BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INTELLIGENT SYSTEMS
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

TRACKING PEOPLE IN THE VIDEO CAPTURED FROM A DRONE

SLEDOVÁNÍ OSOB V ZÁZNAMU Z DRONU

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR **Bc. JAKUB LUKÁČ**
AUTOR PRÁCE

SUPERVISOR **Ing. TOMÁŠ GOLDMANN**
VEDOUCÍ PRÁCE

BRNO 2020

Brno University of Technology
Faculty of Information Technology

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

Master's Thesis Specification

Student: **Lukáč Jakub, Bc.**
Programme: Information Technology Field of study: Intelligent Systems
Title: **Tracking People in Video Captured from a Drone**
Category: Artificial Intelligence
Assignment:

1. Get acquainted with different methods of tracking people in video. More importantly, focus on people tracking in aerial video captured by a drone camera. Ascertain which methods can be used to estimate the distance between a drone and persons based on telemetry data and camera parameters.
2. Study available algorithms to detect and track people in video.
3. Design and implement a client and a server application in Python programming language. The client application that is intended for the drone, should be capable of preprocessing video and send it to the server side. Recognition of people in the video frame and conversion of their positions to local coordinates will be ensured by the server application. Use TCP / IP to communicate between the server and the client application. Perform a visualization of trajectories based on obtained coordinates.
4. Perform experiments focused on determining the accuracy of obtained trajectories.

Recommended literature:

- CIPOLLA, Roberto, Sebastiano BATTIATO a Giovanni Maria FARINELLA. Computer vision: detection, recognition and reconstruction. Berlin: Springer, 2010, 350 s. : il., fot. ISBN 978-3-642-12847-9.
- Růžička M., Mašek P. (2014) Real Time Object Tracking Based on Computer Vision. In: Březina T., Jabloński R. (eds) Mechatronics 2013. Springer, Cham

Requirements for the semestral defence:

- Items 1 and 2.

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

Supervisor: **Goldmann Tomáš, Ing.**
Head of Department: Hanáček Petr, doc. Dr. Ing.
Beginning of work: November 1, 2019
Submission deadline: June 3, 2020
Approval date: April 15, 2020

Abstract

This thesis deals with the problem of determining the location of a person and its approximation. The location is derived from video which is captured using a drone. The goal here is to propose and test existing solutions, and state-of-the-art algorithms for each encountered subproblem. This means overcoming challenges such as object detection, re-identification of persons in time, estimating object distance from camera and processing data from various sensors. Then, I am using the methods to design the final solution which can operate in nearly real-time. Implementation is based on the use of Intel NCS accelerator unit with the cooperation of small computer Raspberry Pi. Therefore, the setup may be easily mounted directly to a drone. The resulting application can generate tracking metadata for detected individuals in the recording. Afterwards, the positions are visualised as paths for better end-user presentation.

Abstrakt

Práca rieši možnosť zaznamenávať pozíciu osôb v zázname z kamery drona a určovať ich polohu. Absolútna pozícia sledovanej osoby je odvodená vzhľadom k pozícii kamery, teda vzhľadom k umiestneniu drona vybaveného príslušnými senzormi. Zistené dáta sú po ich spracovaní vykreslené ako príslušné cesty. Práca si ďalej dáva za cieľ využiť dostupné riešenia čiastkových problémov: detekcia osôb v obraze, identifikácie jednotlivých osôb v čase, určenie vzdialenosti objektu od kamery, spracovanie potrebných senzorových dát. Následne využiť preskúmané metódy a navrhnúť riešenie, ktoré bude v reálnom čase pracovať na uvedenom probléme. Implementačná časť spočíva vo využití akcelerátoru Intel NCS v spojení s Raspberry Pi priamo ako súčasť drona. Výsledný systém je schopný generovať výstup o polohe osôb v zábere kamery a príslušne ho prezentovať.

Keywords

image processing, image recognition, object detection, tracking, drone, distance estimation, Intel Movidius, Raspberry Pi, YOLO

Kľúčové slová

spracovanie obrazu, rozpoznávanie, rozpoznávanie objektov, sledovanie, odhad vzdialenosti, dron, Intel Movidius, INCS, Raspberry Pi, YOLO

Reference

LUKÁČ, Jakub. *Tracking People in the Video Captured from a Drone.* Brno, 2020. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Tomáš Goldmann

Rozšírený abstrakt

Sledovanie polohy ľudí pomocou kamerového záznamu bolo a stále je zložitým problémom. Prebiehajúci výskum však v pravidelne poskytuje nové metódy, ktoré zlepšujú presnosť detekcie objektov a zároveň znižujú potrebný výpočtový výkon na ich fungovanie. Dnes sa pochopenie snímaného obrazu počítačom stáva vďaka pokroku takou bežnou súčasťou, ako fungovanie počítačov samotných.

Ľudia dokážu pomerne ľahko rozpoznať bežné objekty okolo seba najmä na základe vizuálnych podnetov. Dva hlavné problémy, ktorým čelí detekcia objektov sú identifikácia tých častí obrazu, kde sa objekt nachádza a druhým je jeho následná klasifikácia. Tá z pohľadu dneška už nie je takým vážnym problémom. Aby bolo možné predmety alebo ľudí v zázname detekovať a následne pozorovať ich pohyb je nevyhnutné vedieť, že daný obraz obsahuje osoby. Dôležité je ale tiež poznať ich polohu a v ideálnom prípade vidieť celú siluetu, ktorá poskytuje informácie k ďalším fázam spracovania.

K tomu aby bolo možné zistiť polohu osoby sú potrebné údaje nielen z kamery, ale aj z ďalších senzorov. Na ich spracovanie sa využívajú metódy strojového učenia, ako napríklad sofistikované detektory objektov, modely klasifikátorov, detekcia geometrických primitív a ďalšie. Pod povrchom týchto algoritmov sa najčastejšie skrývajú spôsoby založené na neurónových sieťach, respektíve ich variante s využitím operácie konvolúcie. Dnes môžeme tvrdiť, že súčasné architektúry sietí sú takmer porovnateľne presné ako ľudský mozog, pokiaľ ide o špecifické rutinne činnosti akou je aj detekcia a klasifikácia objektov.

Následne potrebné vstupy zahŕňajú použitie informácií o polohe kamery a jej vlastnostiach, ako napríklad konkrétne natočene a uhol snímania. Všetky tieto premenné parametre udávajú lepšiu predstavu toho, kam kamera smeruje pri vyhotovovaní záznamu. Jej orientácia je pochopiteľne nevyhnutná kvôli jej obmedzenému zornému poľu kamery. Poloha musí byť založená na použití správnych meraní údajov opisujúcich stav zariadenia, spolu so súčinnosťou už spomínanej konkrétnej pozície objektu vo videu. Ideálny výber by mal zahŕňať sledovanú osobu ako celok od nôh až po hlavu, čo umožní dosahovanie najlepších výsledkov.

Väčšina metód zaoberajúcich sa meraním vzdialenosti pochádza z oblastí automobilového priemyslu, či experimentálnej robotiky, kde sa určuje rozostup medzi zariadením a predmetom pomocou rôznych senzorov. Z hľadiska tejto práce sú zásadné iba vizuálne prístupy, ktoré opomínajú technológie založené na ultrazvukových alebo laserových meraniach. Vo všeobecnosti môžeme tvrdiť, že kamera je tu využitá ako senzor na meranie vzdialenosti, čo prináša značné nepresnosti a náročnosť spracovania. Na druhej strane je ale kamera ľahko dostupný a cenovo nenáročný senzor. Prípadne prezentované metódy sa testujú a skúmajú ako vhodné riešenia asistentov vodičov, údaje získané z jednej kamery sa používajú na odhad vzdialenosti medzi dvoma účastníkmi cestnej premávky. Tieto znalosti teda môžu byť zovšeobecnené na určenie vzdialenosti objektu od kamery.

Dostupné metódy odhadu vzdialenosti sú odvodené z princípov elementárnej optiky a zákonitosti výslednej projekcie. Objektív kamery ukazuje zaznamenaný objekt v rovine senzora v pomere, ktorý vytvára vzťah medzi veľkosťou objektu a jeho vzdialenosťou od objektívu. Pri použití jedinej kamery bez dodatočných senzorov je skúmaným spôsobom merania vzdialenosti objektu práve odhad vzdialenosti podľa známej veľkosti objektu samotného a odhade založenom na pozícii objektu v obraze. Každá z týchto metód má voči tej druhej niekoľko výhod, pričom každá z nich má špecifické obmedzenia pre samotný objekt alebo jeho okolie. V prípade odhadu založeného na veľkosti je nutné vedieť rozmery snímaného objektu, zatiaľ čo pri druhej metóde je dôraz kladený na nájdenie bodu dotyku objektu a rovnej plochy, na ktorej sa objekt pohybuje.

Druhou dôležitou časťou sledovania osôb sú teda spôsoby ich nájdenia a *zaškatuľkovania*. Inšpirácia pre dnešné algoritmy vychádzala z pôvodnej práce, ktorá navrhla kombináciu odhadu potencionálnych miest obsahujúcich objekt s konvolučnou neurónovou sieťou na ich klasifikáciu, známou ako R-CNN, čo je v preklade *Regionálna konvolučná sieť*. Všetky ostatné hlavné architektúry detektorov vychádzali z tejto myšlienky a postupne zlepšovali svoju presnosť a výkon. Evolúcia jednotlivých detektorov viedla k veľmi populárnej a efektívnej architektúre detektorov objektov YOLO. Prístup, ktorý zaviedla táto rada riešení, je výrazne iný ako systémy založené na klasifikátoroch. Pokúša sa spracovať obraz ako celok a jeho predpovede sú odvodené z globálneho kontextu v samotnom obraze. Skratka YOLO v preklade znamená *pozrieť sa iba raz*. Jeho cieľom je byť univerzálnou odpoveďou pre akýkoľvek systém, ktorý vyžaduje detekciu objektu. Táto architektúra je pri svojej vysokej rýchlosti stále dostatočne presná a umožňuje operácie v takmer reálnom čase. Týmto spôsobom dosahuje výsledky porovnateľné s oveľa zložitejšími modelmi.

