VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

DIPLOMOVÁ PRÁCE

Brno, 2020 Bc. Matúš Zakarovský

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY
DEPARTMENT OF CONTROL AND INSTRUMENTATION

ROBOTICKÉ NÁSLEDOVÁNÍ OSOBY POMOCÍ NEURONOVÝCH SÍTÍ
ROBOTIC TRACKING OF A PERSON USING NEURAL NETWORKS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR
Bc. Matúš Zakarovský

VEDOUCÍ PRÁCE
SUPERVISOR
prof. Ing. Luděk Žalud, Ph.D.

BRNO 2020

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

Master's Thesis

Master's study field **Cybernetics, Control and Measurements**

Department of Control and Instrumentation

Student: Bc. Matúš Zakarovský ***ID:*** 186391

Year of study: 2 ***Academic year:*** 2019/20

TITLE OF THESIS:

Robotic Tracking of a Person using Neural Networks

INSTRUCTION:

The thesis is aiming on the realization of the human tracking mobile robot with Nvidia Jetson hardware and neural networks usage.

1. Prepare and describe robotic platform that will be used for the future solution.
2. Make a research in the field of human tracking and recognition with neural networks application. Test at least two different neural network architectures.
3. Prepare the Nvidia Jetson Nano platform for neural network deployment and test the selected model on the Jetson computer.
4. Get familiar with Robotic Operation System and use it as a part of future software implementation.
5. Realize the software that will be able to detect a human and estimate relative position wrt. robot. This solution will be based on previously chosen neural network.
6. Create software that will be able to track and follow person, based on detection system from previous paragraph.

RECOMMENDED LITERATURE:

Goodfellow, I., Bengio, Y. and Courville, A., 2016. Deep learning. MIT press.

Date of project specification: 3.2.2020 ***Deadline for submission:*** 1.6.2020

Supervisor: prof. Ing. Luděk Žalud, Ph.D.

doc. Ing. Václav Jirsík, CSc.
Subject Council chairman

ABSTRACT

The main goal of this thesis was to create a software solution based on a neural network to enable detection of a person and its subsequent following. This was achieved via completion of the points of the assignment. First, a hardware solution and used libraries and application programming interfaces were described as well as the robotic platform supplied by the Robotics and AI group of BUT Department of Control and Instrumentation upon which the robot was built on. Next, a research of various neural networks used for person detection was conducted. Four detectors were described in detail. Some of them were tested on either a PC or a NVIDIA Jetson Nano computer. Afterwards, a software solution consisting of five programs was created to achieve goals such as, detection of the person using ped-100 neural network, real-world position with reference to the robot estimation using monocular camera and robot control to successfully follow a target. The output of this thesis is a robotic platform able to detect and follow a person that can be used in a real-world applications.

KEYWORDS

Artificial Intelligence, Robotics, Person tracking, Neural Networks, AI, CNN, FPN, ROS, Jetson Nano

ABSTRAKT

Hlavným cieľom práce bolo vytvorenie softvérového riešenia založeného na neurónových sieťach, pomocou ktorého bolo možné detegovať človeka a následne ho nasledovať. Tento výsledok bol dosiahnutý splnením jednotlivých bodov zadania tejto práce. V prvej časti práce je popísaný použitý hardvér, softvérové knižnice a rozhrania pre programovanie aplikácií (API), ako aj robotická platforma dodaná skupinou robotiky a umelej inteligencie ústavu automatizácie a meracej techniky Vysokého Učenia Technického v Brne, na ktorej bol výsledný robot postavený. Následne bola spracovaná rešerš viacerých typov neurónových sietí na detekciu osôb. Podrobne boli popísané štyri detektory. Niektoré z nich boli neskôr testované na klasickom počítači alebo na počítači NVIDIA Jetson Nano. V ďalšom kroku bolo vytvorené softvérové riešenie tvorené piatimi programmi, pomocou ktorého bolo dosiahnuté ciele ako rozpoznanie osoby pomocou neurónovej siete ped-100, určenie reálnej vzdialenosti vzhľadom k robotu pomocou monokulárnej kamery a riadenie roboty k úspešnému dosiahnutiu cieľa. Výstupom tejto práce je robotická platforma umožnujúca detekciu a nasledovanie osoby využiteľné v praxi.

KĽÚČOVÉ SLOVÁ

Umelá inteligencia, Robotika, Nasledovanie osôb, Neurónové siete, AI, CNN, FPN, ROS, Jetson Nano

ZAKAROVSKÝ, Matúš. *Robotic Tracking of a Person using Neural Networks*. Brno, 2020, 71 p. Master's Thesis. Brno University of Technology, Faculty of Electrical Engineering and Communication, Department of Control and Instrumentation. Advised by prof. Ing. Luděk Žalud, Ph.D.

ROZŠÍRENÝ ABSTRAKT

Cieľom tejto práce je navrhnutie softvérového riešenia založeného na neurónových sieťach, pomocou ktorého bolo možné detegovať človeka a následne ho nasledovať. V prvej časti tejto práce bol detailne popísaný použitý hardvér, softvérové knižnice a API, ako aj robotická platforma dodaná skupinou robotiky a umelej inteligencie ústavu automatizácie a meracej techniky Vysokého Učenia Technického v Brne, na ktorej bol výsledný robot postavený. Konkrétne sa jednalo o popis počítača špeciálne určeného na aplikácie umelej inteligencie, NVIDIA Jetson Nano, skladajúceho sa z výpočtového modulu a vývojovej dosky. Ďalej boli popísané dve kamery, Raspberry Pi Camera module V2 a Apeman A80. Objasnené bolo aj množstvo softvérových knižníc, ako napríklad OpenCV a V4L2, ktoré boli použité pri vývoji programov. Ďalej bol podrobne rozobratý Robot Operating System, jeho funkcie a v ňom obsiahnuté nástroje.

Druhá časť práce pozostávala z rozobratia danej problematiky a vysvetlením ako sa k daným problémom bude pristupovať. Tretia kapitola bola venovaná rešerši štyroch neurónových sietí umožnujúcich detekciu osôb. Išlo o dve jednostupňové siete, DarkNet-53 a RetinaNet, obe testované na datasete COCO, jednu zbierku neurónových sieti OpenPose a ako posledná bola sieť PedNet vyvinutá špeciálne iba na detekciu ľudí, chodcov. Sieť DarkNet-53 je všeobecne veľmi rýchla sieť určená na rozpoznávanie väčšieho množstva tried vrátane človeka. Napriek tomu je až príliš náročná na použitie na hardvéri robota a detekcia viac než 80 tried je v danom prípade považovaná za zbytočnú. RetinaNet je v mnohom podobná DarkNet-53, taktiež sa jedná o sieť navrhnutú na dataset COCO a tiež je táto sieť jednostupňový detektor. Napriek tomu je táto sieť oveľa pomalšia. Práve kvôli tomuto faktoru bola táto sieť zavrhnutá a nedostala sa do fázy testovania. Kolekcia neurónových sietí OpenPose, je schopná rozpoznať až 135 bodov na ľudskom tele a následne tak určiť postoj daného človeka. Táto sieť je ale o trochu pomalšia ako sieť DarkNet-53, ale napriek tomu existuje jej varianta optimalizovaná pre zariadenia NVIDIA Tegra, ktorej súčasťou je aj použitý Jetson Nano. Jej použitie by taktiež umožnilo implementáciu pokročilých funkcií riadenia robota na základe giest alebo postoja človeka. PedNet je ako jediná vytvorená špecificky iba na detekciu prítomnosti človeka v obraze. Táto sieť veľmi nezaťažuje zdroje počítača, je rýchla a je optimalizovaná aj na Jetson Nano.

Nasledujúca kapitola sa venovala testovaniu týchto sieti, buď na klasickom počítači s operačným systémom Windows 10 s dedikovanou grafickou kartou EVGA NVIDIA GeForce 1080SC alebo na počítači NVIDIA Jetson Nano. Sieť OpenPose bola testovaná na videu s množstvom osôb pričom maximálne detegovala naraz až 16 ľudí. Na druhej strane na jej plynulý chod na Jetson Nano by bol potrebný oveľa vyšší výkon grafického procesoru. Sieť PedNet bola schopná deteko-

vať ľudí s uspokojivou presnosťou a jej hlavnou prednosťou bola rýchlosť spracovania snímkov posielaných kamerov. Na základe testovania bola teda zvolená sieť PedNet, konkrétne jej varianta ped-100, ktorá bola následne implementovaná vo vytvorenom softvéri.

Kapitola číslo päť sa venovala umiestneniu a nastaveniu jednotlivých prvkov na robotickej platforme a detailnému popisu navrhnutých programov. Išlo o programy riadiace správanie kamery, jej prepojenia na neurónovú sieť, následné určenie vzdialenosti najbližšieho detekovaného cieľa a určenie požadovanej lineárnej a uhlovej rýchlosti plánovacou jednotkou a jej následná transformácia na rýchlosť jednotlivých kolies pomocou riadiacej jednotky motorov. Systém určenia vzdialenosti človeka od robota pomocou monokulárnej kamery využíva kalibračnú mriežku, ktorá sa skladá z piatich vertikálnych a desiatich horizontálnych čiar. Taktiež bol uvedený postup výpočtu príslušnosti k najbližšej čiare a následná aproximácia vzdialenosti v osi X a Y od danej priamky. Výsledná vypočítaná hodnota mala maximálnu absolútnu chybu dva centimetre pre každú os.

Posledná kapitola sa venovala testovaniu robota v rozličných podmienkach s rôznymi úlohami. Počas testovania bola objavená chyba *cuda unspecified launch failure*, ktorá bola dôsledkom inkompatibility reprezentácie obrazu medzi knižnicou OpenCV a CUDA. Túto chybu sa následne podarilo odstrániť implementáciou ramdisku, na ktorý bol daný snímok zapísaný a následne načítaný. Fáza testovania bola rozdelená na štyri testy, a až po úspešnom absolvovaní predchádzajúceho testu bolo možné prejsť na ďalší. Prvý test mal overiť, že robot ostane stáť ak sa rozpoznaný človek nachádza vo vzdialenosti menšej ako 180 centimetrov. V druhom teste mal robot nasledovať priamu trajektóriu k cieľu a zastaviť. Tretí test pozostával z osoby, ktorá sa nachádzala od robota pod určitým uhlom a robot mal nasledovať zakrivenú dráhu. Posledný mal za úlohu overenie schopnosti robota nasledovať pohybujúci sa cieľ.

Výsledný návrh robota uspel vo všetkých testoch a je schopný rozpoznať človeka v obraze, určiť jeho vzdialenosť od robota a potom ho nasledovať.

DECLARATION

I declare that I have written the Master's Thesis titled "Robotic Tracking of a Person using Neural Networks" independently, under the guidance of the advisor and using exclusively the technical references and other sources of information cited in the thesis and listed in the comprehensive bibliography at the end of the thesis.

As the author I furthermore declare that, with respect to the creation of this Master's Thesis, I have not infringed any copyright or violated anyone's personal and/or ownership rights. In this context, I am fully aware of the consequences of breaking Regulation § 11 of the Copyright Act No. 121/2000 Coll. of the Czech Republic, as amended, and of any breach of rights related to intellectual property or introduced within amendments to relevant Acts such as the Intellectual Property Act or the Criminal Code, Act No. 40/2009 Coll., Section 2, Head VI, Part 4.

Brno 1. 6. 2020 .....

author's signature

ACKNOWLEDGEMENT

I would like to thank my supervisor, professor Ing. Luděk Žalud, Ph.D., and to my consultant, Ing. Adam Ligocki, for professional guidance, consultations, patience and contributing ideas to my thesis.

Brno 1. 6. 2020 .....

author's signature

Contents

List of Figures

List of Tables

Listings

Introduction

Artificial Intelligence. This phrase can be heard anywhere nowadays. Its everywhere. In your phone. It can be in your car. It is even implemented in your favourite streaming service. It is one the moving factors of today's society and it is getting more and more integrated in to the everyday life. So far it has been greatly beneficial to our lives. It can help detect cancerous tissue in medicine or predict the development of the stock market. This technology has a great potential in the future as it is still being developed and further improved.

This thesis is focused on one aspect of AI, concretely on use of neural networks in person recognition. Its major goal is to utilize one of such networks to create a robot capable of finding the person, estimating its distance to the robot and following the detected person. The minor but very necessary goal is to create a way of estimating the correct distance with minimal deviation based on the input of monocular camera.

The first section will describe the hardware used. The second will further analyze the problems to achieve the goal. The third section will be dedicated to neural network research. In fourth, some of these networks will be tested for performance and accuracy, and one best performing network will be chosen for final implementation. Robot configuration and software solution will be described in details in the fifth chapter. The last section will be dedicated to the testing of the robot and result analysis.

1 Platform

This chapter describes the platform used in this thesis, its hardware and its key software. The software part is mainly focused on TensorRT framework and Robot Operating System.

1.1 Hardware

In this section, the hardware options used in the solution are listed. The main parts are Jetson Nano Kit made by NVIDIA, Raspberry Pi Camera and an custom-made robotic platform supplied by Robotics and AI group of Department of Control and Instrumentation.

1.1.1 NVIDIA Jetson Nano Developer Kit

This kit consist of Jetson Nano module and a carrier board. Jetson Nano module was specifically created for AI applications, such as computer vision, segmentation and speech processing. It supports parallel neural network execution. Based on [4], there are two possible options for power supply. The first option utilizes 5V power and current of 2A via micro USB connector. The second one, with a barrel jack connector, uses the same voltage, but is capable of supplying of higher current up to 4 amps. This should be used when peripherals draw more current than 2A. In this thesis, the 4 amp power supply is used. The kit also has two power modes, the default is 10W and the other is 5W. This is done by capping the GPU and CPU frequency and the number of active CPU cores.

Jetson Nano Module

It is a System-on-Module (SoM), which is based on NVIDIA Tegra X1 Series System-on-Chip (SoC). This SoC consists of quad core ARM Cortex A-57 MPCore processor clocked at 1.43 GHz, Maxwell architecture GPU with 128 CUDA cores working on a maximum frequency of 921 MHz. It has a performance of 472 GFLOPS at FP16 precision. Furthermore it has integrated 4 GB of 64-bit LPDDR4 memory with operating frequency of 1600 MHz and peak bandwidth of 25.6 GB/s. There are multiple communication interfaces available, including GPIO, I2C, I2S, SPI and UART. Nano module is connected to the carrier board via SO-DIMM connector. More details about the module are available here [6].

Fig. 1.1: NVIDIA Jetson Nano Developer Kit. The Nano module is mounted on the top with the heat sink attached. The bottom part is the carrier board [4]

Carrier Board

The carrier board provides peripheral access, expansion slots and ports. This includes HDMI and DP video outputs, gigabit Ethernet, four USB 3.0 type A slots, M2 Key E slot for WiFi, PCIe (x1), USB 2.0 or I2C. There is also a MIPI Camera Serial Interface (CSI) slot and 40-pin expansion header. Pins are separated into two categories. Power pins and signal pins. There are two 3.3V and two 5V pins. It is also possible to power the kit via the 5V pins. Signal pins use 3.3 volts. Pins numbered 3, 5, 27 and 28 are reserved for I2C SDA and SCL, and pins 8 and 10 are for UART TX and RX respectively. Depending on the power supply current, the carrier board consumes 0.5W at 2A and 1.25W at 4A without any peripherals attached. The layout of the board is described more detail in [4].

