.org/abs/2004.10934.

[16] Bodla, N., Singh, B., Chellappa, R. and Davis, L. S. Improving Object Detection With One Line of Code. *CoRR*. 2017, abs/1704.04503. Available at: http://arxiv.org/abs/1704.04503.

[17] Bradski, G. and Kaehler, A. *Learning OpenCV*. O'Reilly Media, Inc., september 2008. ISBN 978-0-596-51613-0. First edition.

[18] Charvát, M. *Detection of People in Room Using Low-Cost Thermal Imaging Camera*. Jun 2018. Bachelors thesis. Brno University of Technology, Faculty of InformationTechnology. Supervisor prof. Ing., Dipl.-Ing. Martin Drahanský, Ph.D.

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

[20] DeMenthon, D. F. and Davis, L. S. Model-Based Object Pose in 25 Lines of Code. *International Journal of Computer Vision*. june 1995, vol. 15, no. 2, p. 123–141.

[21] Devert, A. *Matplotlib Plotting Cookbook*. Packt Publishing, 2014. ISBN 1849513260, 9781849513265.

[22] Girshick, R. B. Fast R-CNN. *CoRR*. 2015, abs/1504.08083. Available at: http://arxiv.org/abs/1504.08083.

[23] Girshick, R. B., Donahue, J., Darrell, T. and Malik, J. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*. 2013, abs/1311.2524. Available at: http://arxiv.org/abs/1311.2524.

[24] Gough, A. *Image Annotation Types For Computer Vision And Its Use Cases* [Hackernoon – An Independent Tech Media Publication platform]. 2019. [Online; Accessed: 06-06-2020]. Available at: https://hackernoon.com/illuminating-the-intriguing-computer-vision-uses-cases-of-image-annotation-w21m3zfg.

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

[26] Hall, D. *Ansible Configuration Management*. Packt Publishing, 2013. ISBN 1491915323, 9781491915325.

[27] Hartley, R. *Multiple View Geometry in Computer Vision*. 2nd ed. Cambridge University Press, 2004. ISBN 354049698X.

[28] Henderson, G. *Wiring Pi- GPIO Interface library for the Raspberry Pi*. 2020. [Online; Accessed: 10-01-2020]. Available at: http://wiringpi.com/.

[29] Herrmann, C., Ruf, M. and Beyerer, J. *CNN-based thermal infrared person detection by domain adaptation. Autonomous Systems: Sensors, Vehicles, Security, and the Internet of Everything. International Society for Optics and Photonics*. 2018. p. 1064308.

[30] Hochstein, L. *Ansible Up and Running*. O'Reilly Media, Inc., 2015. ISBN 1491915323, 9781491915325.

[31] Ivanov, B., Ruser, H. and Kellner, M. Presence detection and person identification in Smart Homes. [Neubiberg, University of Bundeswehr Munich and Passau, FORWISS, University Passau]. Jul 2014.

[32] Kamal, A. *YOLO, YOLOv2 and YOLOv3: All You want to know* [Medium.com – online publishing platform]. 2019. [Online; Accessed: 18-06-2020]. Available at: https://medium.com/@amrokamal_47691/yolo-yolov2-and-yolov3-all-you-want-to-know-7e3e92dc4899.

[33] Krizhevsky, A., Sutskever, I. and Hinton, G. E. ImageNet Classification with Deep Convolutional Neural Networks. In: Pereira, F., Burges, C. J. C., Bottou, L. and Weinberger, K. Q., ed. *Advances in Neural Information Processing Systems 25*. Curran Associates, Inc., 2012, p. 1097–1105. Available at: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[34] Kroah Hartman, G. *Linux Kernel in a Nutshell*. O'Reilly Media, Inc.,, 2006. ISBN 0596100795,9780596100797.

[35] Kyle McDonald. *Structured Light 3D Scanning* [Instructables website], 29. Dec 2009. [Online; Accessed: 21-06-2020]. Available at: http://www.instructables.com/id/Structured-Light-3D-Scanning/.

[36] Kyle Simek. *Dissecting the Camera Matrix: Part 1,2,3 – Extrinsic/Intrinsic Camera Matrix* [Kyle Simek's Computer Vision Blog]. 2012-2013. [Online; Accessed: 18-06-2020]. Available at: http://ksimek.github.io/2012/08/14/decompose/.