Riešenie práce definuje dve hlavné časti respektíve dve aplikácie, kde jedna je určená pre zostavu, ktorá môže byť umiestnená priamo na malé bezpilotné lietadlo. Takáto aplikačná časť beží na kompaktnom kartovom počítači Raspberry Pi štvrtej generácie, doplneným o výpočtovú jednotku Intel Movidius, v tomto konkrétnom prípade Intel INCS 2. Video a potrebné merania telemetrie prúdia do systému, kde sú spracovávané a užitočné výsledky sú štandardnými spôsobmi komunikácie prenesené na serverovú časť. V nej sú následne agregované a sprístupnené pre užívateľa. Spracovanie prebieha začlenením detektora objektov, pričom táto úloha patrí medzi najnáročnejšie a preto je jej výpočet delegovaný práve čipu na INCS 2. Výsledky sa zo zariadenia presunú späť a tvoria vstup pre identifikáciu osôb v čase, kde sa rozpoznajú ľudia z predošlých snímkov. V prípade keby bola osoba neidentifikovateľná, nebola skôr videná alebo systém si nie je istý či ju už videl, je namiesto toho vytvorená nová identita v podobe jej samostatného nového profilu. Informácie z oboch metód potom poskytujú základ algoritmu odhadu vzdialenosti, kde sa súradnice v obraze prevedú na vzdialenosť od kamery. Tieto odhady sú relevantné k fyzickej polohe a rotácii kamery v čase zachytenia snímku.

Implementované riešenie je schopné spracovať video priamo na hardvéri pripojenom k dronu. Tieto výsledky sú prenesené do sekundárnej časti aplikácie a prezentované užívateľovi vo forme grafu, ktorý obsahuje jednotlivé trajektórie. Systém je testovaný s modelmi detektorov objektov YOLO a SSD, ktoré poskytli pomerne presné výsledky iba za ideálnych podmienok a inak sa trajektórie dosť odlišovali od skutočnosti, čo sa týka vzdialenosti od kamery. Táto chyba v umiestnení je spôsobená nepresnou detekciou osôb a ich umiestnením v obraze v kombinácii s nedostatkami a rôznymi reštrikciami, ktoré si kladú metódy odhadu vzdialenosti.

Tracking People in the Video Captured from a Drone

Declaration

I hereby declare that this Master's thesis was prepared as an original work by the author under the supervision of Ing. Tomáš Goldmann. I have listed all the literary sources, publications and other sources which were used during the preparation of this thesis.

.....

Jakub Lukáč

June 10, 2020

Contents

1 Introduction **2**

2 Determining location of a person using video recording **4**
2.1 Object distance from a camera 4
2.2 Distance estimation in a single-camera system 5
2.3 Additional data from various sensors 8
2.4 Neural networks in image processing 8
2.5 Object Detection – object localisation and classification 14
2.6 Re-identification of person in frames 20

3 System Proposal and Implementation **22**
3.1 Similar works tackling these problems 22
3.2 Platform – The chosen hardware 24
3.3 Inputs – The necessary data for tracking 26
3.4 Additional software – The frameworks and libraries 28
3.5 Solution design 30
3.6 Application implementation details 33

4 Experiments **37**
4.1 Methods summary 37
4.2 Dataset & data sources 38
4.3 Experiments and testing 38
4.4 Overall system performance 40

5 Conclusion **42**

Bibliography **43**

A Usage **48**
A.1 Run instructions 48
A.2 Example of a configuration file 49

B Used libraries summary table **50**

Chapter 1

Introduction

Tracking people by video was and still is a difficult problem; however, ongoing research provides new methods every year that improve the precision of object detection, as well as decrease the computing power required. The understanding of different shapes, light conditions, camera angles and so many other factors make the problem hard to tackle by the machines. When humans are looking around, we see lots of objects. We can recognise at least the common ones quite easily. Our primary visual cortex is superior to old methods which have been implemented in computers.

The trend might be changing right now, and computers are getting better and better at the task, which enables many new applications. Two central problems of object detection are to identify what is an object and classify it. Identifying an object includes finding its exact location in the image. Then, the marked area can be assigned a certain class label. For this thesis, it is not only essential to know that an image can be classified as it includes a human, but also to know their location and ideally see a whole silhouette. A neural network is a way to overcome most of these issues. The current architectures of networks are almost as good as the human brain when it comes to object detection and classification.

The primitive detector design, which can be quickly proposed, is to create bounding boxes and scan the image parts one by one. It was indeed the initial approach of the first available algorithm which first searched for regions inclusive of potential objects. The detectors then used a method from machine learning to classify each region. Lastly, they would adjust the boundary boxes in the image. As it seems, it was quite a complex system. In the following chapters, all different improvements to this design are shown. Especially recently, the more popular single-pass solutions are available. They have a quick, straightforward processing pipeline with comparable accuracy to the previous multi-pass generation of detectors. Nevertheless, it is always a trade-off battle between speed and accuracy of the algorithms.

In order to achieve full tracking capabilities, people in the image have to be re-identified between frames as the need to follow a subject in the video is essential. All this image processing work is not a simple task in computer science and requires a vast amount of resources. Therefore, a final complexity must be an important aspect of a designed solution. This thesis aims to perform all the analysis on video captured from a drone exclusively. When a person is found and identified in the image, the final output of the proposed system is their location. Therefore, finding the subject's distance from the camera is expected. The location approximation is derived according to the camera position, which is effectively the location from drone sensors.

The drone industry is fast growing and offers interesting new utilisations, many of which have yet to be discovered. One of the main improvement in this segment was the ease of control over time. Small drones are packed with sensors and sophisticated electronics to enable simple usage for any type of user. Research and new inventions in the field itself and the field of robotics are helping to bring drones to the current market. Tracking people may be one of these crucial features. Overall, all the popularity opens up the new opportunities for research as vehicles get cheaper and more and more data is available at a minimal cost. All that played a major role for me to pick this project as well. The thesis is trying to build upon those existing systems and use them to create a reliable system and solve the task of positioning a person. The main goal of the resulting application is a capability to generate tracking metadata from detected individuals in the video.

There are several problems with tracking objects and estimating their locations, as outlined above. Hence, addressing the solutions is split into individual chapters in which each part of the final system is analysed from a different perspective. This way, I identified several major fields to cover. In Chapter 2, distance estimation methods, along with necessary image processing techniques, are explained. Especially, showing available algorithms for measuring the distance how far is the object from the camera in Section 2.1, and defining the position of an object within the image in Section 2.5.

Knowing this information is an adequate start for proposing a solution, more in Chapter 3. There could be found all important decisions as selected methods and algorithm or used hardware description. The chapter deals with both the proposed design and its implementation as well. Subsequently, the last two chapters cover all the experimenting work which was done, Chapter 4, and followed by the concluding chapter.

Chapter 2

Determining location of a person using video recording

In order to get the location of a person captured in video, the data from the camera and other sensors has to be processed using all the needed fundamental methods and ideas which are presented in this chapter. It includes using information about a camera's location, rotation and more as it gives a better indication about where the camera is pointing. For example, the orientation is essential due to limited field of view of the camera, more on that in Section 2.3. The location can be only obtained by utilising the correct telemetry data as well as with a particular position in a video frame. This selection of pixels should include a tracked person as a whole, from feet to head for the best results. Therefore, the second part of the chapter deals with the problem of detecting and identifying people in the image.

The field has been improving by leaps and bounds since it was possible to solve computer vision problems with machine learning. Neural networks models, trained with enormous publicly available data, helped many new applications. Convolutional Neural Network (CNN) has become the standard for image classification as a next milestone. They first gained popularity when they were used to compete with others using the ImageNet visual database. The image classifier based on CNNs won the database challenge in 2012 with a significant improvement of error rate [23]. The start of these types of networks was purely as a classifier of well-framed images. Shortly after that success, CNN techniques were used in object detection and image segmentation. It all led to the cutting-edge object detectors models that are described later on in Section 2.5.

In this chapter, I would like to briefly describe methods to approximate a relative position from camera, and neural networks, including ones with convolutional layers. Then, the chapter covers and summarises object detectors, followed by methods which explain re-identification (Re-ID) of person in time. Re-ID helps to maintain a person location history when there are multiple people in the frame. All these solutions help to implement a system which understands the image data and locates the persons.

2.1 Object distance from a camera

Most of the ideas stated in this section derive from the always emerging robotics segment of computer science. The distance measurement is taken between a robot and an object using various sensors. In terms of this work, only the visual ranging approaches are considered

and leave behind the technologies based on ultrasonic or laser measurements. Generally speaking, a camera as a distance measurement sensor is rather inaccurate and processing heavy one, on the other hand, is the one which comes with a low cost for the sensor itself. Robots can be equipped with cameras to avoid obstacles, to interact with the right objects in an appropriate way or to navigate in the environment.

The measurements based on camera sensors are likewise popular within the automotive industry. They are used for various assisting systems to help prevent accidents. Moreover, it gives vehicles extensive smart functionality, including pedestrian detection, lane departure warning, and forward-collision warning. The collision avoidance has an enormous impact on the society which encourage the new development in the field. The eventually presented methods are tested and researched as the demand for a cost-effective solution for drivers assistants. Data collected from a single camera is used to estimate a range between two road users. In the next section, this knowledge is generalised for determining the object's distance from the camera.

Possible alternatives to a single camera sensor

Alternatives are a stereo-vision, a laser rangefinder instrument, a sonar, or other active sensors. However, some of them might struggle with measuring multiple objects at once or in a short period of time. A completely different design was proposed in work [25], where the dual off-axis colour filter is attached to a single camera. This is just one example of an innovative approach to estimate an object's distance from a camera using the advanced computational method of enhanced general optical system. Such works prove that the problem of distance estimation is an important issue in many applied areas.