1.1.2 Cameras

RPi V2

The Raspberry Pi Camera Module V2 [13] is connected to Jetson Nano via MIPI Camera Serial Interface. It is based on SONY IMX219 CMOS image sensor with Exmor R back-illuminated sensor. With 8 megapixels, its maximum still image resolution is 3280 by 2464 pixels and it supports multiple video modes, such as 1080p at 30 frames per second or 720p at 60 frames per second. The camera has a fixed focal length of 3.04 millimeters, 62.2 degrees horizontal field of view and 48.8 degrees vertical field of view. There are few beneficial software features implemented. These features include different metering modes, automatic white balance, exposure modes and trigger types. Following formats are supported by the camera, which could be later utilized in this thesis, JPEG, JPEG + RAW or YUV420 for images

and raw h.264 for videos.

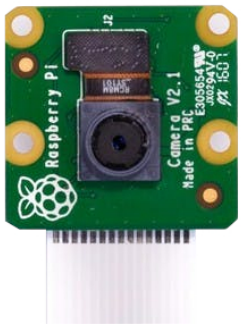


Fig. 1.2: Raspberry Pi Camera Module V2 [13]

Apeman 4K

This action camera is connected to the Jetson Nano via USB interface. It uses 20 megapixel Sony sensor and features up to 170° field of view. It has built-in anti-shaking and electronic image stabilisation (EIS). This camera has variable focal length based on the various possible resolution and view angle settings and also supports control via WiFi. The possible resolution modes are:

Image modes

- 20M
- 16M
- 12M
- 10M
- 8M
- 5M
- 3M

Video modes

- 4K@24fps
- 1440P@30fps
- 1080P@30/60fps
- 720P@30/60/120fps

For video compression it utilizes H.264 codec. The view angle can be set up from 170 degrees down to 90 degrees in approximately 30 degree increments. In this thesis 1080p mode is used with the frame rate of 30 fps and the view angle is set to 110°.

Fig. 1.3: Apeman A80 action camera [23]

1.1.3 Robotic Platform

The robotic platform supplied by BUT Department of Control and Instrumentation represents the mobile base of the robot. It is very similar to the KAMbot platform and, in fact, many components are shared. The main structural element of the platform is built from two plates of plywood connected together by screws and standoffs. On the bottom plate, motors, battery and control electronics is located. The top plate should serve to accommodate the camera stand and Jetson Nano computer.

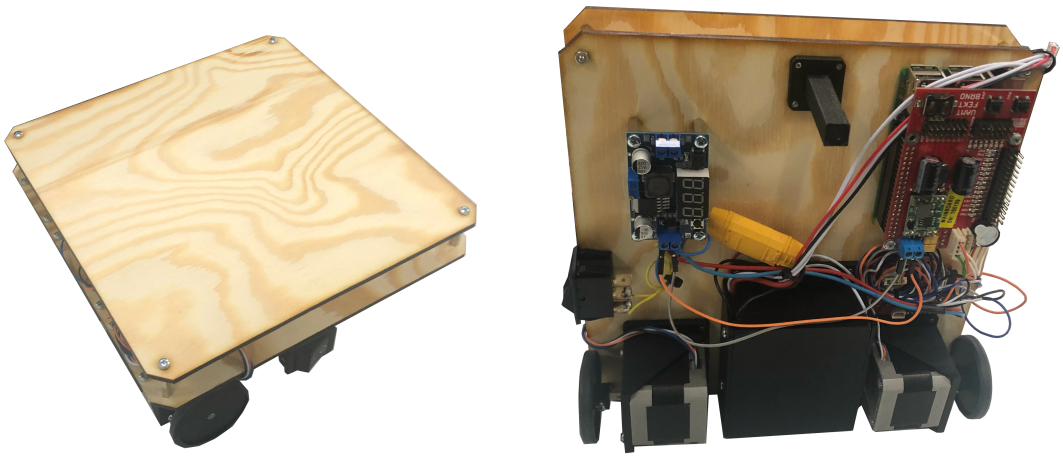


Fig. 1.4: Robotic platform

In following paragraphs, parts used in the robotic platform will be further discussed.

Raspberry Pi 4 Model B

This computer consists of Broadcom BCM2711 quad core cortex A-72 processor clocked at 1.5 gigahertz, 2 gigabytes of LPDDR4 3200 MHz SDRAM, 2.4 and 5 GHz IEEE 802.11ac wireless module and Bluetooth 5.0 module. It also has one gigabit RJ-45 port and 40 GPIO header that is utilized by the MAINBOARD shield. Other ports will not be mentioned, because they are not important to the thesis. The currently running operating system is Rasbian Buster.

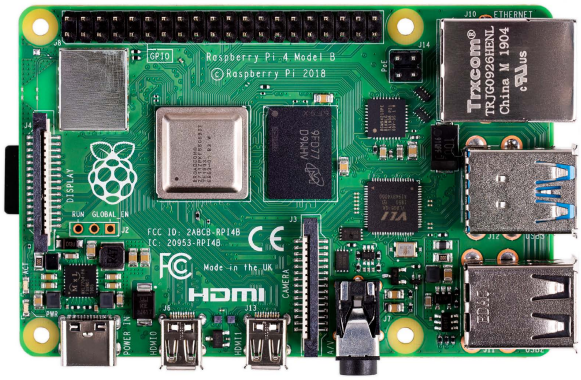


Fig. 1.5: Raspberry Pi 4 Model B [14]

As the main processing unit of the platform, Raspberry Pi is responsible of processing data acquired via ethernet port from Jetson Nano. There are two possible messages that can be received. The speed message, $spd,<left_speed>,<right_speed> to which, same response is sent back. The other possible message is the odometry message, $odm,. To this message, Raspberry Pi reponds via message in format, $odm,<left_wheel>,<right_wheel>, where left and right wheel represent number of encoder pulses since the last reading. This information is determined via communication with KM2 motor module utilizing I2C interface.

MAINBOARD Shield

The main goal of this shield[18] in this thesis is to provide power to the Raspberry and to serve as an interface expansion board. The input voltage has to be in range between 7 - 20 volts and t has reverse polarity protection implemented as well. Then, there are three 3.3V logic I2C headers with 5V power supply, 16 programmable GPIO pins, 7 PWM and 8 10-bit ADC pins connected via I2C and one Real-Time Clock module connected through SPI.

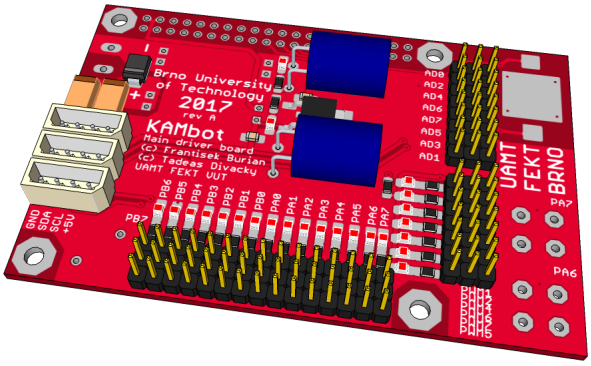


Fig. 1.6: RPi MAINBOARD shield developed at BUT. The three ports on the left are I2C interface, the bottom pins are GPIO and their respective power supply, on the bottom right, there are PWM pins and on the top are analog pins [18]

KM2 Motor Module

Another custom made module by doctor Burian. It is responsible for controlling two stepper motors. For this it uses DRV8825 drivers. It communicates with Raspberry Pi via I2C interface and is available at address 0x71.

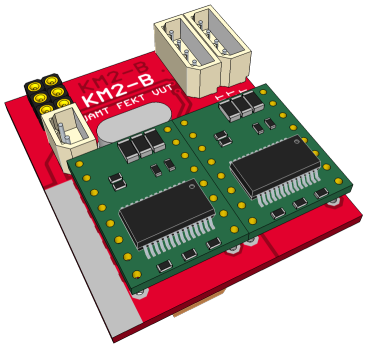


Fig. 1.7: KM2 motor module [18]

The platform is also fitted with two Microcon two-phase stepper motors, four cell Li-ion battery, and 20 watt DC-DC converter to power Jetson Nano.

1.2 Operating System and APIs

All the necessary software, including the OS is supplied by the NVIDIA JetPack software development kit and is available on the official NVIDIA website. It consists

of NVIDIA L4T operating system, TensorRT, cuDNN, CUDA, Multimedia API and Computer Vision package. These libraries, APIs and packages will be further explained in the following pages. The information is based on [5].

1.2.1 Linux for Tegra (L4T)

NVIDIA L4T [5, 7] provides bootloader, Linux kernel version 4.9, drivers and filesystem derived from Ubuntu 18.04 Bionic Beaver. It has 64-bit user space and provides support for OpenGL and Vulkan API. The L4T version at the time of writing was 32.3.1.

1.2.2 NVIDIA TensorRT

What is TensorRT?

TensorRT is CUDA based programmable platform for deep learning inference optimization. It is built on a C++ library, which provides high performance inference on GPUs. It should complement major neural network training frameworks to optimize run of trained network.

What it does?

The process takes a trained neural network runs it trough TensorRT optimizer, and this optimized network is now running using TensorRT Runtime Engine.

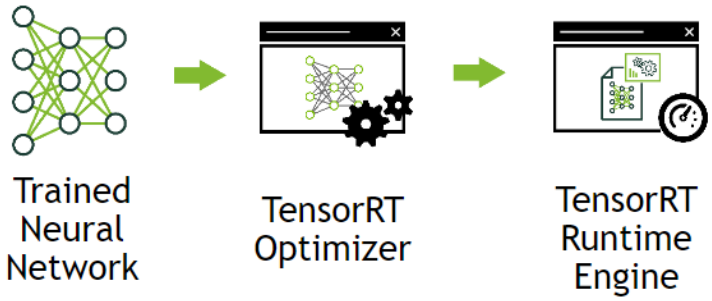


Fig. 1.8: TensorRT Optimization Process - Depending on the neural network model, TensorRT optimizer tries to fit the network to the specific hardware available, making the network run faster with same accuracy [9]

As stated in [9], TensorRT combines network layers, optimizes kernel selection to improve throughput, latency, power efficiency and memory consumption. It is also possible to change network precision to a lower one, if required, which provides

further improvement in performance, whilst reducing memory requirements. Specific optimizations consist of removal of layers which outputs are not used, deletion of operations similar to no-op instruction and fusing convolution, bias and ReLU layers together. Based on chosen precision, a small change in accuracy can occur while using the INT8 precision. FP32 and FP16 should yield results similar to those during training.

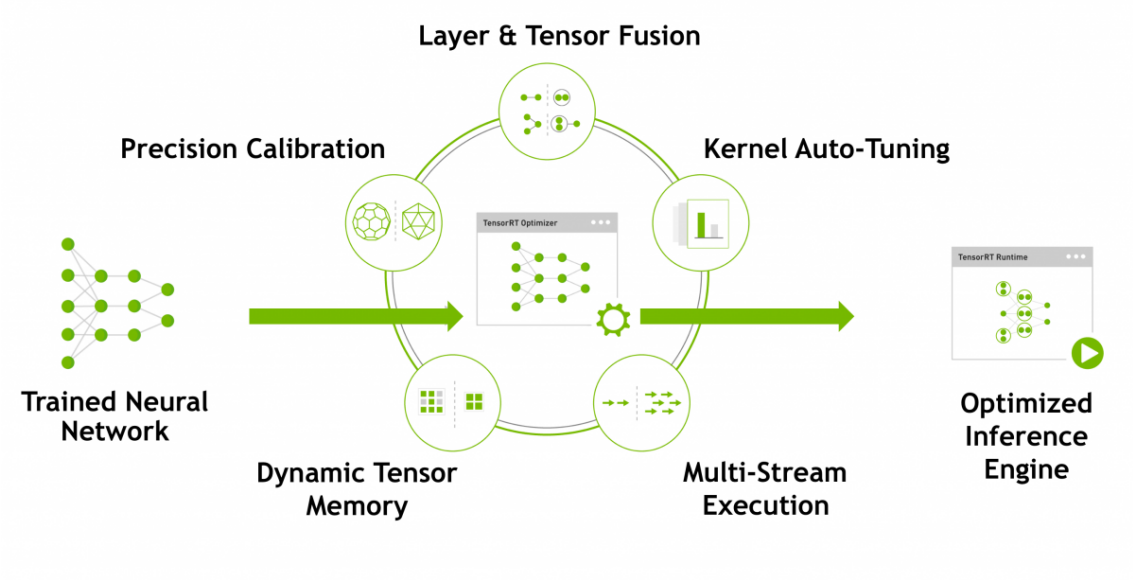


Fig. 1.9: TensorRT Diagram - Neural network is converted to desired precision, allocated in faster tensor memory, its layers are fused together until the fastest network on the specified hardware is found. [8]

TensorRT Benefits

Because neural networks are trained using specific frameworks like TensorFlow, Caffe2, Microsoft Cognitive Toolkit or PyCharm, they leave some unnecessary data in the saved network, such as framework overhead. TensorRT compresses and optimizes this network, removes the overhead and deploys the network as a runtime. For further improvements in latency and throughput, the neural network is normalized and converted to matrix math on the specified precision. There are three supported precision modes. FP32, FP16 and INT8. In [9] is stated, that there are five critical factors used for software measuring. Throughput, Efficiency, Latency, Accuracy and Memory Usage. Throughput defines the volume of output within specified time period. Efficiency is how much throughput is delivered per unit of power. Latency represents inference execution time, accuracy determines the NN's ability to yield the correct output and memory usage is a sum of reserved host and device memory needed for inference, which is highly dependent on used algorithms.

1.2.3 cuDNN

CUDA Deep Neural Network library implements primitives used in deep learning, including convolutions, activation functions and transformation of tensors.

1.2.4 CUDA

CUDA toolkit is an C/C++ IDE used in development of graphics-accelerated programs. It consists of debug and optimization tools, math libraries and a compiler for NVIDIA GPUs.

1.2.5 Multimedia API

It is a low-level application programming interface package made up from two APIs. The camera application, libargus, and the sensor driver, V4L2. The first controls the camera parameters and enables multiple camera support. The latter is used for video encoding and decoding, format conversion or changing of video parameters.

1.2.6 Computer Vision

For computer vision and image processing applications, one package and two libraries are included in JetPack. VisionWorks, OpenCV, and Vision Programming Interface (VPI).

1.3 Robot Operating System

ROS is not an operating system. It consists of many frameworks which simplify development of software used in robotics. ROS implements services like low-level device control, inter-process communication and hardware abstraction. The main client libraries are written in three programming languages, C++, Python and Lisp. There are many packages created by the community around ROS. Some of them implement hardware drivers, SLAM or various algorithms used in robotics, for example path finding. It was created to tackle the goal of achieving robust general purpose robotic software that would be readily available to users.

1.3.1 Structure

Robot Operating System organizes all the processes in a graph architecture. In this architecture, every node a represents a single process connected through topics. Node is assigned a name, which has to be registered with the ROS master. ROS

master is a process that controls all other nodes and creates a communication path for them. The communication is implemented in a peer-to-peer manner.

1.3.2 Core Components

At the lowest level, ROS is responsible to pass publish/subscribe messages. It can capture and play them later on also. These messages are mostly used as an asynchronous procedure calls, but if the synchronous calls are necessary another system called services is implemented. While the separate nodes work as a distributed system, it is possible to change their parameters via global key-value store.

1.3.3 Robot Specific Features

Because of the many types of sensors used in robotics, some kind of message standardization had to be introduced. Therefore, definitions of messages for poses and data from various kinds of sensors, like IMUs, cameras or rangefinders. There are definitions for path planning algorithms including maps and robot odometry. Due to these definitions, the messages can be utilized by every part of ROS library or tools. Another library that is used mainly by manipulators or humanoid robots is the Robot Geometry Library. It is used to monitor many parts of the robot or sensors keeping position and pose data in a single frame of reference. It can update transform data with speeds as fast as hundreds of Hertz. It allows to define static links also. The library assumes that the parts of the distributed system are updated at various times. Robot Description Language is a way to create a description of the dimensions of parts used, their locations and how they look. It is defined as a XML style document.