[37] Lienhart, R. and Maydt, J. An extended set of Haar-like features for rapid object detection. In: *Proceedings. International Conference on Image Processing*. 2002, vol. 1.

[38] Mallick, S. *Head Pose Estimation using OpenCV and Dlib* [Learn OpenCV blog], 26. Sep 2016. [Online; Accessed: 25-05-2018]. Available at: https://www.learnopencv.com/head-pose-estimation-using-opencv-and-dlib/.

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

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

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

[42] 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]. Available at: https://peoplecounting.pro/product/condor-8-people-counter-with-data-logging/.

[43] Pokhrel, S. *Image Data Labelling and Annotation* [Towards Data Science – A Medium publication sharing concepts, ideas, and codes.]. 2020. [Online; Accessed: 06-06-2020]. Available at: https://towardsdatascience.com/image-data-labelling-and-annotation-everything-you-need-to-know-86ede6c684b1.

[44] Raj, S., Raj, S. and Kumar, S. An Improved Histogram Equalization Technique for Image Contrast Enhancement. In:. July 2015.

[45] Raspberry Pi Foundation. *Raspberry Pi 3 Model B+ product description*. 2016. [Online; Accessed: 26-05-2020]. Available at: https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/.

[46] Raval, S. *YOLO Object Detection*, 16. Nov 2017. [Online; Accessed: 21-06-2020]. Available at: https://github.com/llSourcell/YOLO_Object_Detection/blob/master/YOLO%20Object%20Detection.ipynb.

[47] Redmon, J., Divvala, S., Girshick, R. and Farhadi, A. You Only Look Once: Unified, Real-Time Object Detection. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2016.

[48] Redmon, J., Divvala, S. K., Girshick, R. B. and Farhadi, A. You Only Look Once: Unified, Real-Time Object Detection. *CoRR*. 2015, abs/1506.02640. Available at: http://arxiv.org/abs/1506.02640.

[49] Redmon, J. and Farhadi, A. YOLO9000: Better, Faster, Stronger. *CoRR*. 2016, abs/1612.08242. Available at: http://arxiv.org/abs/1612.08242.

[50] Redmon, J. and Farhadi, A. YOLOv3: An Incremental Improvement. *CoRR*. 2018, abs/1804.02767. Available at: http://arxiv.org/abs/1804.02767.

[51] Ren, S., He, K., Girshick, R. B. and Sun, J. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *CoRR*. 2015, abs/1506.01497. Available at: http://arxiv.org/abs/1506.01497.

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

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

[54] Schulte, S. and Demuynck, S. *Thermal-image based object detection and heat map generation systems and methods. U.S. Patent Application No 16/014,112.* 2018.

[55] Singh, S. *Beginning Google Sketchup for 3D Printing (Expert's Voice in 3D Printing).* Apress, 2010. ISBN 9781430233619, 9781430233626.

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

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

[58] The Economist. *In-store detecting – A new industry has sprung up selling "indoor-location" services to retailers* [The Economist newspaper], 24. Dec 2016. [Online; Accessed: 21-06-2020]. Available at: https://www.economist.com/business/2016/12/24/a-new-industry-has-sprung-up-selling-indoor-location-services-to-retailers.

[59] 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.

[60] Viola, P. and Jones, M. Rapid object detection using a boosted cascade of simple features. In: *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001.* 2001, vol. 1.

[61] Viola, P. and Jones, M. Robust Real-time Object Detection. In: *International Journal of Computer Vision.* 2001.

[62] Wojke, N. and Bewley, A. Deep Cosine Metric Learning for Person Re-identification. In: IEEE. *2018 IEEE Winter Conference on Applications of Computer Vision (WACV).* 2018, p. 748–756. DOI: 10.1109/WACV.2018.00087.

[63] Wojke, N., Bewley, A. and Paulus, D. Simple Online and Realtime Tracking with a Deep Association Metric. In: IEEE. *2017 IEEE International Conference on Image Processing (ICIP).* 2017, p. 3645–3649. DOI: 10.1109/ICIP.2017.8296962.

[64] Yadav, G., Maheshwari, S. and Agarwal, A. Contrast limited adaptive histogram equalization based enhancement for real time video system. In: *2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI).* 2014, p. 2392–2397.