Besides the contemporary academic ideas and stand-alone advanced sensors, there are also complete solutions ready to answer the question of tracking distance. This new optics and sensors are available as a package, brought to the market by DJI company and its commercial industry drone Matrice 300. The vehicle's new payload[1] option combines multiple sensors, and it is capable of determining the distance and effectively the location as well of the object directly recorded in real-time. The range of the instrument like that can exceed 1 km with reasonable accuracy.

2.2 Distance estimation in a single-camera system

The available among vehicle distance estimation methods derive from elemental optics principals and the resulting projection perspective. A camera lens shows a recorded object in the image plane that creates a relationship between these two. The simplified system with no lens is an ideal pinhole camera, shown in Figure 2.1. Using just a single camera and no other sensors, the researched ways to measure the object's distance are size based distance estimation and position based distance estimation. Both of which are explained in further detail in [22], which provides the basis of distance estimation methods following in this section. The two methods have each a set of advantages one over the other, disputed that each has specific restrictions for the object itself or its surroundings.

[1] https://www.dji.com/hk-en/zenmuse-h20-series

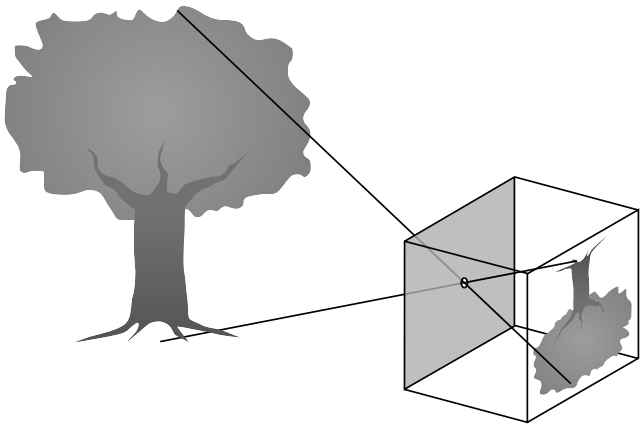


Figure 2.1: A diagram of a pinhole camera. [31].

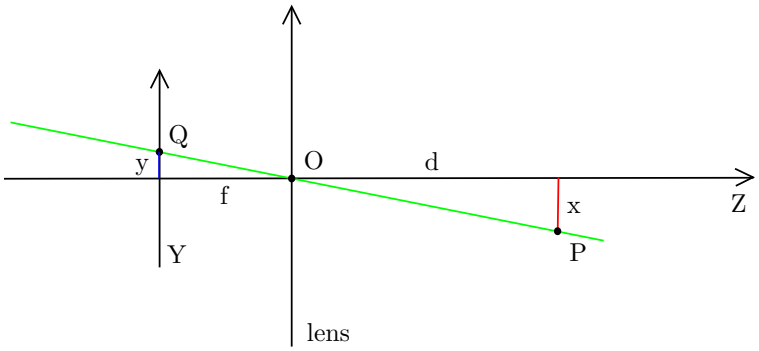


Figure 2.2: Size based distance estimation geometry [31].

2.2.1 Size based distance estimation

The size based approach uses the imaging properties of the camera, namely inversely proportional dimensions of the real object and its image. The object thereby has to maintain the same shape over time, and that is highly restrictive. Therefore, the typical case of use is often with the immutable structured object or rather a special marking sign. The mark has got another benefit as well, it is easy to detect in the image using much simpler methods, for example, colour filtering due to a known mark's colour. In contrast, a person as the object is a more variable element. The detected person should be standing straight or in general needs to keep a selected dimension relatively unchanged. This brings yet another problem into account that people's silhouettes come in all sorts of formats. Hence, a person height makes it the best candidate for the somewhat stable feature. The average height within the population forms one of the key parameters. To sum it up, the method imposes the restrictions for the object of interest.

The estimation fundamentals are illustrated in Figure 2.2, where x is the known object height, Y is the image plane, the object's height is y, f is a focal length of a used camera and d is a real distance from the object to the camera. The described system assumes that the image plane is parallel to the object height measuring plane. Accordingly, any deviation of the image plane or variance in the object height may affect the accuracy. The values are

related as follows

$$\frac{f}{y} = \frac{d}{x} \qquad \text{where} \qquad d = \frac{f \cdot x}{y} \tag{2.1}$$

both f and x are the parameters of the system, set upfront. The y on the other is inferred from the image in the same units as the rest. Image data is given ordinarily as a pixel matrix. Thus, the object, person, needs to be first precisely located in the image and then its height in pixels is converted to standard metric units. The distance is then calculated as

$$d = \frac{f \cdot x \cdot h}{y_p \cdot s} \tag{2.2}$$

where h is a height of the image in pixels, s is a camera sensor height. Both are constant for the particular camera recording. x_p is a height of the object in pixels; therefore, a way to detect and identify the object from the pixel input is required. In Section 2.5, the necessary methods called object detectors are presented. The detectors are based on neural networks which are also covered below to the necessary extent. The information about the object position in the image is as well needed for the next distance estimation principle.

2.2.2 Position based distance estimation

Position based estimation is again heavily dependant on an actual camera state. In comparison to the first method, it requires more information about the exact position and rotation of the camera. The main restriction of this estimation procedure is that it assumes the objects are located on a flat surface. The significant point in the image is a point of the object's contact with the ground. Therefore, there is no obligation to a person's height or posture, although the person still needs to be recognised in the image.

The distance d is defined as

$$d = a \cdot \tan\theta \tag{2.3}$$

where a is camera altitude and θ is the angle of the observed object's contact with the flat ground and camera's altitude plane, as demonstrated by Figure 2.3. The angle can be deduced from $\theta = \theta_c - \theta_{obj}$, θ_c is the angle of camera direction, and θ_{obj} is the angle of camera direction and the object's contact point. The camera position and rotation can be obtained from additional sensors and systems onboard. However, θ_{obj} needs to be computed from the data. Problem schema shows that

$$\theta_{obj} = \tan^{-1}\left(\frac{\frac{h}{2} - y}{f}\right) \tag{2.4}$$

where f is the focal length of the camera, h is the sensor height, and y is the distance from the bottom of the sensor to the object's ground contact point P. The equation (2.4) assumes that the camera's lateral axis, or pitch axis, is parallel with the ground flat plane. In order to avoid using the focal length, it can be expressed as

$$f = \frac{h}{2\,\tan\theta_r}\ , \tag{2.5}$$

θ_r is the half-angle of the camera's field of view angle, thus $\theta_{fov} = 2\theta_r$. Then, it is possible to deduce that distance

$$d = a \cdot \tan\left(\theta_c - \tan^{-1}\frac{(h - 2y) \cdot \tan\theta_r}{h}\right)\ . \tag{2.6}$$

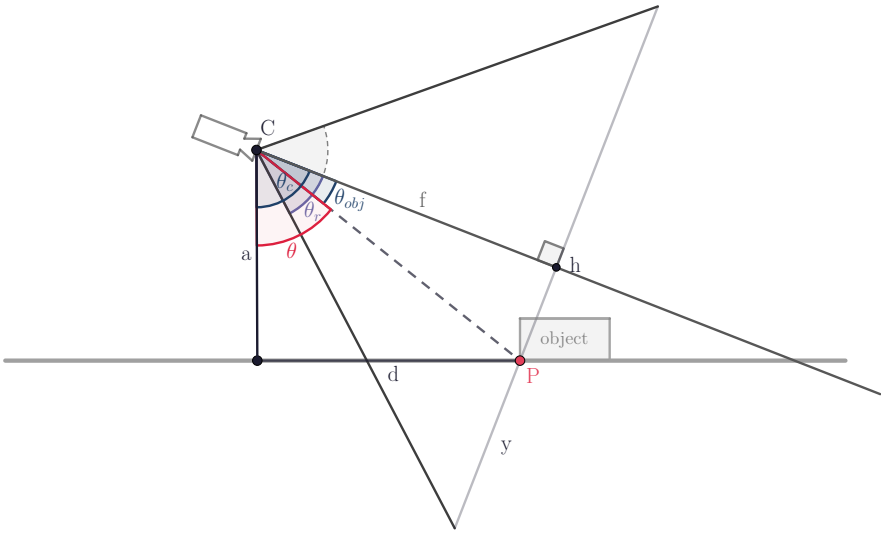


Figure 2.3: Position based distance estimation geometry, created with GeoGebra and based on the figure in [22].

2.3 Additional data from various sensors

The main focus here is on video captured from a drone; therefore, the intended camera placement is one attached to the aerial vehicle. Then, the camera is usually mounted using a gimbal which allows better camera stabilisation. Additionally, it can also provide extra controllability in some axes. The principle and terminology are based on typical aircraft rotations [2]. A drone gimbal often allows a camera adjusting in pitch axis, or more advance gimbal supports rotations in yaw axis as well. All of it with the general location data mentioned above makes an apparent view on how and where the camera is positioned.

Next important information is the parameters of the camera, not only the image specification, but also details about its lens, focal length, sensor size, and field of view. The camera is the primary source of data, but its position in surroundings is also essential for extracting the objects' distances. Supporting sensors are the drone instruments for measuring the precise location in all three dimensions.

2.4 Neural networks in image processing

A neural network is a type of machine learning model. Its primary purpose nowadays is to understand speech and image and support us, humans, at various labour (medicine, science, repeating jobs). The real advantage of the networks shows up when the task to be solved is easy for people to perform but hard for people to describe formally to computers. Neural networks consist of interconnected artificial neurons; effectively, these neurons are nodes in an oriented graph and often organised in layers. In the context of computer vision, artificial neural networks are usually deep neural networks. These are neural networks with multiple hidden or inner layers. Computing system with the structure like that vaguely mimics the biological neural network, thus the name *neural network*. This section draws mainly from book *Deep Learning* [16], other more specific sources are referenced when used in the text.