1.3.4 Tools

ROS contains various tools for data recording and visualization. This speeds up the software development process as the problems can be addressed better. Some of the key tools used are described below. The core tools are designed to be able to work in an CLI environment. This tools include i.e. catkin, rostopic, rosnode or roslaunch.

catkin

catkin is a CMake based open source build system implemented in ROS. It is language independent and works on multiple platforms. Its main improvement over cmake is that it can automatically find package and build multiple project depending on each other at the same time.

rosbag

This tool creates bags, which means it subscribes to on or multiple topics and records the communication happening within the Robot Operating System in to a file. This way the communication is logged and may be used in further playback, which maybe helpful in resolving bugs and issues.

rviz

rviz is a graphical tool that provides 3D visualization of the robot. It transforms all the inputs such as sensor messages into a simulation data in one common frame of reference. This data is visualized to provide a way to monitor what the robot sees and as such analyze problems encountered.

rqt

It is a framework based on Qt used in graphical interface development of a robot. Its plugins can be utilized to create a custom interface showing data useful to user about certain sensor status, its voltage or current output. For example rqt_graph creates a visualisation of the current ROS node configuration, their topics and services. That allows to more deeply comprehend the function of the system. Another such plugin would be rqt_plot, which plots any data that can be measured over time, such as voltages, currents or revolutions per minute. Or rqt_topic and rqt_publisher that are used to monitor topics or publish new messages to existing ones.

Throughout the thesis ROS Melodic Morenia is being used. It was released on May 23, 2019 and it is the latest version of Robot Operating System.

2 Problem Analysis

The objective of this thesis is to develop a mobile robotic system, that detects a person utilizing camera and artificial intelligence, calculate a path to said person and follow him/her.

The designed person following system will monitor its environment for humans. This action is based on image capture done by camera mounted in such position that it is able to see a person at a reasonable distance. The resulting image taken is sent to detector, a neural network of choice, which will determine if there is a person present. In a case when person is detected, it is marked with a probability and has a bounding box is assigned. A set of coordinates based on the bounding box will be computed, which represent coordinates of the middle of his or her feet. This coordinates will be passed through a system that is able to estimate a distance in centimetres with respect to the robot base. If multiple people have been detected, they will be sorted in sorted in ascending order based on distance a person with the shortest distance will be followed. Based on the calculated position a shortest path to the target person will be determined. The respective linear and angular velocities will be recalculated to speed for each of the stepper motors used. These newly acquired speed information will be sent to the robotic platform which will directly operate motors. Simplified pipeline of this process is illustrated in figure 2.1. The actual implementation of this process is described in chapter 5.

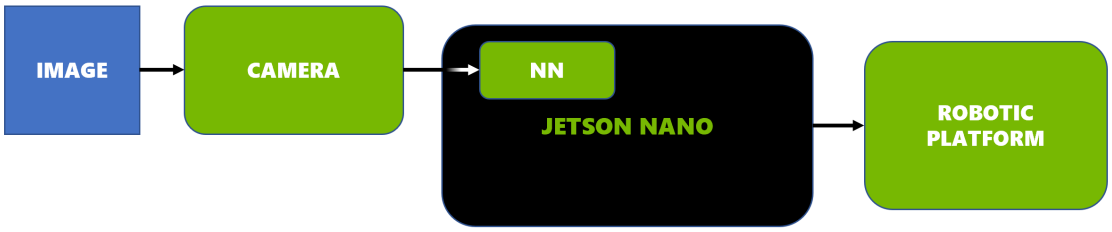


Fig. 2.1: Designed pipeline of the person follower system. Images captured via camera are subjected to detector. Based on presence of a person, its coordinates and path to it are computed and this information is fed to the robotic platform.

Person Detection

To solve this problem, multiple neural networks with various approaches will be researched. Based on the results of the research some of the networks will be further tested on available hardware. Network that will be the fastest in the tests while

maintaining some reasonable level of accuracy will be chosen. Because this is a mobile system working in real-life environment it requires the network inference time to be under at least 500 milliseconds. The maximum time was selected because the system loop should at least run at 1 Hertz.

Distance Estimation

The distance estimation from pixel coordinates will be based on a grid made by points marking certain distances. Set of approximations between these marked points will be used to determine each metric distance component in x and y axis. The camera is mounted right in the middle of the motors. Because of the way a camera is mounted a distance to the center of rotation axis is acquired. Therefore an easy motor control is achieved. If necessary a distance to the center of mass or front of the robot can be calculated. This is done via Denavit-Hartenbert matrix [10] shown in equation 2.1, which defines spatial link between reference frames.

$$\mathbf{H} = \left[\begin{array}{ccc|c} & \mathbf{R} & & \mathbf{T} \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \tag{2.1}$$

where $\mathbf{R}$ is a rotation matrix and $\mathbf{T}$ is a translation column vector. The row vector of zeros represents perspective and the last element defines scale.

Determination of Velocities

Now that the person-to-robot position is known, a shortest path to it has to be determined. The system analyzing the data will output the resulting linear and angular velocities. This information will be recomputed from the acquired vector to speed of the motors, and then converted in to a format of the message specified and later sent to the robotic platform.

3 Neural Networks

Multiple neural networks for detection of people from camera feed have been described in this thesis. These networks vary in multiple aspects, such as complexity, speed or accuracy and are described in detail in the following sections.

3.1 YOLOv3

YOLOv3 is a second improvement of the You Only Look Once detector [27], developed since 2016. The first YOLO network introduced simultaneous object detection and classification via single neural network in a single run. It formed the bounding box prediction and class probabilities as a single regression problem instead of utilizing a separate region proposal network used in [17]. This technique made it a lot faster compared to the networks available at that time. The biggest difference was that YOLO looked on the image as a whole and marked bounding boxes and their respective classes straight on that image. On the contrary, the other detectors, use a separate network like Region Proposal Network (RPN) to determine possible objects. Afterwards, a classifier outputting bounding box runs on all the possible regions of interest and classifies the detected objects into their respective classes. However, the network cannot predict more than a two bounding boxes with the same class in the same cell grid. It has problems correctly classifying groups of small object, e.g. a flock of birds. Another downside of this approach is that does not output correct prediction if the object has a different, unencoutered before aspect ratio. The first improvement to YOLO, YOLO9000 [25], is supposed to improve the recall and localization of the network while at least maintaining the precision. Therefore a features like batch normalization, higher resolution classifier, direct location prediction, dimension clustering and anchor boxes. The latest version of You Only Look Once is bit bigger than the previous versions but it is more accurate. Some of the key changes are described in the following subsections.

3.1.1 Bounding Box Prediction

Bounding box is predicted as a anchor box via dimension clusters. Each bounding box has determined four properties, x and y coordinates, width and height.

$$b_x = \sigma(t_x) + c_x \tag{3.1}$$
$$b_y = \sigma(t_y) + c_y \tag{3.2}$$
$$b_w = p_w e^{t_w} \tag{3.3}$$
$$b_h = p_h e^{t_h} \tag{3.4}$$

where c_x and c_y represent offset from the top left corner, p_h is prior height, p_w is prior width and t_x, t_y, t_h and t_w are coordinates predicted by the network. $\sigma(x)$ is the sigmoid function of x.

This article uses bounding box objectness score to determine the best possible bounding box. If the bounding box prior overlaps a ground truth better than all of the prior bounding boxes, it gets score of 1. If it is not the best, but overlaps the ground truth by more than the threshold, 0.5, the prediction is ignored. Every ground truth object has only one bounding box. If a ground truth does not have bounding box assigned, there is no loss for coordinate or class prediction. There is only loss in objectness prediction.

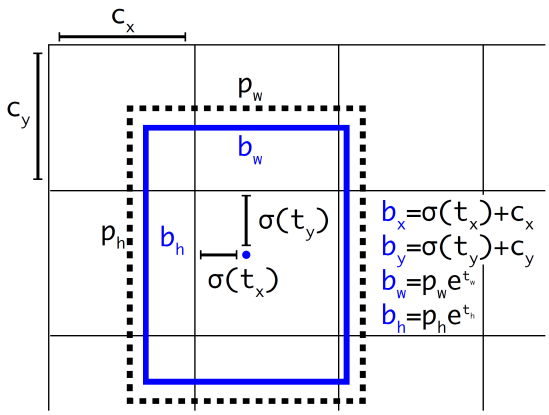


Fig. 3.1: Bounding Boxes with dimension priors and location prediction [26]

3.1.2 Class Prediction

For class prediction a multi-label classification is used. Performance-wise the use of softmax layer has been deemed unnecessary, because it assumes that one bounding box contains only one class. Independent logistic classifiers have been used instead. Utilization of a binary cross-entropy loss for class prediction during training. These settings perform better on more complex datasets containing overlapping classes. Boxes are predicted on three scales. The principle is based on upsampling the feature map of one feature extractor layers two times. Then another feature map

from earlier part of the network is taken and concatenated with the upsampled feature map. This process is repeated twice. Bounding box priors are determined using K-means algorithm.

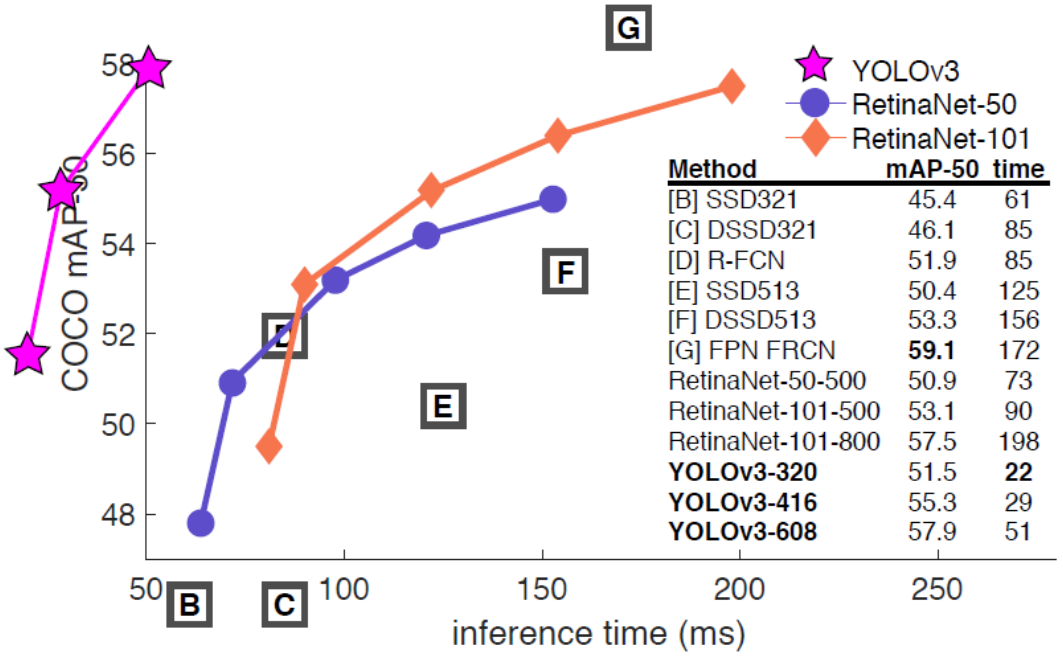
3.1.3 Network Architecture

This network contains 53 convolutional layers therefore it is named Darknet-53. It is an improvement over Darknet-19 used in YOLOv2 [25]. Darknet-53 utilizes fusion of Darknet-19 and residual networks. The full network architecture can be seen in the following table.

	Type	Kernels	Size	Stride	Output
	Convolutional	32	3x3	1	256x256
	Convolutional	64	3x3	2	128x128
	Convolutional	32	1x1	1	
1x	Convolutional	64	3x3	1	
	Residual				128x128
	Convolutional	128	3x3	2	64x64
	Convolutional	64	1x1	1	
2x	Convolutional	128	3x3	1	
	Residual				64x64
	Convolutional	256	3x3	2	32x32
	Convolutional	128	1x1	1	
8x	Convolutional	256	3x3	1	
	Residual				32x32
	Convolutional	512	3x3	2	16x16
	Convolutional	256	1x1	1	
8x	Convolutional	512	3x3	1	
	Residual				16x16
	Convolutional	1024	3x3	2	8x8
	Convolutional	512	1x1	1	
4x	Convolutional	1024	3x3	1	
	Residual				8x8
	Average Pool		Global		
	Fully Connected		1000		
	Softmax				

Tab. 3.1: Darknet-53 [26]

3.1.4 Summary



Method	mAP-50	time
[B] SSD321	45.4	61
[C] DSSD321	46.1	85
[D] R-FCN	51.9	85
[E] SSD513	50.4	125
[F] DSSD513	53.3	156
[G] FPN FRCN	**59.1**	172
RetinaNet-50-500	50.9	73
RetinaNet-101-500	53.1	90
RetinaNet-101-800	57.5	198
YOLOv3-320	51.5	**22**
YOLOv3-416	55.3	29
YOLOv3-608	57.9	51

Fig. 3.2: Speed and accuracy comparison of YOLOv3 with other state-of-the-art networks. [26]

Based on the information from the figure 3.2, YOLOv3 is a the fastest network. It also has a comparable accuracy to the more complex networks like RetinaNet or FPN FRCN. The Darknet-53 was trained and this test was done on NVIDIA Titan X GPU, which way more powerful than the hardware used in this thesis. The detector is trained for classification of 80 classes of COCO dataset, including people. This network is very complex but accurate and fast. It is very good to detect smaller object but its ability to detect medium and large objects has worsened as cited in [26].

3.2 RetinaNet

RetinaNet [22] is another one stage network like YOLOv3 [26], but it utilizes different approach. This network tries to reduce the drawback between speed and accuracy. To achieve this it introduced focal loss and created a network based on FPN which fill be described in the following sections.

3.2.1 Focal Loss

It is a loss function based on a scaled cross entropy loss with decaying scaling factor to zero with increasing confidence. A way to ensure detection in a scenario with a very big imbalance between background and foreground classes is proposed called focal loss.

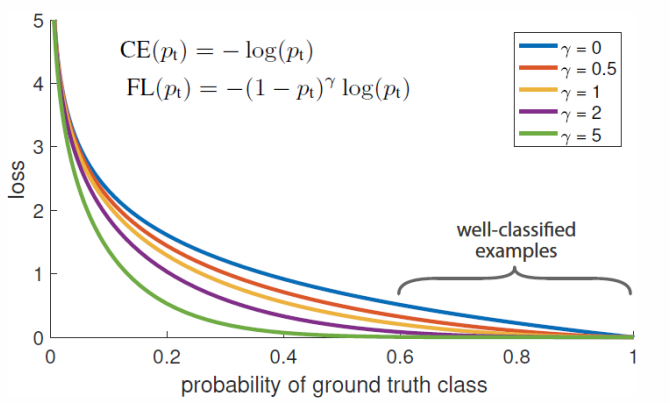


Fig. 3.3: Focal loss which adds $(1-p_t)^{\gamma}$ to standart cross entropy. Setting γ bigger than zero reduces the loss for correctly classified examples. Setting p_t bigger than 0.5 puts bigger focus on wrongly classified samples. [22]

3.2.2 Detector

The RetinaNet detector is a network unification that relies on a Feature Pyramid Network and two subnets as can be seen in figure 3.3. The FPN backbone computes convolutional maps and feeds the data to the two subnets. The box subnet is responsible for regression of bounding box. The other subnet is used for class labeling. The FPN is a convolutional neural network that propagates layers in vertical fashion and also having lateral connections. Each layer is used to predict object detection at different scales. Each pyramid level uses translation invariant anchors. New anchors are added to the standard anchor sizes resulting in 9 anchors in each layer. The classification subnet is a Fully Convolutional Network (FCN) consisting of 3x3 convolutional layer with ReLu activation, followed by another convolution and sigmoid activation. It is connected to every layer of the FPN and it does not share any parameters with the box subnet. In the bounding box regression network a small FCN is connected to every layer of the Feature Pyramid Network again. The design of this network is almost identical to the subnet for class labeling, with the only difference being the size of the output.