The difficulties faced by the inability to describe task formally suggest that neural networks need the ability to acquire the necessary knowledge by extracting patterns from raw data. Each neuron acts as an independent computing unit that takes a set of input values, performs computation and produces a single output value. The computation uses internal neuron's parameters, weights and biases. The weight represents the relative importance of input value or connection value. The bias can shift the resulting value of the output. Then, neuron, also called perceptron, operates as follows: weights are applied to input values by finding the dot product,

$$\mathbf{w} \cdot \mathbf{x} + b = \sum_{i=1}^{m} w_i x_i + b \tag{2.7}$$

where $\mathbf{w}$ is the vector of weights, $\mathbf{x}$ is the vector of input values, and b is a bias [42]. The output, as shown in equation (2.7) can vary a lot; therefore, its value is given to activation function. This function adjusts neuron's behaviour according to the final application. A simple use-case for the desired output can be states *ON* (1) or *OFF* (0), depending on input connections. Various examples of activation functions can be found in the following Section 2.4.2 alongside with their typical usages.

Neurons can be connected in almost any possible configuration, although there are several well-researched architectures of the neural connections [51]. The most important ones, from this work perspective, are feed-forward networks. Where the information flows from the front to the back, there are no cycles nor loops in the network. In general, two adjacent layers are usually fully connected. Feed-forward network architecture consists of:

1. Input layer – holds the initial input data, it can be numeric data, pixel values of an image (frequently converted to greyscale), text or any digital signal data (speech). The layer typically holds data from the environment, no computation takes place here, the information is passed to the hidden layer.

2. Hidden layers – are all the interconnecting layers between the input layer and the output layer. The layers hold information about recognised patterns, each hidden layer can perform different computation (specialised layers).

3. Output layer – provides results based on outputs of all previous layers, it can be a discreet value (affiliation to a particular class), or a continuous value (probability).

This architecture facilitates straightforward training, tuning the weights and biases, of the network. Feed-forward neural networks are usually trained with back-propagation, a popular supervised learning method.

2.4.1 Convolutional Neural Networks

Equally important convolutional neural network (CNN) [24] is the improvement of machine learning methods which were mentioned above. This particular method is primarily used for image processing but can also process other types of input, such as audio or time series. In general, CNNs handle well any data which has a known grid-like topology.

As the name of the network suggests, it employs a mathematical operation called convolution. The input data is fed through convolutional layers instead of normal ones, meaning not all neurons are connected to all neurons. Each neuron only connects with adjacent neighbouring cells from the previous layer, usually not more than a few. These convolutional layers also tend to decrease in size as they are deeper in the network. Furthermore,

the pre-processing required for CNNs is a lot lower when compared to other architectures. As in feed-forward networks, data is filtered by manually engineered algorithms. Convolution layers have the ability to learn these filters with enough training examples. For instance, the layers reduce the image into a structure which is easier to process, without losing features which are critical for getting an accurate prediction. Besides the convolutional layers, CNN also incorporates downsampling layers, called pooling layers. This type reduces the level of details for afterwards more unequivocal prediction making. Both layers are linked using ReLU as an activation function.

An example of underlying CNN architecture is shown in Figure 2.4. All the newly introduced layers are part of the diagram as they would be in real network design, the order of operations as shown is rather typical for CNNs.

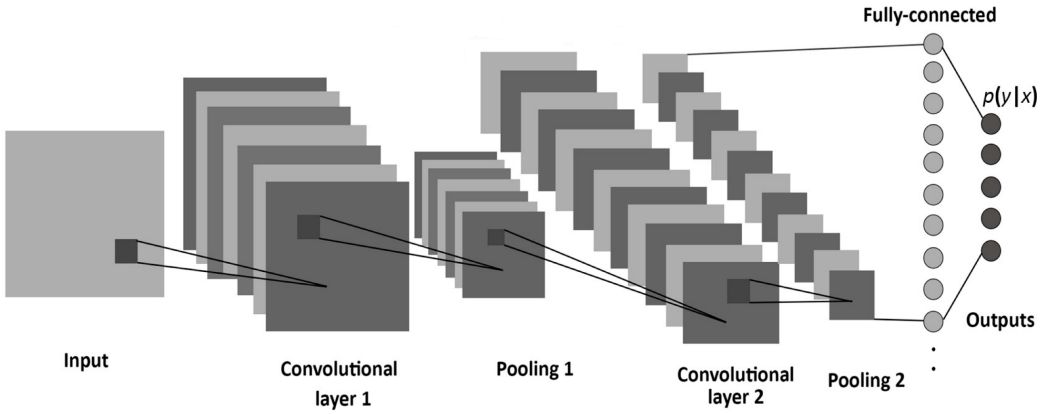


Figure 2.4: Diagram shows the fundamental architecture of the convolutional neural network [49]. It consists of convolutional layers, subsampling layers also called pooling layers, and fully connected feed-forward network layer at the most right which predicts the final category for input.

Convolution

In machine learning applications, convolution as a front operation extracts and preserves essential features from the input during the learning stage. The operation takes as the input a multidimensional array of data, and the kernel which is a multidimensional array of parameters.

Finally, an image is a two-dimensional array therefore for an input image I and a two-dimensional kernel K of size $m \times n$ convolution [16] is

$$S(i,j) = (I * K)(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n) \ . \tag{2.8}$$

The formula is straightforward to implement and well-known in image processing. However, when it comes to practical usage, machine network libraries often implement a related function called the cross-correlation. It is the same as convolution but without flipping the kernel and frequently still refers to that as *convolution*. A practical example of the operation for a two-dimensional 3×3 kernel applied to a two-dimensional array of size 5×5 is illustrated in Figure 2.5.

Each convolutional layer holds the self-obtained characteristics. When the network tries to predict an output, this layer type indicates if the feature is included in an input

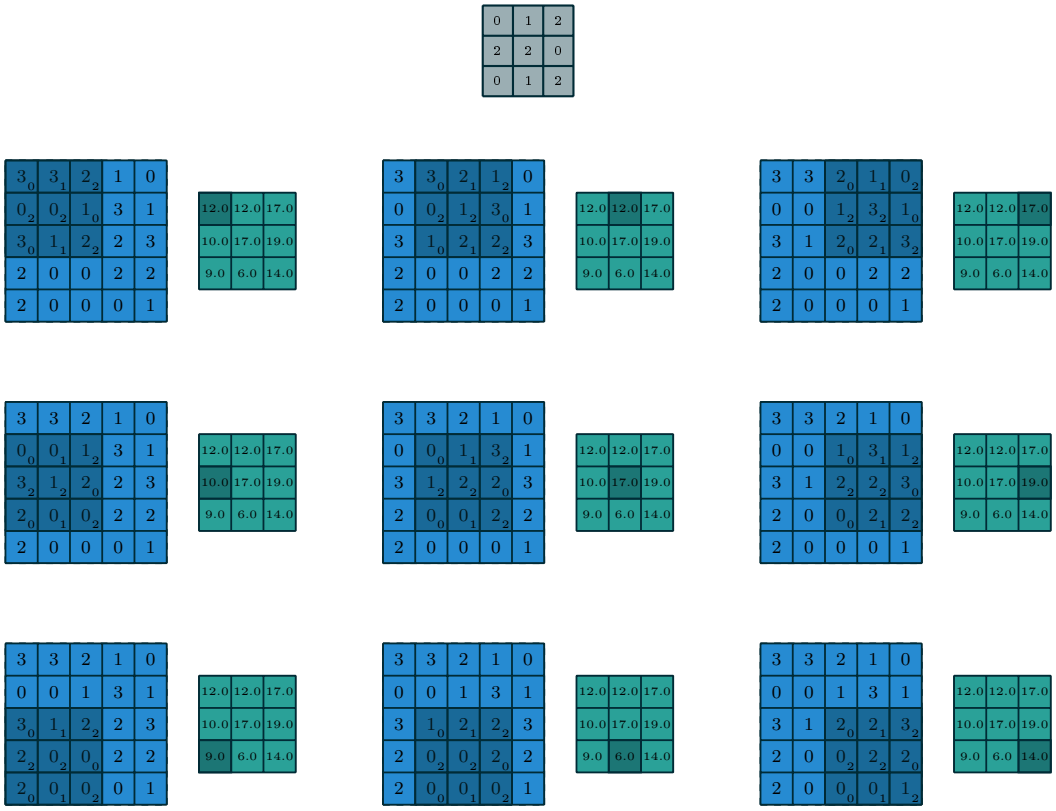


Figure 2.5: An example of a convolution operation from [9], a kernel at the top in grey, an input array in blue, and a result in green.

or not. Lastly, the fully connected feed-forward layers predict the output of network based on presence or absence of the features. The features are a more abstract concept of all the individual pixels of an image. Front layers tend to carry information about fundamental aspects such as edges and their orientation. Going deeper into the network, layers holds more and more abstract characteristics. They detect the object as a whole and understand its image representation better. This brings efficiency to the terminal layers but also enables more precise predictions regardless of infinite variations of objects sizes and angles it was captured from.

Pooling

Downsampling layers called pooling layers generally succeed the convolutional ones. Pooling is a method to filter out the level of detail, by reducing the input size of the layer which comes next. The goal is to scale down the dimensionality of each input matrix but retain important information. Each pooling defines a neighbouring window of a given size which is downsampled to a single value. There are a few commonly used types of function. *Max pooling* takes the largest element from the set feature subregion. Besides that, *average pooling* could take the average of all elements. Analogically, outputting sum of all elements in the subregion is called *sum pooling*. Two of the types used in CNNs are shown in Figure 2.6.

A process of adjusting the pooling layers can reduce the computational power needed, as dimensionality decreases for following layers. The drawback, of course, is losing the possibly significant details which might improve precision. Hence, max pooling also helps to suppress noise from its input and can deliver better results than, for instance, average pooling [44]. Another expected benefit of pooling is persisting the necessary dominant features in rotational and positional invariant manner.

$$\begin{bmatrix} 12 & 21 & 86 & 1 \\ 8 & 51 & 19 & 38 \\ 35 & 28 & 76 & 95 \\ 54 & 15 & 42 & 63 \end{bmatrix} \xrightarrow{\text{pooling with 2x2 window and stride 2}} \begin{cases} \text{if max pooling} & x = \begin{bmatrix} 51 & 86 \\ 54 & 95 \end{bmatrix} \\ \text{if avarage pooling} & x = \begin{bmatrix} 23 & 36 \\ 33 & 69 \end{bmatrix} \end{cases}$$

Figure 2.6: Practical example of different pooling operations.

2.4.2 Activation functions

Activation functions [18] add non-linear transformation, which enables neural networks to perform better and learn more complex patterns. They sit in between the raw output of the current neuron and its output going to the next layer. Another aspect of them is that the function has to be incomplex to compute as it must be calculated over and over for sometimes millions of neurons. The important non-linear functions, which are normally used in image classifiers and CNNs, are listed here:

- tanh – Hyperbolic tangent is usually used as activation for hidden neurons, its values are zero-centred and set between -1 to 1. This helps to make the learning of neurons much easier. It is very similar to a popular sigmoid function, also known as a logistic function. However, both of them are computationally expensive:

$$f(x) = \tanh(x) \ . \tag{2.9}$$

- ReLU – Rectified Linear Unit is the most widely used activation function nowadays. Similarly to *tanh*, it is part of hidden layers, especially implemented right after convolutional ones. On the contrary, the main advantage of the function is its simplicity, using just basic mathematical operations. The efficiency allows the network to converge faster; therefore, ReLU is a go-to function for an arbitrary problem. Although, it has some disadvantages, for example, the dying ReLU problem [28] which for some cases can be suppressed by *Leaky ReLU* version of the function (gives proportionally small negative output for negative input). ReLU gives an output of x if x is positive and 0 otherwise:

$$f(x) = \begin{cases} 0 & \text{for } x \leq 0 \\ x & \text{for } x > 0 \ . \end{cases} \tag{2.10}$$

- Softmax – This function often outputs overall predictions of a neural network. Where the output is normalised by the sum of all the outputs, which gives a value between 0 and 1. For instance, the result of softmax can be directly interpreted as the probability of a particular class in the context of classifiers. Hence, the function can handle yielding probabilities for multiple categories.

Besides the listed activation functions, other ones can be used as well, which depend on the data and intended application. When building a model and training a neural network, the choice of the right function is crucial. Often, experimenting with different activation functions might lead to much better performance. Researchers still bring new proposals for the task, promising replacement of ReLU could be seen in *Swish* [33] function. The final choice for activation function is generally influenced by a problem domain and the designer's experience.

2.4.3 Well-used Convolutional Neural Networks architectures

The arrangement of convolutional layers and their properties are evolving across time, which resulted in many popular network architectures. The networks compound of many diverse layers sizes and incorporate complicated operations in pursuit of achieving the best results. They are slowly becoming just tools to get the job done, and systems often treat them as black-boxes. The fundamental performance of each model replaces the importance of knowing and understanding its internal structure. For instance, utilised models include AlexNet, VGG, Inception or ResNet. They are usually incorporated into the frameworks in order to get their most optimised versions to end-users.

The mentioned designs, as well as other ones, form a fundamental core of object detectors covered in following Section 2.5. The overall history of all different architectures is evolving, and new innovative layer arrangements are frequently researched. They might be intended as both general purpose ones or narrowly focused for a particular task where they can excel and beat human craftsmanship. Table 2.1 illustrates that the commonly used CNNs can differ from each other by accuracy but also by the size of each model. The fact, how large the model is, determines its eventual computational complexity and plays a crucial role in the final production application. It is always the speed vs accuracy trade-off; the used table is just an example from one of the deep learning frameworks. The applications may run the inference for each one of them using straightforward calls that can give programs an added value for the user.

The models are also peeled off their training envelopes, which speed up the weights setting up phase. The extra processing allows the back-propagation of the errors while processing the training samples. Without all this, they are efficient and prepared to infer the knowledge they gathered from a training dataset. Next, the number of parameters, the depth, or inference processing time are yet another defining factors for the CNNs practical application.

Model	Size	Top-1 acc	Top-5 acc	Parameters	Depth
Xception	88 MB	0.790	0.945	22,910,480	126
VGG16	528 MB	0.713	0.901	138,357,544	23
VGG19	549 MB	0.713	0.900	143,667,240	26
ResNet50	98 MB	0.749	0.921	25,636,712	-
ResNet101	171 MB	0.764	0.928	44,707,176	-
ResNet152	232 MB	0.766	0.931	60,419,944	-
ResNet50V2	98 MB	0.760	0.930	25,613,800	-
ResNet101V2	171 MB	0.772	0.938	44,675,560	-
ResNet152V2	232 MB	0.780	0.942	60,380,648	-
InceptionV3	92 MB	0.779	0.937	23,851,784	159
InceptionResNetV2	215 MB	0.803	0.953	55,873,736	572
MobileNet	16 MB	0.704	0.895	4,253,864	88
MobileNetV2	**14 MB**	0.713	0.901	**3,538,984**	88
DenseNet121	33 MB	0.750	0.923	8,062,504	121
DenseNet169	57 MB	0.762	0.932	14,307,880	169
DenseNet201	80 MB	0.773	0.936	20,242,984	201
NASNetMobile	23 MB	0.744	0.919	5,326,716	-
NASNetLarge	343 MB	**0.825**	**0.960**	88,949,818	-

Table 2.1: Comparison of different CNNs architectures in terms of their size and accuracy [8]. The accuracy (acc) was measured using the ImageNet dataset.

2.5 Object Detection – object localisation and classification

The state-of-the-art detectors are described in the next few sections. Inspiration for today's algorithms originated from the initial paper which proposed a combination of region proposals with a convolutional neural network, also known as R-CNN [15] or Region-based Convolutional Network. All other main detectors architectures built upon this idea and incrementally improved each previous architecture's precision and performance.

The goal of an object detector is to find a boundary box that contains an object and then classifying the located object, an example of results can be seen in Figure 2.7. To summarise, this was an initial approach of the first available algorithms which integrated convolutional neural networks in detection. R-CNN was demanding a lot of computational power to do so as a result of many redundant operations. The follow-up architectures fixed one problem after another to gain better performance. Popular methods to locate multiple objects in a single image are summarised in this section below.

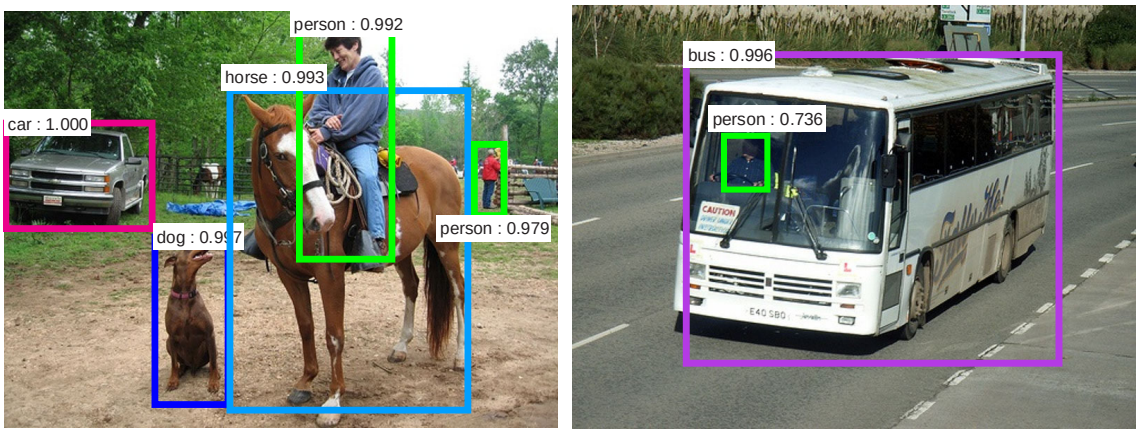


Figure 2.7: An example of output images from an object detector with various detected objects [40]. Showing bounding boxes for each detected object with its type and confidence score (from *not confident* – 0.000 to *very confident* – 1.000)

2.5.1 R-CNN, Fast R-CNN, Faster R-CNN

Object detection is effectively finding regions with different objects in the image and classifying them as individual segments, in the same way, R-CNN [15] performs literally that. The previous generation of detectors was a sophisticated collection of specific methods to cover all the different aspects of an image. R-CNN authors proposed a much more straightforward and scalable approach. They combined region suggestions and the breakthrough algorithms [23] from convolutional neural networks field.

R-CNN detector is organised into three main modules, as shown in Figure 2.8. The first extracts region proposals by using selective search method without an obligation to know what exactly is in the actual region. The second part wraps the proposals pixels to fulfil constraints of CNN which then obtains defining features. The final stage is composed of a support vector machine (SVM) that classifies whether it is an object and to which class it belongs. Additionally, resulting bounding boxes can be tightened by linear regression, so the coordinates suit better the actual dimensions of the objects. The critical drawback of this solution is that the last two parts are executed for each proposed region. However, there are no doubts about the genuine accuracy of R-CNN, with an improvement by more than 50% relative to the previous algorithms.

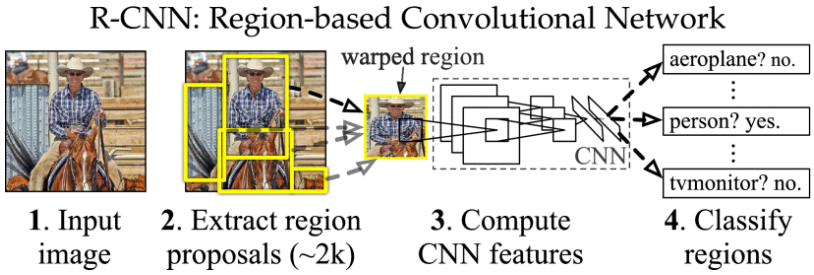


Figure 2.8: R-CNN object detection method proposed in [15], showing the outcomes of the input image processing during the detection stages.