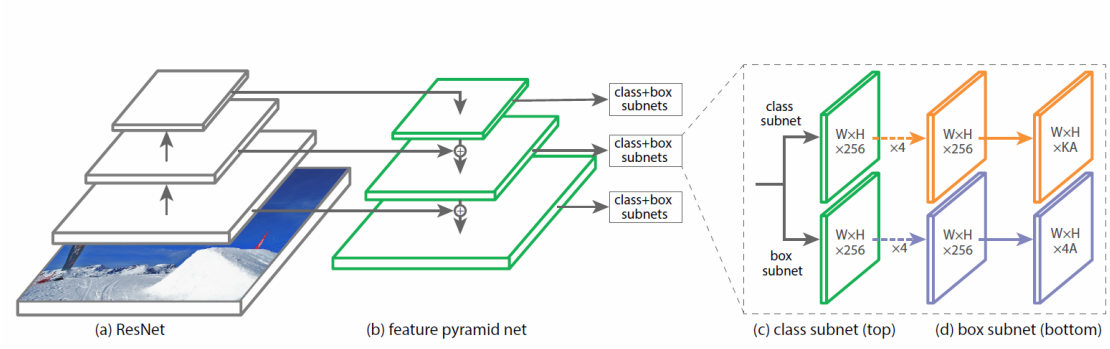


Fig. 3.4: RetinaNet archtecture. It based on Feature Pyramid Network followed by two simultaneously subnets. One for class and one bounding boxes. [22]

3.2.3 Network evaluation

RetinaNet was tested on COCO dataset and compared to other state-of-the-art detectors as can be seen in figure 3.5. Authors of the paper state that while the network performs better than their biggest competitor, DSSD [16], it is also faster.

	backbone	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Two-stage methods							
Faster R-CNN+++ [16]	ResNet-101-C4	34.9	55.7	37.4	15.6	38.7	50.9
Faster R-CNN w FPN [20]	ResNet-101-FPN	36.2	59.1	39.0	18.2	39.0	48.2
Faster R-CNN by G-RMI [17]	Inception-ResNet-v2 [34]	34.7	55.5	36.7	13.5	38.1	52.0
Faster R-CNN w TDM [32]	Inception-ResNet-v2-TDM	36.8	57.7	39.2	16.2	39.8	**52.1**
One-stage methods							
YOLOv2 [27]	DarkNet-19 [27]	21.6	44.0	19.2	5.0	22.4	35.5
SSD513 [22, 9]	ResNet-101-SSD	31.2	50.4	33.3	10.2	34.5	49.8
DSSD513 [9]	ResNet-101-DSSD	33.2	53.3	35.2	13.0	35.4	51.1
RetinaNet (ours)	ResNet-101-FPN	39.1	59.1	42.3	21.8	42.7	50.2
RetinaNet (ours)	ResNeXt-101-FPN	**40.8**	**61.1**	**44.1**	**24.1**	**44.2**	51.2

Fig. 3.5: Network performance comparison. As can be seen in the last two rows both RetinaNet networks with different backbones perform better than the rest of the networks in terms of AP. The only exception is the detection on large objects. This table is acquired from. [22]

3.2.4 Summary

Since this network was evaluated on COCO dataset, it is able to detect humans. Inference on one image took approximately 125 milliseconds, which on such complex dataset is pretty good. On the other side it used very powerful graphical processor (NVIDIA TESLA M40) compared to the one available in Jetson Nano and therefore is considered unsuitable to be used in this thesis.

3.3 OpenPose

OpenPose [2] is a network for multi-person pose estimation. It detects 135 key-points across human body, including face, hands and feet [30, 28, 3, 2]. In this manner it tries to recognize the pose of every person. Pose estimation utilizes Part Affinity Fields, which are a set of two dimensional vectors encoding location and orientation of limbs in the image as stated in [2]. In this paper was proven that better accuracy is achieved through refining part affinity fields. Body part refining did not play an important role in the process.

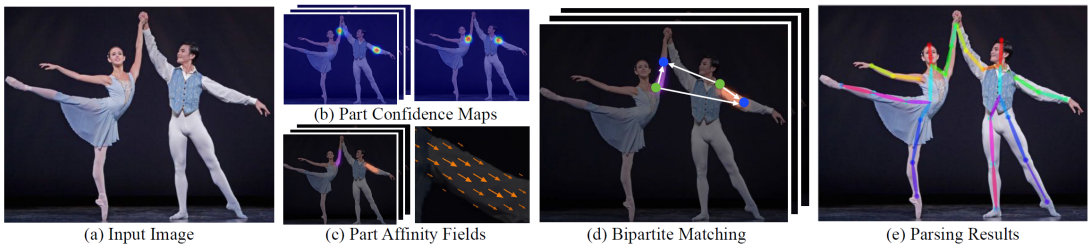


Fig. 3.6: Classification pipeline [2].

As shown in figure 3.6, system produces a two dimensional location of anatomical parts from color input image. Then a feed-forward network predicts a set of 2D confidence maps and a Part Affinity Field 2D vector set. At the end of the pipeline a greedy inference parses all the confidence maps and Part Affinity Fields and produces a two dimensional key points of all people.

3.3.1 Network Architecture

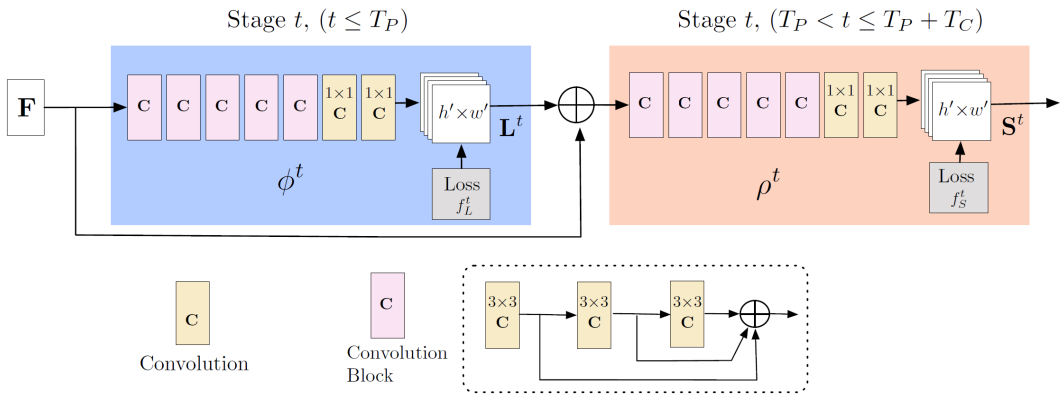


Fig. 3.7: Network architecture [2].

As illustrated on figure 3.7, the first cascade calculates Part Affinity Fields $\mathbf{L}^t$, and the second one predicts confidence maps $\mathbf{S}^t$. Convolution block consists of three convolutional layers with 3x3 kernel size and their outputs are concatenated at the end. This approach adds a nonlinear layer for each layer, thus allowing the network to work with both low and high level features.

3.3.2 Summary

OpenPose is a cross-platform neural network with its own input feeder, available for Windows, Ubuntu, MacOS and even for embedded devices such as NVIDIA Tegra due to CUDA support. It also enables selection of networks recognizing different body parts. OpenPose runs at 22 frames per second on a computer equipped with a NVIDIA 1080Ti graphics card. As seen in table 3.2, it performs very good in comparison with other state-of-the-art networks. While it is not the best in terms of mAP score or individual detection keypoints, its speed does not change with the number of people in image, as illustrated in figure 3.8.

Method	**Head**	**Sho**	**Elb**	**Wri**	**Hip**	**Knee**	**Ank**	**mAP**
DeeperCut [20]	78.4	72.5	60.2	51.0	57.2	52.0	45.0	59.5
Levinko et al. [21]	89.8	85.2	71.8	59.6	71.1	63.0	53.5	70.6
ArtTrack [19]	88.8	87.0	75.9	64.9	74.2	68.8	60.5	74.3
Fang et al. [11]	88.4	86.5	78.6	70.4	74.4	73.0	65.8	76.7
Newell et al. [24]	92.1	89.3	78.9	69.8	76.2	71.6	64.7	77.5
Fieraru et al. [12]	91.8	89.5	80.4	69.6	77.3	71.7	65.5	78.0
OpenPose [2]	91.2	87.6	77.7	66.8	75.4	68.9	61.7	75.6

Tab. 3.2: Validation results of human pose estimation networks on MPII dataset. This table is adapted from [2].

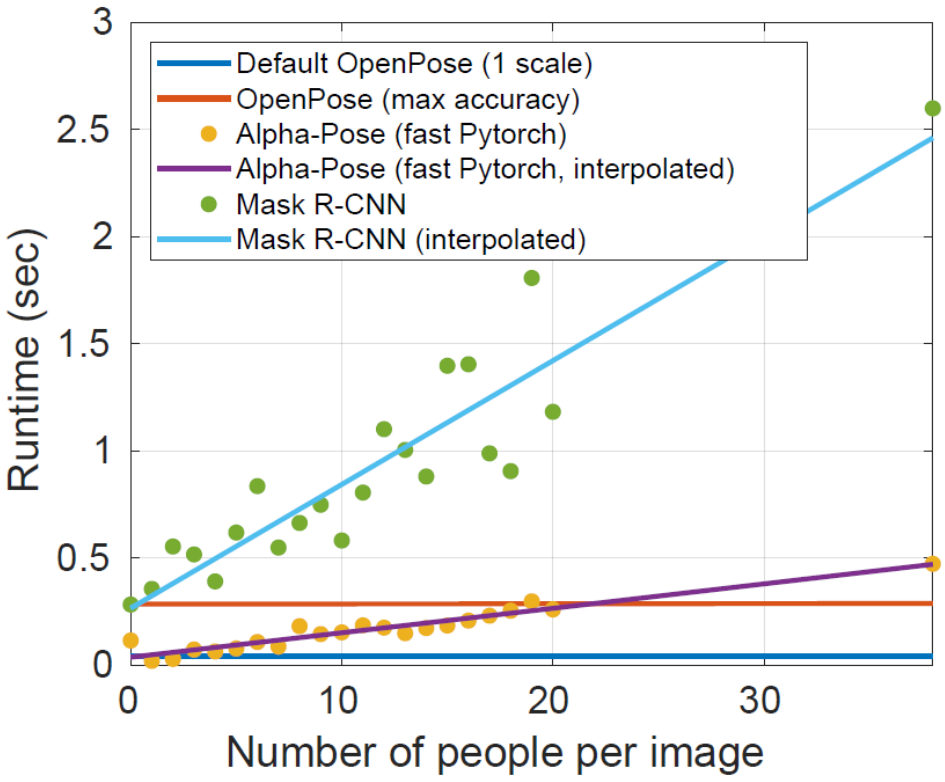


Fig. 3.8: Speed/Person count dependence [2].

OpenPose would present many benefits in implementation of this thesis, such as gesture recognition, which would allow use of more complex commands to the robot. On the other side, it requires more powerful hardware than is available, therefore real-time detection would not be possible. Due to this issue, a better solution is to use a more lightweight neural network, like PedNet.

3.4 PedNet

PedNet [29] is a convolutional neural network used for segmentation of pedestrians. It upsamples and downsamples the feature maps using the encoder-decoder principle. The detailed description of this network is located in the following subsections.

3.4.1 Network Architecture

The input of the network consists of the sequence of three consecutive images. The previous, the current and the future frame. This temporal information helps to achieve optimal segmentation of the middle frame. The encoder network analyzes features like edges or curves, while the decoder network provides higher levels of knowledge. The number of feature maps is not increasing in the decoder. It is

concatenated instead. That way it contributes to segmentation. In the figure below, a simplified diagram of PedNet architecture is illustrated.

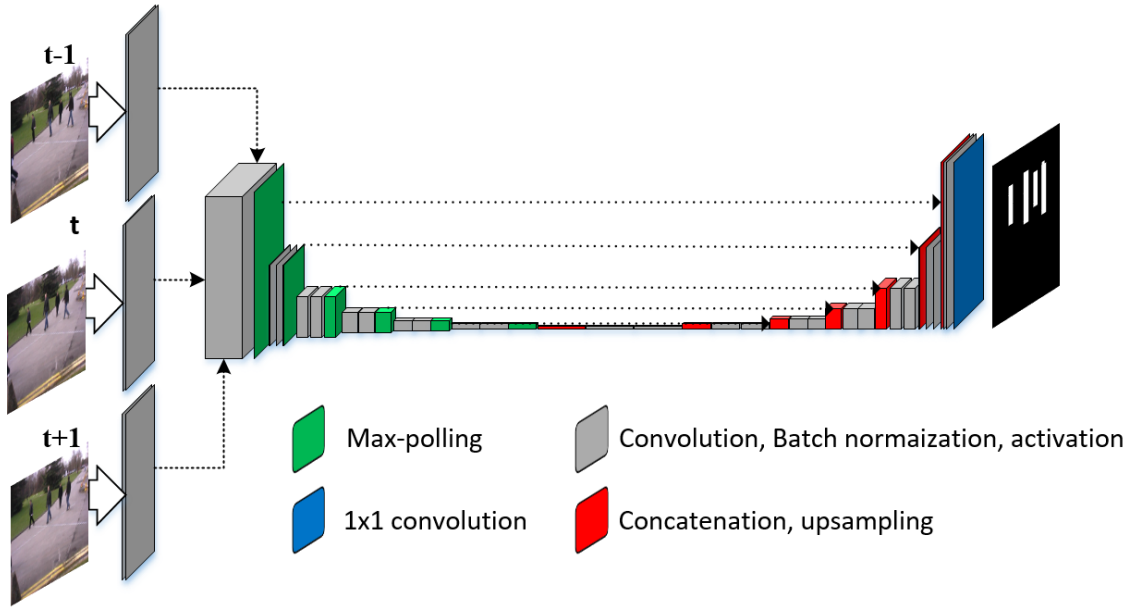


Fig. 3.9: Proposed architecture. [29]

The encoder network is made of 17 convolutional layers with kernel size of 3x3, followed by ReLU activation function and then downsampled by max pooling layer, which has 2x2 kernel and stride of 2. The decoder upsamples the feature map, concatenates it with corresponding map from the encoder. This result is twice convoluted with each layer activated by Rectified Linear Unit. The last step involves a convolution with the kernel size 1x1. This way a binary mask segmenting the pedestrians in the image is created.

3.4.2 Training

Dataset has been annotated using binary masks in such way, that white region represents person and the black is background. The batch size was 21 frames. For end-to-end network optimization a stochastic gradient descend was used. The authors of [29] state that the classical stochastic gradient descent has three problems. It is slow, it has a certain attraction to getting stuck in local minimum and it has an oscillatory behavior during convergence. Therefore, RMSprop was used to resolve this obstacle. It introduces running average of the squared gradients, which help to update the network hyper-parameters.

$$I_{dw_t} = \gamma I_{dw_{t-1}} + (1-\gamma) dw_t^2 \tag{3.5}$$

$$I_{db_t} = \gamma I_{db_{t-1}} + (1-\gamma) db_t^2 \tag{3.6}$$

$$w_{t+1} = w_t - \alpha \frac{dw_t}{\sqrt{I_{dw_t}} + \epsilon} \tag{3.7}$$

$$b_{t+1} = b_t - \alpha \frac{db_t}{\sqrt{I_{db_t}} + \epsilon} \tag{3.8}$$

where $I_d w$ and $I_d b$ are running average of squared gradients, w represents weights and b represents bias, dw and db are squared gradients with respect to weights and bias. γ is a moving average parameter, α is learning rate and ϵ is a small number added to prevent division by zero. The initial learning rate was set to 10^{-4} and γ was set to 0.9.

As mentioned before, the network input consists of three consecutive images. Image taken at a time t, image $t-1$ taken before the image t and image $t+1$ taken after it. The binary segmentation mask is calculated only for the image t. The other two are used for better segmentation as they provide temporal information. This way mask prediction is simplified, because the direction in which the pedestrian is walking is learned.

Data Augmentation

The network was trained on only 3979 samples, which were further augmented to prevent network underfitting. Approximately 50% of images were left as is. In the rest either the frame was cropped or a Gaussian noise followed by gamma corection was added. This technique is illustrated on figure 3.10.

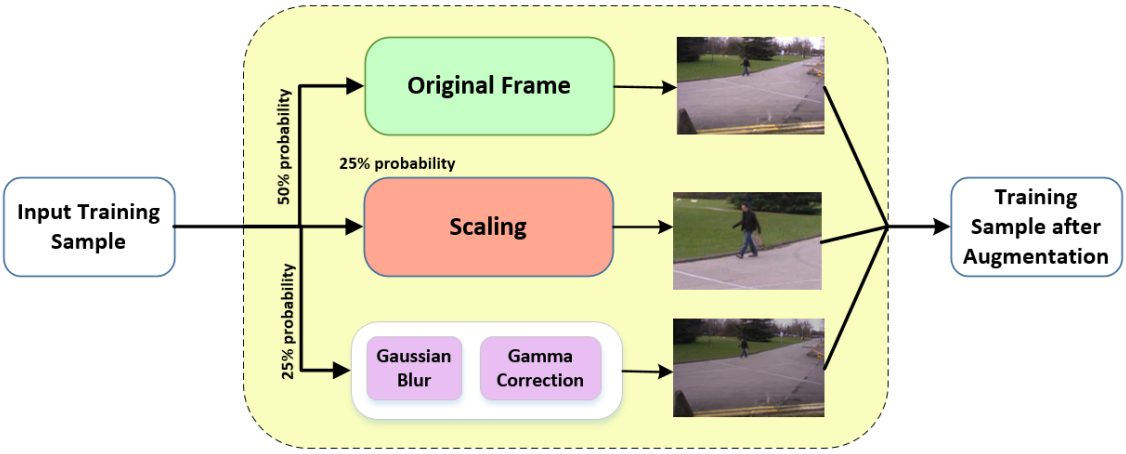


Fig. 3.10: Data augmentation technique. [29]

3.4.3 Summary

This network was validated on multiple dataset commonly used for tracking of persons, including CamVid, Pets2009 and TUD.

Dataset	Recall	Precision	F_1	F_2	mIoU	Accuracy
CamVid	0.877	0.945	0.910	0.890	0.764	0.988
Pets2009 (View 1)	0.883	0.953	0.917	0.896	0.755	0.989
TUD-crossing	0.880	0.955	0.916	0.894	0.741	0.989
AFL-footbal	0.880	0.955	0.916	0.894	0.746	0.987
Pets2009 (View 5)	0.877	0.943	0.909	0.889	0.724	0.988
Pets2009 (View 7)	0.878	0.949	0.912	0.891	0.767	0.989
TUD-Stadmitt	0.880	0.955	0.916	0.894	0.741	0.988
TUD-Campus	0.886	0.944	0.914	0.897	0.692	0.989

Tab. 3.3: Results of validation of PedNet on various datasets. This table is adapted from [29].

Based on the information provided in the table 3.3, this network is robust and performs very well on eight different datasets. It specifically trained for pedestrian segmentation, which is one of the main goals of this thesis. It is able to segment multiple persons within one frame. Because of all these features, this network was selected for testing.

3.5 Summary of the Researched Networks

This section provides a quick summary and comparison of researched networks. Most of the networks are trained and evaluated on a powerful workstation graphics card like NVIDIA GTX 1080Ti or NVIDIA TESLA M40, while PedNet used NVIDIA TITAN X. Starting with the slowest network, RetinaNet, while it performed better than YOLOv2 at the time of creation, it was bested by the YOLOv3. RetinaNet took between 73 - 198 milliseconds to compute an inference depending on the network model used on the GTX 1080Ti card. YOLOv3 had better performance than all RetinaNet variants while the inference time was drastically reduced. One inference took 22 - 51 milliseconds. The times were taken from figure 3.2. Because of the slow inference RetinaNet was discarded from further testing as such slow times were not viable to use on a mobile platform. YOLOv3 is pretty fast multi-class detector. It is recommended for further testing. OpenPose is a very interesting set of networks that can estimate human pose based on more than one hundred key-points. It ran on at 22 frames per second, which is approximately 46 milliseconds for one inference

on the same graphical card as YOLOv3. The network is also optimized for NVIDIA Tegra devices, which Jetson Nano is part of, but the authors recommended to use at least Jetson TX2 for testing. It is a little bit more complex than necessary to achieve the goal, but it opens new choices in advanced control of the robot's behavior if such goals are further pursued. The last network, PedNet did not specify the speed that it ran on, but it is the only network specifically created for person detection. Furthermore, it was part of the AI example by NVIDIA for Jetson Nano, and it also optimized to run on Tegra devices. Even though there is a lack of information, the optimization and the ready availability for the Jetson platform is definitely worth the testing.

Network	**Type**	**Tegra Optimized**	**NVIDIA GPU**	**Inference**
Darknet-53	CNN	No	GTX1080Ti	22 - 58 ms
RetinaNet	FPN CNN	No	TESLA M40	125 ms
OpenPose	CNN	Yes	GTX1080Ti	46 ms
PedNet	CNN	Yes	TITAN X	NA

Tab. 3.4: Quick comparison of the researched networks.

4 Testing of the Neural Networks

In total, three of the networks described earlier were tested. Darknet and OpenPose were being tested on a Windows 10 computer equipped with EVGA NVIDIA GeForce 1080 SC graphics card clocked at 2012 MHz, quad core Intel Core i7 6700K CPU @ 4.5 GHz and 16 GB of 3200 MHz DDR4 RAM. Pednet variants ped-100 and multiped-500 were tested on the Jetson Nano.

4.1 Darknet

Unfortunately, it was not possible to test Darknet. This was due to many problems with the compilation of the network in the Windows environment, as this network was created on a Linux machine. The main source of this problems was the build of OpenCV library on Windows PC, error checking and repair of the wrong paths and linking of these dependencies.

4.2 OpenPose

This network [2] was tested on a provided example video filmed at a plaza of city. In the video, there are many people walking in every direction in various distance from the camera. The network does not need the whole person to be visible. Just couple of the watched key-points needs to be detected. OpenPose detected maximum of 16 people in the video, mostly those closest to the camera. During the run many moving targets were correctly recognized and none false detection was recorded. It was able to run on a GPU at an average of 19 frames per second as can bee seen on figure 4.1. It provides fairly good results on the people within the detection range.

4.3 PedNet

During testing two variants of the PedNet network [29] were tested, each with video sample in two different resolutions. The first sample format was 640x480 pixels and the other was Full HD. Both of these networks were tested on NVIDIA Jetson Nano. The main difference between these variants is that they are optimized with different target group inn mind. Network ped-100 should only detect people, presumably alone or in a small group. Multiped should detect bigger groups of people and any luggage they carry.

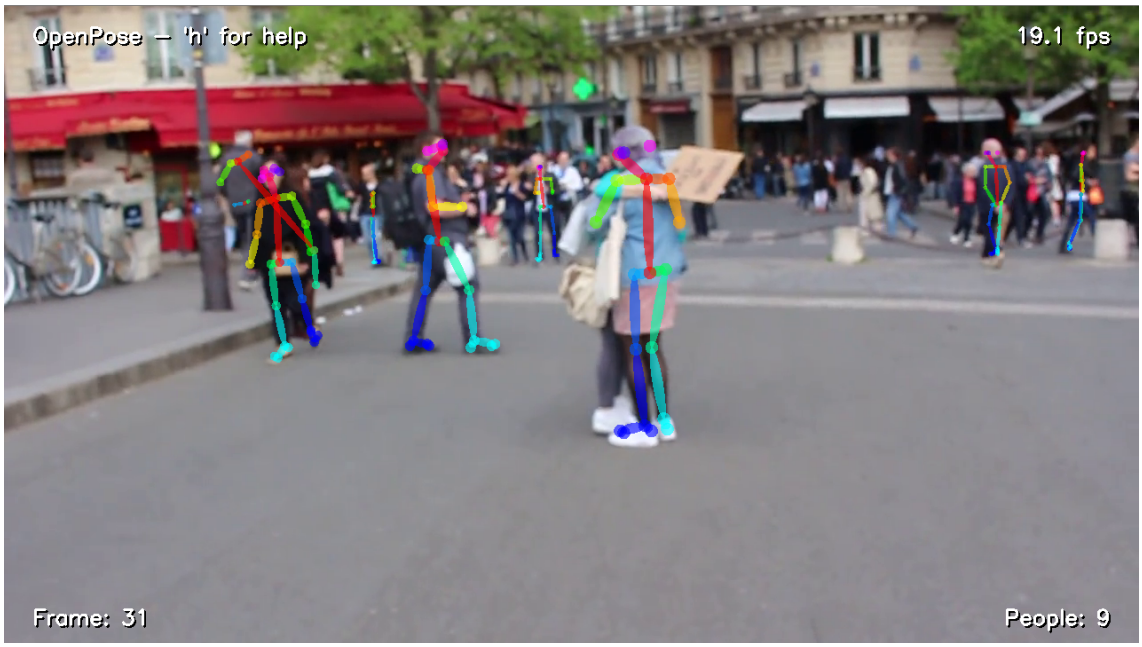


Fig. 4.1: Image from the example video of OpenPose [2] testing. Nine people were detected at the time. All of the people had their detected body key-points marked and joined.

4.3.1 ped-100

The network was tested in home conditions, it was able to recognize multiple people at the same time, although it generates false positives from distant objects which are mistaken for a person. It also confuses hanged clothing for a person and, in some cases, is overconfident of the detection. In both image resolutions the network performs fairly similar. Both run at 8 to 9 frames per second. The only difference is, that CPU computation time rises by 2 ms when Full HD is used, from 120 ms to 122ms. CUDA takes 120 ms and is same for both resolutions. As can be seen in the figures 4.2 and 4.3, the lower resolution distorts the shape of a person, which in turn results in lower detection percentage in pednet example.

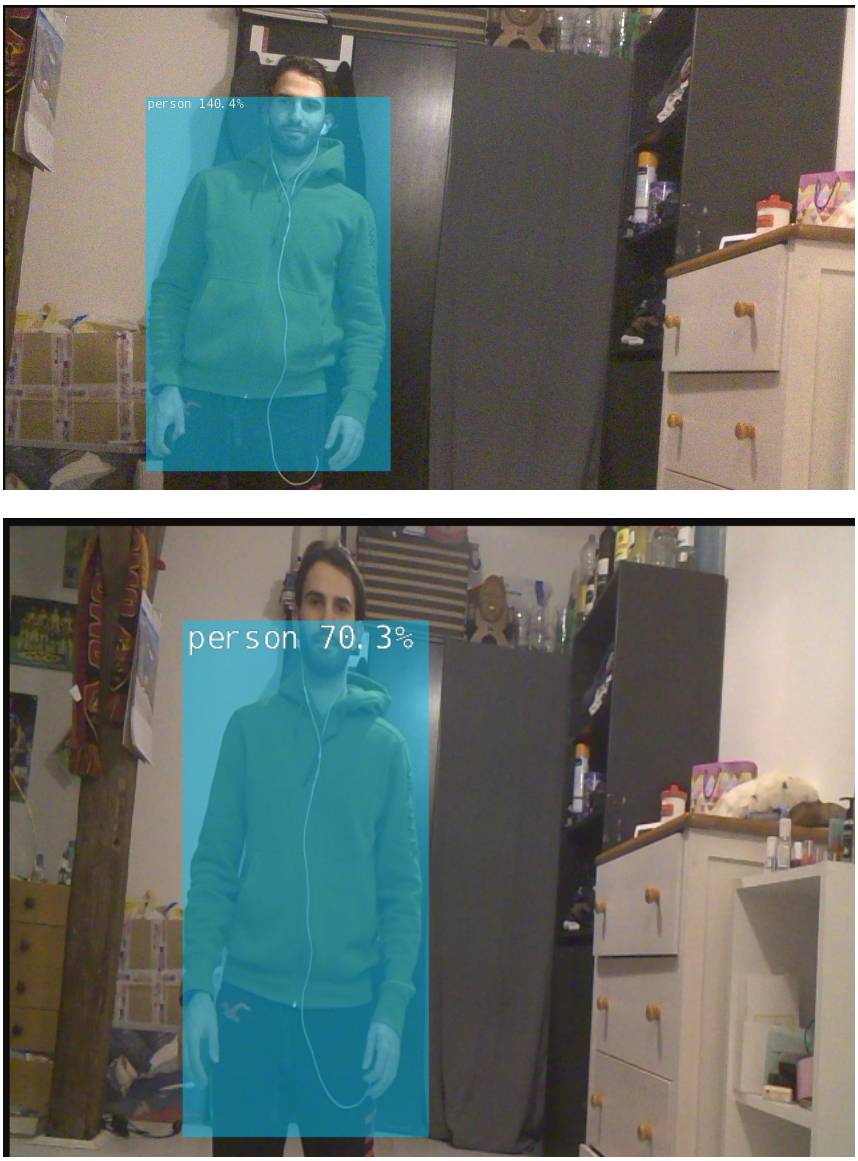


Fig. 4.2: ped-100 testing. Left image is in Full HD resolution. Right is 640 by 480 pixels.

4.3.2 multiped-500

The network was evaluated in the same conditions as ped-100. It generates way more false positives and fails miserably when in use in Full HD. In multiple occasions it did not detect person in a image. Both test ran at 9 frames per second, with CUDA time of 120 ms. The lower resolution network CPU time was 123 ms while the Full HD network had CPU time of 153 ms.

Fig. 4.3: multiped-500 testing. Left image is in Full HD resolution. Right is 640 by 480 pixels.

4.4 Summary of the Tested Networks

Three neural network detectors were chosen to be tested.

The first of them was the network from YOLOv3 [26] paper. Despite the inability to test the DARKNET-53, due to library and compilation problems, it was evaluated based on the theoretical knowledge acquired from the paper. As stated there, the fastest variant had inference time of 22 miliseconds on NVIDIA GeForce 1080Ti GPU. This graphics card has 3584 CUDA cores compared to the 128 of the Jetson Nano. Therefore a big performance drop was expected if implemented which could be well under 5 frames per second. Furthermore, the classification of such vast

number of classes is unnecessary for this thesis. Based on the theoretical knowledge, this network is suitable for fulfilling the goal of this thesis, but it is not optimal.