The number of generated regions for the method is around 2000 category-independent proposals [15] for an average input image. Every one of them requires a forward pass of the

CNN, and that could be a computationally complex task. Moreover, models included in R-CNN are trained separately, which makes the learning phase hard too. An improvement was then inevitable, a new design of Fast R-CNN [14] tackled both these issues.

Fast R-CNN still generates a set of object proposals then passes the image through CNN only once. This development was achieved with a new algorithm known as Region of Interest (RoI) pooling that allowed a shared computation over the proposals. It effectively shares the forward pass of the convolutional network, and the output features for each region are obtained by selecting a corresponding region of the convolutional feature map, as shown in Figure 2.9. Specifically, for each object proposal, the RoI pooling layer extracts a fixed-length feature vector. Furthermore, the number of models is also reduced with the intention of simplifying the training process and fine-tuning each. The original SVM is replaced by a softmax classifier where their both performed equivalently. That all led to more unified training, rather than having the three training stages of the first R-CNN.

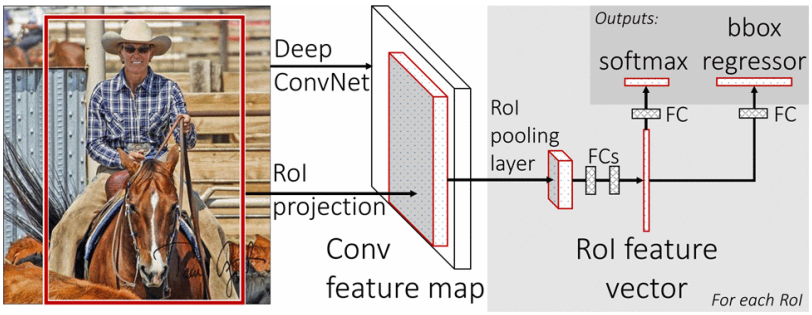


Figure 2.9: Fast R-CNN object detection method, showing the improved architecture with the shared convolutional neural network (*Conv feature map*) and a much straightforward training process, proposed in [14].

Both proposals are still suggesting to use selective search that is a complicated and slow algorithm. More importantly, this part of the detector was identified as the next bottleneck. Therefore, it led to the succeeding architecture called Faster R-CNN [40]. Selective search is replaced by already in place convolutional network, reusing it to search for region proposals as well. A one CNN which helps to find an object and classifying it, also enabled training only a single model. The improvement is pushing R-CNN detectors family towards the faster single-pass detectors. It is accomplished by a fully convolutional network which is added after the features extraction step, creating the independent region proposal network or RPN. The idea behind the RPN is that it moves a sliding window over the feature map and proposing the bounding boxes and scores for them. These bounding box proposals are afterwards examined even further to determine how likely they really include an object. The picked sliding windows should accommodate the objects of certain common aspect ratios and sizes, then these are also called anchor boxes.

2.5.2 Mask R-CNN

The previously covered methods are detecting an object, and the result is a rectangular bounding box, more specific its coordinates. Mask R-CNN [19] added to the mix complete binary map as a more granular result. The detector predicts, for each pixel in the input image, if it is part of the object or not.

The new architecture directly adopts both stages from previous improved R-CNN networks, namely the RPN network and following class predictor. However, Mask R-CNN outputs a binary mask for each proposed region. This is achieved by another convolutional network added in parallel with the second stage of the original design. The extra branch predicts segmentation masks in order to separate the objects from their background.

The authors soon realised that the regions of the feature map selected by RoI pooling in prior Fast R-CNN were slightly misaligned. The level of pooling precision needed for bounding boxes was much lower than one needed for the pixel segmentation. RoI pooling is using quantisation when downsampling a feature map that causing the misalignment between regions in the input image and the extracted features. The classification is yet robust enough to compensate for these small translations. Nevertheless, this negative effect on the segmentation had to be handled by a new method called *RoIAlign*. In RoIAlign, a sampled point is computed by bilinear interpolation from its neighbourhood, to get a more precise binary map prediction and avoid the problematic rounding of RoI pooling.

2.5.3 Single Shot MultiBox detector

Single Shot MultiBox Detector [26] or SSD is one of the next-generation architectures of object detectors. In contrast to R-CNN based detectors, SSD focuses on a single forward pass detection from the beginning hence its name. Outstanding performance with low computational power required is a key to real-time object detection.

SSD skips the process of generating object proposals and instead sets default boxes with various aspect ratios and scale. The architecture is composed of several convolutional layers or filters. Then, the outstanding high accuracy is obtained by delivering predictions at different scales of feature maps as well as at diverse aspect ratios. This led to improvements in low-resolution input images correspondingly. The one convolutional neural network concept also makes it easier to train and creates better ground for optimisations. SSD design still competes rather well against previous cutting edge object detectors, but it is much faster.

2.5.4 YOLO, YOLO9000, YOLOv3

An approach, which was introduced by this line of detectors, is significantly unlike classifier-based systems. It tries to process the entire image as a whole, and this way, predictions can derive information from the global context in the image itself. YOLO, which stands for *you only look once*, is therefore another example of a single-pass detector. It has got much popularity and aims to be a versatile solution for any system that demands object detection. There are three main sequential versions of the detector where each is improving particular deficiencies of its predecessor. Moreover, the 4th iteration of the detector [4] was published just recently, this time by a different group of authors. The architecture is fast and yet precise, enabling real-time operation capabilities, and accuracy reaches the results of far more complex models.

The architecture details, shortly explained in this very paragraph, are gathered mostly from the original paper *You Only Look Once: Unified, Real-Time Object Detection* [36]. The goal was simple, to create one neural network and feed it with an image, then get the detection done in a single-pass thus the network output is a collection of labelled bounding boxes. This brings to a mix mainly a problem of how to train the model such as that. Authors of YOLO came up with new methods to tackle the dilemma. First, they had to define a structure of the single-pass output. It consists of many predictions and accordingly, their confidence score. Neural networks can output these many values without any prob-

lems, for instance, ImageNet works with predictions for hundreds of different classes. To summarise, YOLO transforms the problem of detection to a problem where an input image outputs the corresponding tensor. This tensor encodes all the possible object predictions as the separate sets of values which represents location, class and confidence properties. An individual image is cut into S by S grid then each of these grid cells is responsible for B bounding boxes, the boxes centres fall in that cell. The box details involve five attributes altogether: confidence value, x coordinate, y coordinate, width and height. In addition, every cell determines C class probabilities which means it predicts only one object regardless of the number of boxes B. The final output tensor Y of the YOLO architecture is then defined as

$$Y : S \times S \times (B * 5 + C) \ . \tag{2.11}$$

For example, a cell in YOLO implementation, trained on Pascal VOC dataset [10], has two bounding boxes, for each the 5 values, and it holds 20 different class probabilities. The used grid size is 7, final tensor is $7 \times 7 \times 30$, which prompts neural network to generate about 1500 output parameters.

A YOLO network has 24 convolutional layers, or 9 for its fast version, followed by 2 fully connected layers. The training process of the proposed network is more important than the further details about the structure which are covered well by the original paper. From the definition of the output tensor, only one bounding box is responsible for object detection. The right box is selected based on the maximal similarity with the ground truth from a training set. Boxes specialise at predicting specific sizes and aspect ratios this way. The architecture optimises for a simple sum-squared error between the output and the ground truth. Subsequently, the loss calculation adds together classification, localisation and confidence loss. It weights all the errors equally, which might lead to instability during training. Therefore, the authors use two extra parameters λ_{coord} and λ_{noobj}, the first to increase a box position loss and the second to decrease confidence loss of a box that only contains background. The labelled images are converted to tensor representation accordingly. The right class is assigned to a cell which includes the centre of an object. A cell's box with an object and the highest IoU, see Section 2.5.5, gets its confidence increased, all other boxes get it decreased. The coordinates of the box, which its confidence is being increased, are also adjusted to match the ground truth. This high-level description of loss function demonstrates the ideas behind the method, the equations describing the whole error arithmetic can be found in [36]. The network training process additionally uses pre-training on ImageNet, stochastic gradient descent with decreasing learning rate and necessary data transformations. The tensor represents a raw output, hence the prediction sets are effectively filtered by a minimal confidence threshold and reduced by removing the duplicates. If the cell contains a bounding box with high enough confidence score, then the box class is decided based on the cell class probabilities. Finally, the box is the concluding object detection result.

However, there are a few disadvantages to YOLO architecture. It struggles with detecting small objects since the grid cell can only predict a finite number of bounding boxes and just one class, which results in ignoring some of these objects. Another essential flaw, which was discovered in comparison to other systems, was misalignments in the localisation. The loss function does not compensate for errors in small bounding boxes versus large bounding boxes. A little mismatch in a large box is generally negligible, although the same shift in a small box has a much greater effect on IoU. Lastly, the model uses quite rough granularity of features as it downsamples the input multiple times. As an illustration, SSD architecture has higher accuracy while still maintaining the real-time processing capabilities. Despite

these facts, YOLO's authors also tested the final model on artwork images, the network outperforms other object detectors. It is often interpreted as the model generalising better in other domains or for unseen images. Certainly, the new approach is strongly present, but the used methods depend on previous conclusions and past work in fields of image processing, especially in the object detection domain. In general, YOLO has performed well in real-time applications, has got a high frames-per-second rate, but *Faster R-CNN* was still better with respect to accuracy.