The second network was OpenPose [2]. This network is able to detect multiple people within same frame and is one step farther compared to the other networks tested. It can estimate the pose of the person. This feature would be very beneficial in advanced control of the robot, for example, by gestures. However, to fulfill the goal of this thesis, it is not required. The network was again tested on a graphical processor that is way more powerful than the one in Jetson Nano and it ran at approximately 20 frames per second. Even though it is optimized for NVIDIA Tegra devices of which Nano is part of, it is reccomened to run on at least Jetson TX2. While adding interesting features, performance-wise this network seemed to be very slow on the hardware used in the thesis. It was marked as not suitable for implementation.

The final network tested was PedNet with its two variants, ped-100 and multiped-500, both tested in two different resolutions. As the Full HD resolution did not distort people in the image and performed better overall, it was to go for resolution. The first tested variant was ped-100, which is more suitable for a single person detection or detection of a smaller group of people. It had inference time of 120 milliseconds when directly tested on the Jetson Nano. It provided fairly good recognition, even though it generated false positives on very distant objects and sometimes got a bit overconfident (140% confidence of a person class in figure 4.2). On the other hand, multiped-500 is designed more in focus to bigger groups of people and their luggage. During the tests, it had little confidence in the detected class and generated many bounding boxes of wrong size. Its inference time was only three milliseconds bigger compared to ped-100, but its CPU time was almost 153 milliseconds. Overall it did not perform better than ped-100 and its possible implementation was discarded.

Based on these results PedNet was selected, concretely its variant ped-100, as it provides detection of multiple people, is lightweight, fast, and despite some of its flaws, accurate.

5 Hardware Settings and Software Solution

This chapter describes the various settings and structure of the software, its functions and approach to problems, helping to achieve the goal of tracking the person.

5.1 Hardware Settings

For the final design, Apeman A80 action camera was used, because it was able to perceive a whole human of average height at a minimum distance of 2.5 meters, when mounted on a pole 56 centimetres above the ground. The Raspberry camera module could capture up to the human's waist only at 4.5 meters from the robot, when mounted at the same height. The comparison of the cameras is shown in figure 5.1.

Fig. 5.1: Comparison of shots from both cameras. The target is 4.5 meters away from the robot. The left shot is from Raspberry Pi V2 camera module, the right from Apeman A80 action camera.

The pole upon which the camera is mounted on the top plate of the robotic platform, right in the middle of the two stepper motors. This means it is placed in the rotation axis of the robot. It is connected to the Jetson Nano via USB cable. Camera has been calibrated via OpenCV using a chessboard pattern on 34 images and algorithms from [31, 1]. The resulting camera matrix and distance vector are.

$$CameraMatrix = \begin{bmatrix} 1540 & 0 & 776 \\ 0 & 1558 & 540 \\ 0 & 0 & 1 \end{bmatrix} \tag{5.1}$$

$$DistortionVector = \begin{bmatrix} -0.3003 & -0.0439 & -0.0119 & 0.0054 & 0.2171 \end{bmatrix} \tag{5.2}$$

Based on these parameters, every frame captured is undistorted before any further convertions happen.

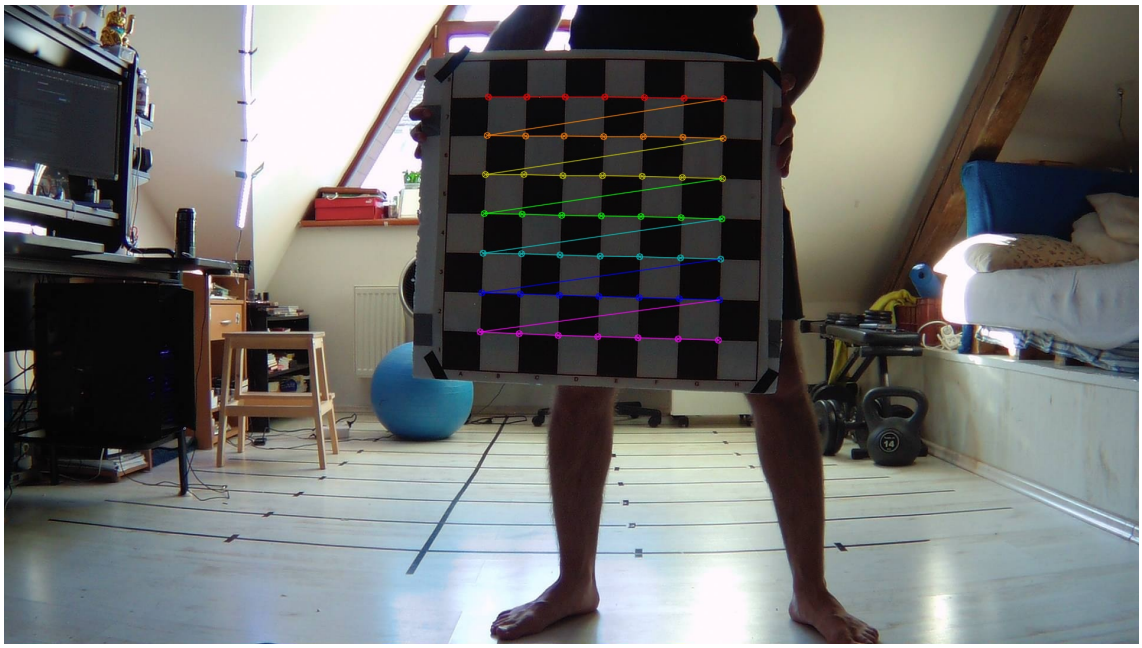


Fig. 5.2: Camera calibration example image.

The NVIDIA computer is mounted approximately 5 centimetres in front of the camera pole. It connects to the Raspberry Pi via ethernet cable and is supplied power from the DC-DC converter as it requires different voltage than the rest of the electronics mounted on the platform.

For distance calibration a set of lines was drawn on the floor creating a makeshift grid. Black tape was selected as it is very easily seen on the lightly colored flooring. This makes line detection easier as there is a big color difference resulting in sharp edges.

Fig. 5.3: Design of the robot.

5.2 Software Structure

The software is divided into five separate programs written in C++. Camera, Neural Network, Odometry, Route Planning Unit (RPU) and Motor Controller Unit (MCU). Each program, representing a ROS node, communicates with specified programs via ROS topics. MCU and Odometry nodes utilize UDP packets to communicate with the robotic platform, behaving like a UDP server. Communication diagram can be found in figure 5.4. Every program loop runs at 5 Hz.

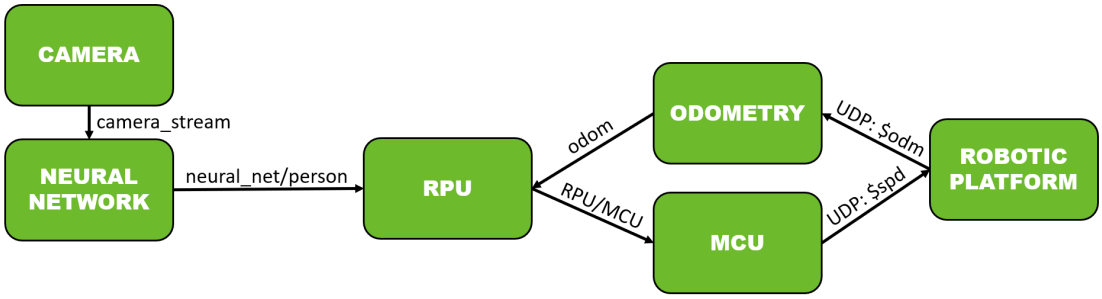


Fig. 5.4: Software structure. Arrows represent means and direction of communication.

5.3 Programs

Purpose of this section is to briefly describe the logic behind each program and its solution to the facing problems.

5.3.1 Camera

This program is responsible for camera stream creation and its succeeding posting to topic. It can create either stream for CSI camera using GStreamer or a stream for USB camera utilizing V4L2 driver. Currently, the program uses V4L2 for camera communication but GStreamer option is configured as well. The camera stream is configured to use 1920 x 1080p (FHD) resolution at 30 frames per second. Some settings are inherited from the camera itself, e.g. ISO and view angle. If the program fails to find camera it sends FATAL ERROR message. In the next step, a frame is captured. If the is empty, ERROR message is sent. Otherwise, every frame is undistorted using camera matrix and distortion vector acquired from camera calibration. This now undistorted frame is converted into RGBA format and later published to the camera_stream topic.

5.3.2 Neural Network

Pednet (ped-100) neural network is loaded in this program, if it is unable to load, a ROS FATAL ERROR is sent. Loading of the network and all the subsequent operations with it are done by the code provided by NVIDIA. TensorRT optimizer is run on the first ever loading of the network to provide better performance and lower memory usage. If the network was loaded previously, instead of running the optimizer, already optimized model is loaded. The program subscribes to camera_stream topic afterwards. Frames acquired from the topic are classified by the

network and each detection with higher than a 50% probability has its bounding box recorded. Every person is represented as a point in two dimensional space. Its coordinates are calculated as follows.

$$X = (Right_{BB} - Left_{BB}) \,/\, 2 \tag{5.3}$$

$$Y = Bottom_{BB} \tag{5.4}$$

where $Right_{BB}$ and $Left_{BB}$ represent x coordinates of the right and left side of the bounding box respectively and $Bottom_{BB}$ is the y coordinate of the bottom side of the bounding box.

Such representation is implemented because of the need to calculate distance between the robot and person. It makes more sense to measure distance to the feet of the person than to its visual centre of mass, because the later option would assume that the person is further away than it actually is. Every 2D point representing the person is then added to std::vector object, based on which number of messages, corresponding to the number of detected people, regarding every person's position are published via neural_network topic.

5.3.3 Route Planning Unit (RPU)

RPU is directly responsible for calculation of the distance of the person in metres. Such calculation based on a monocular camera only is possible because of known distance of certain pixels. This was done using a grid with 50 points in reference distance. Based on these points 15 line equations were approximated as can be seen in figure 5.5. Ten in Y direction, the first starting at 180 centimetres and the others 30 cm apart, ending at 450 centimetres from the robot. There are 5 lines representing the X distance, evenly spaced by 50 centimetres. The intersection of these lines should lie in one point, the focus. There is however a small deviation in the figure 5.5, which is due to small errors in marking the exact pixel representing the distance. The coefficients of these equations are stored in a data file in a slope-intercept form (eq. 5.5) and are loaded at the start of the program as two std::vector objects yLines and xLines.

$$y = mx + b \tag{5.5}$$

Route planning unit then subscribes to two ROS topics. neural_net/person and odom. Based on the pixel position information acquired from neural network real distance from the person is calculated. Maximum number of detected people is determined by the ROS topic queue size, which is set to five. This process consist of multiple steps.

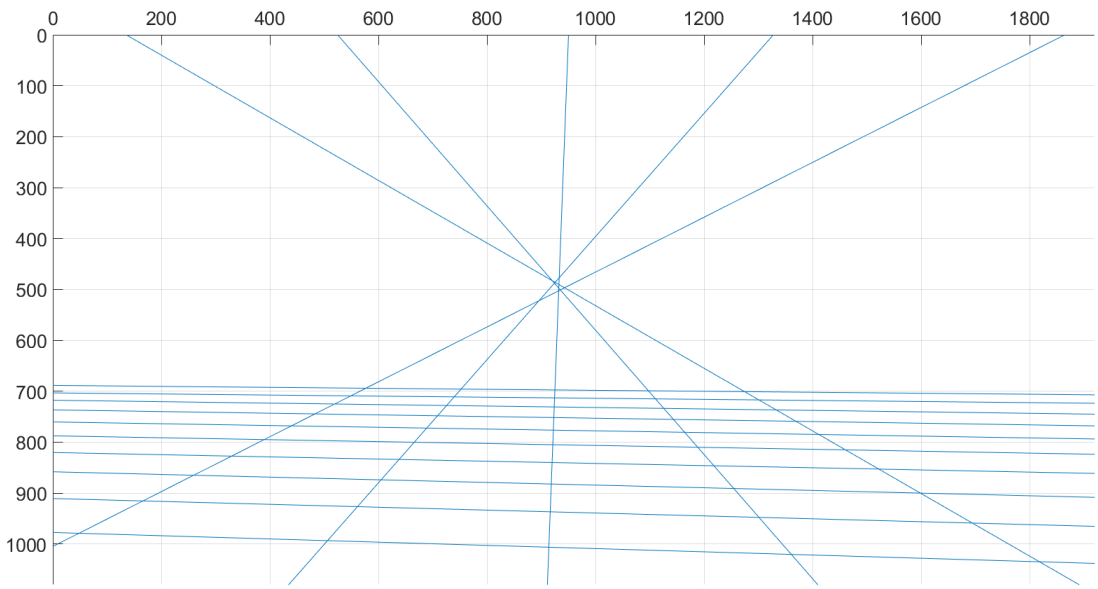


Fig. 5.5: Grid of lines representing actual, real-life distance from the robot's point of view. The five lines, lets call them vertical, mark distance in X axis from the center of the robot, which is represented by the third vertical line. This line is therefore marked as a 0 cm line. The vertical lines to the left of center line are represented as a negative distance from this line. Each line is spaced out by 50 centimetres. Therefore the leftmost vertical line represents the distance of -1 m from the center line. The right side is defined in the same way but in a positive distance from the center line, resulting in the the rightmost vertical line to mark the +1 m distance. The horizontal lines represent how far back an object is from the robot. The closest, the one on the bottom, marks the 1.8 meter distance, while the topmost represents the distance of 4.5 m. The lines are evenly spaced by 30 centimetres. Perhaps a better explanation can be found in figure 5.6.

Step 1: Find closest lines

Iterate through the lines marking X distance substituting into the line equation. The sign of the result shows the relative position of the point to the line. Minus sign suggests that the point is to the left and plus that it lies to the right. If there is a sign change between two iterations, the point lies in between these lines. The magnitude of the result represents the distance from the line, thus the closest line is selected. In similar manner Y line is selected. The only difference is that the minus sign represents direction to the top and plus to the bottom.

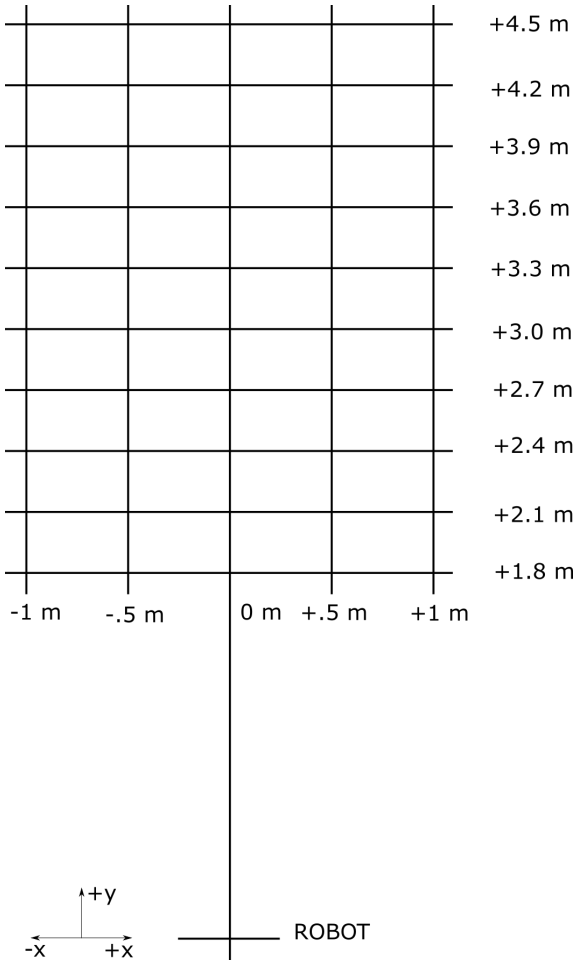


Fig. 5.6: Grid of lines representing actual, real-life distance from the top view.

Step 2: Calculate X distance

The result depends on both lines calculated in the previous step. There are two possible scenarios.

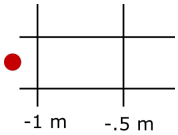
Scenario A: Point lies on the line

In this scenario, the distance of the line from the robot is known and is returned right away.

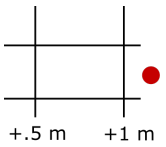
Scenario B: Point does not lie on the line

Then, the line is translated to pass trough the point and the nearest intersect with Y line is computed. The nearest intersect is used in target distance calculation. Based on the point's distance and optionally, the line position from it, there are four possible scenarios. In each scenario lower and upper intersect has to be calculated and one line has to be marked as closer to line representing 0 centimetres in X distance.

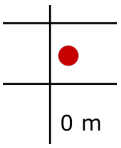
If the distance is smaller than the -100 cm line, the lower intersect is determined by the -100 cm line and the closest Y line. The upper intersect is then calculated using the -50 cm line and the same Y line. The -100 cm line is then labeled as the line closer to zero.



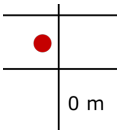
The second option is that the person is behind the +100 cm line. Then the lower intersect is calculated with respect to the +50 cm line and the corresponding Y line. The upper intersect is determined by the +100 cm line and the Y line. The line closer to zero is the +100 centimetre line.



If the person is to the right of the nearest line, the lower intersect between the X line and the closest Y line is calculated. The upper intersect is then computed between the next X line to the right and the Y line. The line closer to zero is determined based on the position to the 0 cm line. If it lies to the left, one line to the right is marked, otherwise the current X line is tagged.



The final option deals with the detection lies on the left side of the nearest line. Again the lower and upper intersects are calculated between the Y line and the line left of the current X line (lower) or the current line (upper). The line closer to zero is determined similarly to the previous statement, with the exception of checking if the line lies to the left instead of right, and if it does, then marking the line to left of the current X line.



Based on the learnt information, a distance in pixels between the intersects is computed. This distance represents 50 centimetres in real world. This allows to compute the distance in centimetres one pixel represents, and thus, the distance from the closest line is known. The total distance in X direction is calculated as follows:

$$X_{cm} = \begin{cases} DL \cdot -sgn(x), & \text{if } k = 2 \\ (k-2) \cdot SW + DL \cdot sgn(k-2), & \text{otherwise} \end{cases} \tag{5.6}$$

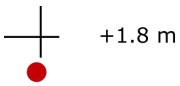
where DL is the distance from the closest line, x is the X coordinate of the person's position, k represents the index of the line (0 - the leftmost, 4 - the rightmost) and SW is the sector width (50 cm).

Step 3: Calculate Y distance

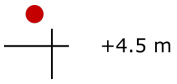
The approach to determine the Y distance in centimeters is in many steps similar to the previous step. The calculation heavily relies on the closest Y line computed in the step 1. **Scenario A: Point lies on the line**

The line's distance is returned. **Scenario B: Point does not lie on the line** Intersect with the 0 cm X line is computed, which is used in the later stage of this step. Again lower and upper intersect are calculated as well as a line closer to the robot is marked. There are the four same possible scenarios. All intersect calculation use the 0 centimetre X line as one parameter, written as the X line in the following paragraphs.

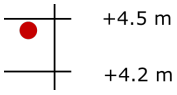
If the person is closer than the first line. The lower intersect is calculated between the first Y line and the X line. For the upper intersect the second Y line is utilized. The first Y line is labeled as the lower line.



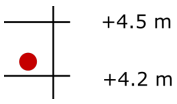
If the person is further than the last line. The upper intersect is determined via the current line and the lower uses one line closer to the robot. The last line, 4.5 metre away from the robot, is tagged as the lower line.



If the person stands under the closest line. The upper intersect is computed with the closest Y line and the X line. The lower utilizes one line closer to the robot. The lower line is the same as line used in the computation of lower intersect.



If the person stands behind the closest line. The lower intersect uses the current Y line. The upper is determined based on the next line farther away from robot. The current line is marked as the lower.



The pixel distance between the lower and the upper intersect, representing 30 centimetres allows to calculate the proportion between the real and pixel distance. Then the distance from the closest Y line is determined. The total distance in Y distance is computed in the following manner:

$$Y_{cm} = \begin{cases} BD - DL, & \text{if } k = 0 \\ BD + k \cdot SH + DL, & \text{otherwise} \end{cases} \tag{5.7}$$

where DL is the distance from the closest line, BD represents the base distance from the robot to the first line, k represents the index of the line (0 - closest to the robot, 9 - furthest from the robot) and SH is the sector height (30 cm).

Step 4: Find the closest person

This step calculates metric distance for each detected person based on the x and y coordinates acquired in steps 2 and 3. In an iterative process an euclidean distance for every person is computed. Based on this distance, coordinates of the closest person are recorded.

Step 5: Go to the closest person

When the position of a person with the respect to the robot is known, a way to navigate to him/her is determined. There is a condition which has to be checked first. If a distance between the robot and a person is less then the base distance of 1.8 meter, the robot stands still. This is due to the minimal distance the robot can determine, which is approximately 1.7 metres. Otherwise a control devised from differential equations of two wheel differential drive robot model.

$$\dot{x} = v \cdot \cos\phi \tag{5.8}$$

$$\dot{y} = v \cdot \sin\phi \tag{5.9}$$

$$\dot{\phi} = \omega \tag{5.10}$$

where v is the velocity of the robot in m/s, ϕ is the robot's current rotation and ω represents angular speed in rad/s.

Therefore, to control the robot, velocity v and angular speed ω need to be calculated every iteration. The magnitude of velocity is set to be directly proportional to euclidean distance, but to saturate at the 80 percent of the motor's maximum speed. This maximum speed is computed via the formula 5.11. The reserved 20% serve to accommodate the potential need to turn the robot. If there was not one, a failure to reach the goal could occur.

$$speed_{max} = 2\pi \cdot R \cdot RPS \tag{5.11}$$

where R is the wheel radius and RPS are the maximum revolutions per second of the stepper motor.

After substituting the values in to the equation, this formula is acquired. $speed_{max} = 2\pi \cdot 0.025\ m \cdot 4\ rps$. The maximum speed for the actual configuration of the robot is 0.63 meters per second.

To determine the angular speed, the desired angle ϕ_D needs to be computed. The equation used is shown in 5.12.

$$\phi_D = \arctan\frac{y}{x} \tag{5.12}$$

where x and y are the coordinates of person obtained in the previous section.

Then, an error between the robot's angle of rotation and the desired angle is calculated. Furthermore, there is a need to keep the angle constrained between the $-\pi$ and $+\pi$ boundaries. These steps are done using formulas 5.13 and 5.14.

$$e = \phi_D - \phi \tag{5.13}$$

$$e^* = \arctan2 \frac{\sin e}{\cos e} \tag{5.14}$$

where e is the angle of rotation error, ϕ_D is the desired angle of rotation, ϕ is the actual angle of rotation of the robot, and e^* is the constrained error value.

P controller is used to output the value of ω with the proportional gain K_P of 1.

5.3.4 Motor Control Unit (MCU)

As seen in the figure 5.4, subscribes to the RPU/MCU topic. Its purpose is to determine the velocity values of each motor based on the info acquired from the Route Planning Unit and then send them as a UDP packet via ethernet connection to the Raspberry Pi 4B serving as a control unit of the robotic platform. It receives a ROS geometry_msgs::Twist message from the beforementioned topic containing the desired linear and angular speed to reach the goal. There are three constants required to compute the motor's speed. Maximum speed the motor is able to provide, denoted as V_MAX, and then the WHEEL_RADIUS and TRACKWIDTH of the robot. As previously stated V_MAX equals 0.63 metres per second, WHEEL_RADIUS is 0.025 metre and TRACKWIDTH is 0.22 metres.

The following equations show how speed of the left and right motor is calculated.

$$v_{left} = \frac{2v - \omega \cdot TW}{2 \cdot WR} \tag{5.15}$$

$$v_{right} = \frac{2v + \omega \cdot TW}{2 \cdot WR} \tag{5.16}$$

where v is the linear speed, ω is the angular speed, TW is the TRACKWIDTH and WR is the WHEEL_RADIUS.

At this time, speed for both motors is determined in metres per second, but the robotic platform needs to input speed within the -1000 to +1000 interval. Therefore the speed needs to be mapped to these new boundaries. This is done via the equation 5.17.

$$s^* = 1000 \cdot \frac{s + V\_MAX}{V\_MAX} - 1000 \tag{5.17}$$

where s is the speed acquired from the either of the previous equations and s^* is the new calculated speed.

Afterwards a message to the platform is constructed. It is based on the following format. $spd,<left_speed>,<right_speed>. As a next step, the message is passed as a payload to the UDP packet sent to the platform.

5.3.5 Odometry

This program acquires data from the robotic platform and publishes the to the odometry ROS topic. It proceeds to accomplish this task in following manner.

The behavior of this process is supposed to mirror the one of the UDP client. A request is sent to the platform via the $odm, (comma included) message. Then it awaits the answer. After the reception the answer is checked whether it conforms to the specified $odm,<left_pulses>,<right_pulses> format. If it does not, it will be rejected. The actual distance travelled for each wheel is computed via the following formula.

$$distance = 2\pi \cdot R \cdot pulses \tag{5.18}$$

where R is the radius of the wheel and $pulses$ represents number of encoder pulses since the last message.

Afterwards, a single path that the robot travelled is calculated as a average of the sum of distance travelled by each wheel.

$$distance_{center} = \frac{left + right}{2} \tag{5.19}$$

where $left$ represents the distance travelled by the left wheel and $right$ the distance travelled by the right wheel.

The new position of the robot is then determined using the last known location and angle of rotation.

$$x = x_{prev} + D_C \cdot \cos\phi_{prev} \tag{5.20}$$

$$y = y_{prev} + D_C \cdot \sin\phi_{prev} \tag{5.21}$$

$$\phi = \phi_{prev} + \frac{D_R - D_L}{TW} \tag{5.22}$$

where x, y and ϕ represent the pose of the robot, subscript $prev$ marks the previously known position, D_C is the distance travelled by the robot, and D_R and D_L is the distance travelled by right and left wheel respectively.

These new values are published as a ROS nav_msgs::Odometry message. The last known coordinates and angle of rotation are then updated to correspond with the new values.

Furthermore, the Route Planning Unit and Odometry programs have had a delay of 8 seconds introduced to match the startup of the neural network. Then a bash script was created to change the environment for possible run of the launch file. This script is registered as a cron job to start at reboot, wait two minutes and then execute the launch file. The two minute delay is to provide enough time to manually switch camera to operational mode.

6 Field Testing of the Robot

After the final assembly, the test stage was ready to commence. It was done in multiple stages explained below.

Simulation

At the time of writing the code a simple simulation of sending of data was done to test if the process ran smoothly. Nodes including RPU, MCU and odometry were successful when given pixel coordinates of a person. It was basic goal to goal navigation where the position and pose of the robot was estimated via odometry and the target did not move. The planning unit was successfully able to determine position of the person with maximum error 2 centimetre in each axis. It was also able to choose correct values for speed and angular velocity to reach the desired goal. Via motor controller unit, the acquired values were successfully recalculated to match the required speeds for each wheel. Based on these correctly computed values, the robot was able to reach the goal, stopping 100 centimetres away from the target. The simulation proved the design to be plausible.

Problems encountered at runtime

This part of the test proved to be difficult as some strenuous errors have been encountered. The Jetson Nano is very difficult to work with as it does not support many programs or libraries that native Ubuntu does because of the arm64 processor architecture. Due to this and many other bugs, it had to have its libraries, e.g. ROS, reinstalled or the whole operating system had to be flashed and install all the libraries and dependencies all over again. It proved to be minor obstacle compared to others. The biggest one was when out of nowhere multiple *CUDA unspecified launch failure* occurred. It did not specify where this error happened or what was its origin. Research on the internet to at least find the area in which it happened proved to be unsuccessful as the range of the problem could be from out of memory (OOM) error under synchronization, scaling up to the worst possible scenario, chip damage.

At first, ROS image transport package was thought to be the source of the problem as it introduced collisions with OpenCV libraries due to different version usage (v3.2 vs. v4.1.1). This was solved via merging camera and neural_network programs/nodes to one, omitting the camera topic. However, the cuda problem still did persist, but as this solution solved the library conflicts, it was kept as well.

The next step in the problem analysis was to narrow down the origin of the error. Therefore, NVIDIA example camera detection demo was used, as it worked

for the Raspberry Pi camera module during the tests of neural networks. Run of this example has shown that it was unable to get image frame from the Apeman action camera due to problem GStreamer had with V4L2 driver. Based on this, Raspberry camera was hooked up to the system again. When used with the 15 centimetre flex cable it worked flawlessly, but when connected via the long, one metre cable, the example suddenly started crashing with error statement, *cuda unspecified launch failure*. The results were again inconclusive, as usage of the camera in conjunction with the short cable was not very suitable, because it would require changing the whole layout of the robot just to accommodate the Jetson Nano somewhere along the camera mounting pole, thus changing weight distribution and introducing more severe vibrations to the camera. Furthermore, it was verified that my program was able to capture camera frame and save it to disk without any corruption to the image. Learning from this piece of information, there had to be error in image format conversion as the frame was captured in RGB and network required RGBA. As specified by the *detectNet::Detect* function the input image was supposed to be a float pointer. Because of this, the camera part contained following code to convert it to the required format.

Listing 6.1: Image conversion from cv::Mat to float*

```
float* temp_img = new float[4 * img.total()];
cv::Mat img_RGBA(img.size(), CV_8UC4, temp_img, 4);
cv::cvtColor(img, img_RGBA, cv::COLOR_BGR2RGBA);
```

This code indeed did produce float* image but for unbeknownst reason, the neural network would still throw the unspecified launch failure error. As the documentation to the library was not great and it only specified that the format should be float pointer, as it was, a more thorough digging into the code was needed to solve this problem.

It was discovered that the neural network uses the RGBA image represented as float pointer which behaves as cuda float4 double pointer data type. This proved to be a daunting task to solve. No possible conversion to this format was found and no new one was discovered as conversion from cv::Mat which contains pixels, to float pointer which somehow contains the whole image could not be done. Instead, after long reading through the code, an image loading function was found. Its declaration looked like this.

Listing 6.2: loadImageRGBA declaration [15]

```
bool loadImageRGBA( const char* filename,
                    float4** cpu, float4** gpu,
                    int* width, int* height,
                    const float4& mean )
```

The cpu and gpu parts represent a space where the image is allocated in CPU and GPU memory. Parameter mean is optional.

Based on it, it would be possible to load a previously saved image in one from the standard formats, i.e. JPEG. However, there still was not any conversion from float4 double pointer to plain float pointer. Typecasting the variable as a float pointer did not help. After further digging in the code, a way to achieve this was found. It is shown in the following code.

Listing 6.3: Loading image as float pointer [15]

```
const char* filename = "<path_to_image>/img.jpg";
float* img_cpu = NULL;
float* img_cuda = NULL;
int width = 0;
int height = 0;

loadImageRGBA( filename, (float4**) img_cpu,
               (float4**) img_cuda,
               &width, &height);
```

If parameters width and height are set to 0, they are automatically set based on the size of the loaded image. Otherwise the image would be resized to match the desired dimensions.

Quick test on the neural network has shown that this approach solved the *cuda unspecified launch failure*, as they did not appear during the runtime anymore. However, it created another problem. Saving the image to the disk is slow. The SD card upon which the OS is installed has write speed of 24 megabytes per second, which is 2 times slower than any ordinary hard drive and cannot be even compared to state-of-the-art NVMe drives that operate with speeds higher than 3000 MB/s. It would have introduced a relatively big delay between the data acquisition and the actual physical response from the robot. Some kind of speed up had to be devised.