Another paper from the same authors shortly followed all the work above. The second version of the YOLO object detector was revealed in *YOLO9000: Better, Faster, Stronger* [37]. The new version tries to attain an objective of compensating for imperfections in the architecture, mainly improving localisation and sensitivity in order to get all the detections. The first improvement was adding a pre-training phase to the used ImageNet model with a bigger image resolution. The effect was an increase in features extraction and better overall accuracy when compared to the original design, pre-trained with just the half image resolution. The model tuning with the larger input size is rather time-consuming, therefore, it is done at the beginning of the training for only 10 epochs. Next, the authors decided to experiment with the anchor boxes, similarly to the Faster R-CNN ones. Quite a novel approach was extracting the common boxes sizes and aspect ratios from the training data by K-Means Clustering method. However, the anchor boxes resulted in a slight decrease in detection quality. The full list of all incremental enhancement is nicely summarised in the paper, these are just the main ones.

The original YOLO uses the fixed input resolution, with robustness in mind the support of various input sizes was incorporated to the model for both training and inference. The only constraint is that the image size should be dividable by 32, as the network is downsizing images by this factor. This is possible on the grounds that the network uses fully convolutional layers and parameters can be reused when used like that. The variable input size enables the model to be used with smaller or bigger images to improve the processing speed or the accuracy, respectively. The hypothesis behind this is that the change acts as data argumentation, and the network is able to recognise various object sizes, as the object size inevitably change accordingly with the image resolution. This way, the original dataset can be extended, moreover, it might prevent the over-fitting to some extend during a large number of training epochs while looking at the images repeatedly.

Next change is a custom network design as a backbone, introducing *Darknet19*. The model has 19 convolutional layers, hence the name, and several max pooling layers. It comes as an object detection network that can be a foundation for the next innovations. By detaching the classification error back-propagation and the object detection error, Darknet19 has the ability to train itself on both the detection datasets (COCO) and also on the classification datasets (ImageNet). When the image metadata includes a known class label, then the classification error is back-propagated as in a regular classifier. However, when the image data is richer and includes a class label along with its location, then both errors are back-propagated. The practical implication could be a creation of object detector which can detect, for example, dogs but the enhanced classification labels their bounding boxes even furthermore with the respective breed.

Moreover, the authors showed the network could detect objects for which it has never seen the bounding boxes during the training phase while encountered just their classes alone. This was mainly possible with the defined structure of words and hierarchical classification. The words are sorted in a tree structure which goes from abstract root of *physical object* label to more and more specific labels down to specific leave labels, such as particular

dog breed. The structure also solved the problem of training with a dataset composed of less specific labels. In this case, the network can still give a high confidence inference for dog object while giving a less confident answer for its exact breed. With the hierarchical classification, the architecture is capable of detecting more than 9000 classes that explain the name *YOLO9000*.

The latest version of YOLOv3 [38] object detector was another incremental update and followed well the original message of the architecture: being extraordinarily fast and accurate. It came with just small improvements and changes in the design which reflect the recent breakthroughs in detection at the time.

New network design is now fully responsible for feature extraction, as Darknet19 is replaced by 53 layer Darknet53. The structure needs less floating point operations than state-of-the-art residual networks; however, achieves an akin accuracy much faster, authors claims about two times speeding up. The accuracy comes with a price though as version 3 is slightly slower than its predecessor. In general, the improvements helped YOLO to achieve similar accuracy as Faster R-CNN and therefore finally overcame SSD detector as well. Nonetheless, it is still a reasonable option when speed matters.

2.5.5 mAP (mean Average Precision)

Mean average precision (mAP) serves as a metric to compare the accuracy of popular object detectors listed above. Intersection over Union (IoU) measures the overlap between 2 boundary boxes. IoU with a set threshold determines whether the prediction is a true positive or a false positive. The mAP is an average precision among all different classes which system can recognise. Then, average precision is gathered using IoU over some threshold. If IoU value is above the threshold then the prediction is considered as correct.

2.6 Re-identification of person in frames

The re-identification or re-id problem is often related to multiple cameras surveillance systems where the system needs to able to monitor a person's movement. This work intends to use only a single camera setup which makes the problem much more manageable. The implementation can depend on relative localisation constraints within the frames, meaning the person cannot suddenly relocate from one side of one frame to another side of the next frame while the premise is that the frames are taken in a short period of time. Besides, the system has to operate with such a restriction wisely and consider a person leaving a field of view and then reappearing somewhere else as a valid case. This section gives a solution for the system problem of connecting the location estimates for a specific person in time with the intention of assembling the trajectory of their movement. However, re-id can be difficult, but for the particular proposed system, a false negative recognition is not a major flaw. The system would create a new identity for a person when the match is not confident enough, which results in the trajectory interruption.

A Person re-identification task is the problem of identifying identical people across images in time or across images from multiple cameras. The re-id method extracts and compares the features of a person from an image with the already saved features of other persons in past images, and determines whether there is a match. It is a well-known problem in surveillance systems, and all the knowledge of the field can be clearly applied for the solution. Moreover, video recordings are usually taken from a distance that makes the video very similar to the one from drone's camera. In the context of this work, the

identification is a crucial part of the tracking mechanism which allow the system to match multiple individuals within camera frames in time.

For instance, face recognition is not a valid approach as the people are captured from a significant distance. Instead, it is necessary to use information about their whole body look. Therefore, the methods, which allows the use of features such as body figure proportion, clothing, or walking pattern, are better suited for this. The resulting algorithm needs to take into account that people's images have very low resolution, the lighting conditions are unstable, and the background around them may change drastically in time. The pose of people may vary too, and they can be partially or entirely occluded. All of this makes precise identification hard and challenging. The methods which can tackle the stated problem and help to build the final system are *Rethinking Person Re-Identification with Confidence* [1], *AlignedReID* [29], [53], *SORT* algorithm [3], [52], or primary image features comparison methods for example based on histograms.

Chapter 3

System Proposal and Implementation

Training people in video solution can be simply described as estimating the distance to each individual in a currently processed image. The estimate is further used to locate the individual relatively to a camera positioning system. This chapter consists of both hardware and software implementation parts of the proposed solution. The first few sections aim to describe the targeted use case and its hardware, including the important limitations of the hardware. The choices for the hardware are also defined by used software toolkit, especially OpenVINO[1] software to take advantage of Intel Neural Compute Stick 2, the USB stick featuring a manycore vision processing unit. Its brief description and relevant usage are illustrated below, together with other design choices. The essential part of Solution Design, Section 3.5, goes through the system architecture proposal. Then, the final realisation section, Section 3.6, analyses the parts of the final implementation.

The almost like real-time processing is the goal here, however using restrictive hardware, advanced object detection, and identification methods may adversely affect this target. The more realistic solution of getting the locations of people in a frame would achieve results in subsecond time or achieving frames per second (FPS) rate in the orders of ones per second. This is covered more comprehensively in Chapter 4, which immediately follows.

3.1 Similar works tackling these problems

All the comparable systems are briefly presented in this very section. The critical aspect of the discussed solutions here is the use of a single camera sensor as the main source for examining the surroundings. As proposed earlier, the monocular camera setup is quite popular in the automotive industry. The camera as a sensor is a cheap multi-purpose detection device.

Some of the key studies, which are by some means related to the overall problem of object tracking, are [32], [43], [46] and especially [22] which provides the basis for distance estimation methods, Section 2.2. They all use video from a camera to detect enclosing vehicles and estimating their distance in order to assist a driver. The detection methods vary and are aimed, of course for vehicle detection. They often use unique features of the environment such as the road which might be a notable restriction in terms of using these proposed solutions. For example, finding the position of vanish line or the vanishing point

[1] Open Visual Inference and Neural network Optimisation

in an image could be an essential clue which is based on road lines. By knowing its accurate position, it gives this approach much-needed precision over the traditional pinhole camera principle. Another widespread technique across previous works was an object behaviour prediction. An object's trajectory is modelled to obtain the most likely future position, based on its previous positions and motion. It slightly shifts the object tracking from a single frame processing to a more motion like analysis, where speed and movement direction are the crucial factors depending on multiple frames sequence. Nonetheless, the later proposed system intends to process only the current image to maintain its own complexity. In addition, papers include lots of well covered past work surveys that make them a valuable information source. Besides vehicles centred works, there are several more general ones dealing with distance estimation [7], [45]. Both articles relate to depth or distance estimation, respectively. In the case of the second one, the distance is useful information to orient in 3D space, in particular, to navigate. A camera on a drone provides supplementary knowledge about the environment for the purpose of much stable flight and landing.

Another analogous system could be provided with a drone itself, manufactures often include better or worse tracking systems to capture improved aerial footage. The DJI is a world-leading manufacturer of these small aerial vehicles, further in this chapter the data from DJI drone is handled as well. The early solutions used a GPS beacon worn by a tracked subject, often it was the drone controller itself. GPS information supplies unprecedented accuracy but lacks any intelligence needed for obstacle avoidance. Nowadays, the tracking uses both the visual sensors and GPS information to keep itself focused. It usually works well with a range of popular objects like persons, bikers, cars, or others. This conception mimics, to some extent, the desired intended application or at least its initial steps. However, the so far briefly described combination of the two purposes for intuitive flying action, described solution from DJI is called ActiveTrack system [12].

The next available sources are examples of partial problems solutions which could be the most influential for the proposed system below. Mainly, the independent projects show ways to track either people or other objects in the image, not necessarily taken from a higher viewpoint. They use a variety of object detection methods and run it efficiently on the hardware, namely Raspberry Pi small single-board computer. Moreover, an accelerator device regularly supports the limited performance of such a compact board. The complex task of the object detector inference is offloaded to this dedicated device [17][41]. There is undoubted performance improvement using the accelerator chip overrunning the whole detection network just on CPU. These found blog posts generally create a comparative idea that tracking would be possible even with lower performance hardware. Complementary to this, in [21] for the sake of strengthening Raspberry Pi's performance, the remote machine learning API can be utilised to get an inference of an object detector network. When the API is accessed over a common URL request, the detection is as simple as it could get. An input image is attached to the request, and the response would necessarily be the detected entities. Similar to a locally ran detector, it returns individual bounding boxes, encoded in a standard JSON format.