The best solution came in using a ramdisk. It could bridge the necessary writing and reading of the image, while keeping the high speed. Basically it is a disk partition, but it is created on a portion of RAM memory. It should achieve higher speeds both at reading and writing that the aforementioned NVMe drives.

It was created in the following manner.

Listing 6.4: Ram disk creating procedure

```
mkdir /mnt/ramdisk
mount -t tmpfs -o size=10m tmpfs /mnt/ramdisk
```

To automate the process on boot up an entry had to be added to */etc/fstab*.

Listing 6.5: fstab entry to mount ramdisk on boot up

```
tmpfs /mnt/ramdisk tmpfs nodev,nosuid,noexec,nodiratime,
size=10M   0 0
```

The only downside to this approach was that if the RAM became full, the partition would be pushed to swap partition, which would in turn, result is way slower speeds, because swap is located on a SD card. Therefore, the whole code execution would be delayed and physical response of the robot would become slower, but because of the tradeoff that had to be introduced due to format conversion, it cannot be helped. A small delay of 800 microseconds was introduced to make sure that there was enough time to finish image writing before being read.

Testing

On the test run the robot performed as expected. The boot process and start of the program is slower. The Apeman A80 action camera needs to be switched to act as a PC camera to allow image capture. Afterwards, when the neural network is loaded in to the memory, the whole process starts. There are four cases that have been tested.

The first, where the test subject, me, is standing at a distance closer or equal to 180 centimetres. Upon detecting the person, the robot was supposed to stay put, which it did. The first case was considered a success.

The second one consisted of straight path path to target. Again, the robot was able to detect the person and close the distance to 180 centimeters before stopping. Another test passed.

The third test was constructed to test the ability of robot to reach a person that is not straight ahead but is seen at an angle. This test also determined if the angular speed controller supplied correct values. It did not succeed on the first run,

because the neural network detected a poster of a football player on the wall that was closer than the desired target. Based on the programmed behavior, the robot did everything right, but the test failed. On the second run, the poster was removed. Now, the robot correctly computed the path to target and was able to reach it.

The final test consisted of robot trying to follow a moving target. This was tested in two different locations. The first one was in controlled in environment. During the test, the robot behaved as expected a was able to follow a moving person. However, the second test conducted outside did not go as planned. Because of the rough structure of the pavement, the robot lost its way as the camera cable connector is prone to get loose and fall out, killing the software execution. Therefore, to run this particular robot in a real world application, a smooth terrain needs to ensured. When the test was executed in such environment, it was able to complete the test successfully.

Summary of the Tests

At first a simulation of the control loop was done to ensure a correct computation of distance the person was located, with reference to the robot, and if the output values of linear and angular speeds corresponded with the location of the person. During the first robot runtime a fatal error was discovered that impeded person recognition process. The error happened because of the different image representation of the OpenCV and CUDA libraries. No possible conversion between this formats has been found. It was solved by saving the image first, and then loading it again via the CUDA library. To keep the process at a reasonable speed a ramdisk was introduced. The downside of such use is when the RAM gets full, as this partition is then moved to swap. Afterwards, a series of four tests was completed. Each test has been completed with positive results. Even though during some tests minor errors were detected, they have been quickly resolved. There is a need for a relatively flat and smooth operating environment to ensure correct operation of the robot.

Overall, these tests concluded that the robot is able to successfully detect a person, estimated the distance between him/her and the robot and the follow it. Therefore, the robot with its configuration as is, is able to perform in a real-world application if all environmental restrictions are met.

7 Conclusion

The goal of the master's thesis was to create a system that would detect a person via neural networks and follow it. It was structured into six chapters. Chapter 1 describes the software and hardware platform upon which, the robot will be built. It consists of explanation and specifications of parts that were utilized including NVIDIA Jetson Nano Developer Kit, cameras and robotic platform supplied by the Robotics and AI group of Department of Control and Instrumentation. In the following section the operating system and application programming interfaces were introduced. The end of the chapter was dedicated to Robot Operating System (ROS), its functions, packages and various tools.
Chapter 2 analyzes the problems faced and offers brief insight on how the solution should look.
Chapter 3 contains details about four researched neural networks for detection of persons based on an image input. The first network is DarkNet-53 explained in the YOLOv3 paper. It is a single stage multi-class detector tested on the COCO dataset. It is fast compared to other state-of-the-art networks while preserving accuracy, but is complex for the desired use and available performance. The second network researched is RetinaNet, another single stage network, which offers great accuracy but it is considerably slower. The third one is OpenPose, a network that is able to estimate position of human limbs and face expressions. It is optimized for mobile CUDA devices based on NVIDIA Tegra, but with higher computational power than the hardware available. Therefore, a real-time usage would not be possible. The third and the last network is PedNet. It is a pretty fast and lightweight network for detection of multiple people, and it has been optimized for CUDA devices. The end of the chapter is dedicated to summary of the researched networks, their positives and negatives.
Chapter 4 is a report on network testing. Three of the researched networks have been selected for testing. DarkNet-53 was unable to be launched on a Windows machine. OpenPose performed well and certainly would introduce more ways to control the robot via gestures or poses, required more computational power than the Jetson Nano computer was able to deliver and was not selected for potential implementation. Two variants of PedNet were tested directly on Jetson Nano. In terms of performance, both networks were similar, but the more advanced, multiped-500, required more CPU time during detection. Ped-100 network performed considerably better as multiped-500. Their comparison can be seen in figures 4.2 and 4.3. Pednet-100 generated fewer false positives and provided more reliable detection. Based on the results of these tests, pednet-100 was also selected for final implementation.
Chapter 5 describes in detail the configuration of the robot and actual settings used.

Furthermore, a very thorough explanation of the five programs responsible for the behavior of robot. The first program is responsible for camera image acquisition which will be passed to the selected neural network. The second program utilizes the neural network and outputs the pixel coordinates of feet of the detected person-/people. It is followed by the Route Planning Unit, which task is to determine the real metric distance from the acquired pixel coordinates. This estimation is based on a grid which is shown in figure 5.5 from the robot's point of view, and in figure 5.6 from the top view. The distance calculated in metres has an absolute error of two centimetres in each axis. This program is also responsible for calculation of the magnitude of linear and angular velocities in order to reach the target. This problem was formulated as a goal to goal behavior and no objects blocking the path were introduced throughout the whole thesis. The task of the last two programs, odometry and motor controller unit, was to communicate with the robotic platform via UDP packets and acquire the necessary info. MCU is also responsible for calculation of speed for each wheel.
Chapter 6 is the last chapter, and as whole dedicated to the testing of the robot as whole and describing the errors encountered and their solutions.
Robot was able to successfully reach its desired target in most of the tries and stopping within the specified distance during the testing. One of the biggest problems faced was the image conversion from OpenCV to CUDA format. It was solved by introducing an intermediary save and load operation. While this is not the optimal solution, it suffices, but it would be desired to remove this step altogether and perform a direct conversion. Another problem has risen from the chosen type of camera, as it has to be manually switched to the desired operating mode and that the connector to the computer is wobbly, resulting in unwanted disconnections.
Performance-wise the best improvement would be to use a more powerful member of the Jetson family, which would be able to run the network faster or utilize a network with better accuracy.
Overall, the goal of this master's thesis was reached. The designed robot is able to successfully recognize and follow the target. The main benefit of this thesis that it created a small mobile robot with the potential use in perimeter guarding applications if improved, and it researched on neural networks like OpenPose which could introduce advanced control of the robot via human interaction.

Bibliography

[1] Jean-Yves Bouguet. “Camera calibration toolbox for matlab”. In: 2001.

[2] Zhe Cao et al. “OpenPose: realtime multi-person 2D pose estimation using Part Affinity Fields”. In: *arXiv preprint arXiv:1812.08008*. 2018.

[3] Zhe Cao et al. “Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields”. In: *CVPR*. 2017.

[4] NVIDIA Corporation. *Jetson Nano Developer Kit*. URL: https://developer.download.nvidia.com/assets/embedded/secure/jetson/Nano/docs/NV_Jetson_Nano_Developer_Kit_User_Guide.pdf?h756G_0bEgBinJ9pOHGbTX0SnjK4YfCQHDhgDGdknDST8xHH-yeymCP0g29zOkp1UEHSWL3-jy8HiIRBQkoujtb3LB2ocYFJA8pW8xGpYdDgaMRovTtzZxSivMeTIrHlmxYOdb0gZ1RmYpocW0jbMSl8O81m1rzt27DTHsvGlzcCWPGASt7dkKmwdsf2umQNrp0 (visited on 05/18/2020).

[5] NVIDIA Corporation. *NVIDIA JetPack*. URL: https://developer.nvidia.com/embedded/jetpack (visited on 12/14/2019).

[6] NVIDIA Corporation. *NVIDIA Jetson Nano System-on-Module*. URL: https://developer.download.nvidia.com/assets/embedded/secure/jetson/Nano/docs/JetsonNano_DataSheet_DS09366001v0.8.pdf?rnrbqN4Of8FfqQJ4Zx8L7DjBS6uJZ3jDeauwA7qPR1uSU-WtDf8QdWt0i2k_W8WJuatdaVY0Y_VdWCXEiiaLHv84SBJ_0yUIhp88cLwChojb-HWRmhR5jYHax4YCDCyAjJ40NYtCpBwjNV5J28shUKB8bGAEIbIeOGncPTP4izfMZXbi3bqF7-wRDDUgvw (visited on 12/14/2019).

[7] NVIDIA Corporation. *NVIDIA L4T*. URL: https://developer.nvidia.com/embedded/linux-tegra (visited on 12/14/2019).

[8] NVIDIA Corporation. *NVIDIA TensorRT*. URL: https://developer.nvidia.com/tensorrt (visited on 11/28/2019).

[9] NVIDIA Corporation. *NVIDIA TensorRT Developer's Guide*. URL: https://docs.nvidia.com/deeplearning/sdk/pdf/TensorRT-Developer-Guide.pdf (visited on 11/28/2019).

[10] J. Denavit and R.S. Hartenberg. *A Kinematic Notation for Lower-pair Mechanisms Based on Matrices*. ASME, 1955. URL: https://books.google.cz/books?id=nCDOoQEACAAJ.

[11] Hao-Shu Fang et al. *RMPE: Regional Multi-person Pose Estimation*. 2016. arXiv: 1612.00137 [cs.CV].

[12] Mihai Fieraru et al. *Learning to Refine Human Pose Estimation*. 2018. arXiv: 1804.07909 [cs.CV].

[13] Raspberry Pi Foundation. *Camera Module*. URL: https://www.raspberrypi.org/documentation/hardware/camera/ (visited on 11/14/2019).

[14] Raspberry Pi Foundation. *Raspberry Pi 4 Model B*. URL: https://static.raspberrypi.org/files/product-briefs/200206+Raspberry+Pi+4+1GB+2GB+4GB+Product+Brief+PRINT.pdf (visited on 05/20/2020).

[15] Dustin Franklin. *Jetson Utils*. https://github.com/dusty-nv/jetson-utils. 2020.

[16] Cheng-Yang Fu et al. *DSSD : Deconvolutional Single Shot Detector*. 2017. arXiv: 1701.06659 [cs.CV].

[17] Ross Girshick et al. *Rich feature hierarchies for accurate object detection and semantic segmentation*. 2013. arXiv: 1311.2524 [cs.CV].

[18] Ph.D. Ing. František Burian. *Platforma KAMBot*. URL: https://sites.google.com/a/vutbr.cz/bprp/prednasky/2018/02 (visited on 05/11/2020).

[19] Eldar Insafutdinov et al. *ArtTrack: Articulated Multi-person Tracking in the Wild*. 2016. arXiv: 1612.01465 [cs.CV].

[20] Eldar Insafutdinov et al. *DeeperCut: A Deeper, Stronger, and Faster Multi-Person Pose Estimation Model*. 2016. arXiv: 1605.03170 [cs.CV].

[21] Evgeny Levinkov et al. *Joint Graph Decomposition and Node Labeling: Problem, Algorithms, Applications*. 2016. arXiv: 1611.04399 [cs.CV].

[22] Tsung-Yi Lin et al. "Focal Loss for Dense Object Detection". In: *CoRR* abs/1708.02002 (2017). arXiv: 1708.02002. URL: http://arxiv.org/abs/1708.02002.

[23] Apeman International Co. Ltd. *Apeman A80*. URL: https://www.apemans.com/product/product&path=20_21&product_id=68 (visited on 05/11/2020).

[24] Alejandro Newell, Zhiao Huang, and Jia Deng. *Associative Embedding: End-to-End Learning for Joint Detection and Grouping*. 2016. arXiv: 1611.05424 [cs.CV].

[25] Joseph Redmon and Ali Farhadi. *YOLO9000: Better, Faster, Stronger*. 2016. arXiv: 1612.08242 [cs.CV].

[26] Joseph Redmon and Ali Farhadi. "YOLOv3: An Incremental Improvement". In: *arXiv* (2018).

[27] Joseph Redmon et al. "You Only Look Once: Unified, Real-Time Object Detection". In: *arXiv* (2015).

[28] Tomas Simon et al. “Hand Keypoint Detection in Single Images using Multiview Bootstrapping”. In: *CVPR*. 2017.

[29] Mohib Ullah, Ahmed Mohammed, and Faouzi Alaya Cheikh. “PedNet: A Spatio-Temporal Deep Convolutional Neural Network for Pedestrian Segmentation”. In: *Journal of Imaging* 4 (Sept. 2018), p. 107. DOI: 10.3390/jimaging4090107.

[30] Shih-En Wei et al. “Convolutional pose machines”. In: *CVPR*. 2016.

[31] Z. Zhang. “A flexible new technique for camera calibration”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22.11 (2000), pp. 1330–1334.

List of symbols, physical constants and abbreviations

AI	Artificial Intelligence
API	Application Programming Interface
CLI	Command-Line Interface
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
FCN	Fully Convolutional Network
FOV	Field of View
FPN	Feature Pyramid Network
FPS	Frames per Second
GPU	Graphical Processing Unit
GFLOPS	Giga Floating Operations per Second
IDE	Integrated Development Environment
MCU	Motor Controller Unit
ML	Machine Learning
NN	Neural Network
OS	Operating System
RAM	Random Access Memory
ReLU	Rectified Linear Unit
ROS	Robot Operating System
RPU	Route Planning Unit
SC	superclocked – providing higher level of overclocking by factory
SLAM	Simultaneous Localization and Mapping
SoC	System-on-Chip
SoM	System-on-Module
UDP	User Datagram Protocol
XML	eXtensible Markup Language

List of appendices

A Source Code

The source code of this project is available on Azure DevOps repository. It is dependent on OpenCV library, CUDA toolbox, ROS, jetson-inference and jetson-utils repositories. The source code is accessible by this link:
`https://dev.azure.com/zak2no/_git/sentry`

B Contents of the DVD

- `/` — root folder of the DVD
  - `code` — Folder containing code and dependencies
    - `src` — Folder with C++ source code
      - `camNN.cpp`
      - `motor_controller.cpp`
      - `odometry.cpp`
      - `planning_unit.cpp`
    - `include` — Folder with headers
      - `def.h` — Header with constants and data types
    - `data` — Folder with data files
      - `config.txt` — Camera calibration data
      - `lines.txt` — Line equations
    - `launch` — Folder with launch file
      - `sentry.launch` — ROS launch file
    - `CMakeLists.txt`
    - `package.xml`
  - `Zakarovsky,RobotTrackingPerson.pdf` — Masters's thesis