In terms of the vision accelerator units, there are several affordable devices available on the market, usually in the shape of a USB stick: Colar[2] USB Accelerator by Google, PLAI[3] PLUG USB artificial intelligence accelerator chip by Gyrfalcon Technology, Jetson Nano[4] a pocket-size board module by NVIDIA. And the list is completed by perhaps the

[2] https://coral.ai/products/accelerator

[3] People Learning Artificial Intelligence, https://www.gyrfalcontech.ai/solutions/plai-plug/

[4] https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-nano/

most popular USB device Intel Neural Compute Stick[5], covered below in Section 3.2.2. All have a vision processing unit capable of accelerating machine vision algorithms, for instance, convolutional neural networks which are the backbones of the advanced object detection methods. Secondly, the object detection as a service is again offered by multiple API provides: The Machine Learning API[6] by NanoNets, Vision AI[7] by Google Cloud, Watson Visual Recognition[8] by IBM, Amazon Rekognition[9]. Even though, this approach is only stated as an alternative for the overall picture, it is not considered any further in this solution.

3.2 Platform – The chosen hardware

An idea for the whole work is to create a flexible and lightweight positions tracking system for aerial footage. Requirements for such a system are that it should be compact enough and yet still powerful to handle advanced computer vision tasks. As mentioned in the previous section, there are a few proven setups for object detection on the card size computers. These tiny single-boards could be attached to an unmanned aerial vehicle (UAV) or, as it has been already mentioned, commonly known as a *drone*. An application intended for this supplementary hardware payload should be able to perform the inference of a deep neural network and transmit the results over for further processing. The well established and advised combination to use for the case is Raspberry Pi 4 small computer with Intel Neural Compute Stick 2 accelerator unit, both shown in Figure 3.1. The supervisor of this thesis lent the recommended computational devices in order to test the final solution.

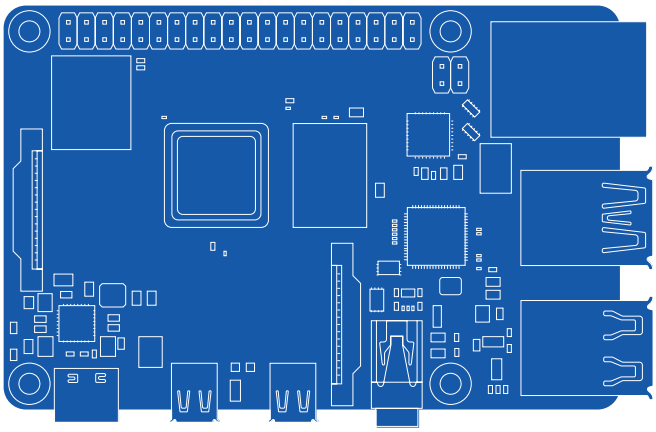
Although, the final solution for tracking people is written in Python programming language as a generic multiplatform application, only dependant on the specific constraints where it is essential. Therefore, extending the application to a different type of accelerator can be quickly done by adapting the specific single method implementation, which was responsible for utilising the previously used unit.

Finally, the second part of the application receives locations and ensures a user can freely review them. It performs a visualisation of trajectories based on the obtained relative coordinates. This is the brief definition of the targeted platform, and more details follow.

3.2.1 Raspberry Pi 4

Raspberry Pi is a small single-board computer which is dedicated for educative purposes in schools. Despite that, it is quite popular for prototyping and research work or even used in robotics. All the technical details in this section are sourced from [34], the latest Raspberry Pi 4 family, which is the primary testing device for the work here, particularly Model B with 4 GB of RAM, and an insertable 64 GB Micro-SD card as internal storage. The board can be managed by the officially supported operating system (OS) called Raspbian [35]. The guides and setting up of the computer is straightforward as there is a focus on teaching and opening up the process to the less technical public. The 4th generation obviously includes wireless connection, after the initial configuration of a Wi-Fi adapter, the development could

[5] https://ark.intel.com/content/www/us/en/ark/products/140109/intel-neural-compute-stick-2.html
[6] https://nanonets.com/
[7] https://cloud.google.com/vision
[8] https://www.ibm.com/cloud/watson-visual-recognition
[9] https://aws.amazon.com/rekognition/



(a) Raspberry Pi 4 [48]

(b) Intel Neural Compute Stick 2 [20]

Figure 3.1: Illustrations of Raspberry Pi 4 (schematic blueprint) and Intel Neural Compute Stick 2, both devices are displayed with their realistic dimensions, of total width 85 mm and 72.5 mm, respectively.

continue remotely. The OS is based on a Debian Linux distribution, and besides the minor modifications, the system can be operated the same way as any other Linux system. The community has grown over the previous board generations and offers necessary support. Raspberry 4 embeds a quad-core ARM Cortex-A72 processor and can be powered from a battery pack or a drone internal power circuit via a USB-C port. In terms of power, the inserted Intel Neural Compute Stick has to be taken into consideration, and a needed current should be supplied.

3.2.2 Intel Neural Compute Stick

Intel Neural Compute Stick (INCS), also known as Movidius Neural Computing Stick, is a low-cost USB stick that has inside a Myriad vision processing unit (VPU). These kind of processors are a reasonably new concept of microprocessors which can accelerate a various computer vision task, often seen in the robotics applications. The Intel Movidius VPU is as well embedded in many smart devices on the market for automated analysis and a better understanding of the real world around us. INCS itself is enhancing the capabilities of a regular main processor from the host computer. The stick and its VPU is optimised for the models' inferencing, for instance, the convolutional neural networks. In general, it can be an alternative to a cloud vision computing service.

There are two generations of INCS and the second one from 2018 accelerates the proposed solution later on. The INCS 2 unit can be powered directly from a USB 3 port of the host device. The VPU is utilised for the interface through Intel's OpenVINO Toolkit

which is introduced the next section. A pre-trained model can be loaded to the chip for inferencing either from the targeted Raspberry Pi or any laptop. The toolkit helps to relieve the user of a hardware specific configuration and instead let them focus on the application.

3.2.3 OpenVINO

OpenVINO is a set of tools to orchestrate the whole developing and deploying process of vision-oriented solutions on Intel's supported hardware. The toolkit is a set of versatile software to complete any image or video related project. It can not only help with INCS 2, as it was already stated but, the tools are built for much more hardware options, the kit supports Intel's FPGA initiative, Intel Graphics or even their traditional CPUs. The deep learning or machine learning frameworks and formats are widely supported, namely TensorFlow[10], Caffe[11], ONNX[12] the exchange format and others. The vital aspect is though support for Raspbian, the toolkit includes guidelines on how to install and configure everything on a resource-constrained device such this. Through the experience, the guidance is sufficient but required a little amount of time.

3.3 Inputs – The necessary data for tracking

The objective in this part is to answer a question of what data it takes to track people in the drone footage. The details in the section are based on the data samples provided by the supervisor of this work, notwithstanding the ideas might be applied to any source with a similar range and flavour of data.

Generally speaking, two separate data types include what is necessary for providing the application results. Many times discussed and analysed sensor is a camera, it gives the rich data which demand a high level of processing in order to extract the knowledge of it. Then, a more state describing data is essential too; it defines the angles of the camera, its position and other miscellaneous yet still crucial information. Practical examples of the available dataset well support all this and give better insides for each part. The used drone for data collection was an affordable mini drone DJI Spark[13]. Despite that small size, it is capable of capturing high-quality video and carries multiple sensors for intelligent flight control, moreover it features a mechanical gimbal as well. Despite this, the following analyses are demonstrated on this type of device, but nonetheless, a considerable amount of the principals would be the same for other UAVs, especially the ones manufactured by DJI. The less portable parts are the descriptions of gathering the pieces of data from the specific logs of the drone. Whereas the structure of information and its scope is comparable to the other DJI's drones has not been determined, but one can assume a high level of similarity within the same brand.

The drone has its own controller, which is connected to a smartphone and the operation is administered by installed DJI GO[14] application. The application is responsible for video recording and flight tracking. A flight log can be extracted for each drone's take-off, the recorded video samples and the entire flight log file specifically. The flights, during which the samples were captured, are described further on. The flight journal incorporates assorted

[10] https://www.tensorflow.org/
[11] https://caffe.berkeleyvision.org/
[12] https://onnx.ai/
[13] https://www.dji.com/spark
[14] https://www.dji.com/goapp

details about every possible drone component, with the overall entries recorded several times per second. In summary, the extensive list of drone's states has to be filtered for the essential and the most accurate information which is then used for precise people tracking. This section deals with where the data can be found and how are they structured, it extends Section 2.3 and gives a pragmatic view of the needed measurements.

3.3.1 Camera

A camera is an excellent sensor to capture the environment around us. Nowadays, it is an important source of information for countless applications. The camera attached to a drone is the main source for people tracking in the proposed solution here too. Its image data is crucial for object detection and subsequently for re-identification of previously detected objects. As a result of the past research, covered in Chapter 2, primarily Section 2.4, the pixels in the image can be well interpreted and understood by machines. The individual frames of video are processed to obtain the positions of objects, persons, in the image coordinates. As mentioned, the coordinates define the bounding boxes around objects, to clarify, all this is done just with image data alone. Up to this point, the input for the analysis is a camera stream, and the quality aspects are the high resolution and sharpness of the frames.

Next, additional insides about the camera are required in order to convert the object's positions from the pixel domain to their real locations relative to the camera placement. The sensors, among other things, responsible for intelligent flight assisting of a drone, are used to define accurate camera position and rotation. The telemetry data reflects the drone's and also effectively the camera's physical positioning as well, more on that in the following block. In contrast, not only the position information is necessary, but the camera parameters as well. Unlike the movement of the camera, which is constantly changing as it is attached to the vehicle, this information is associated with the camera itself and remains constant during the entire flight. The essential values for distance estimates evaluation are focal length and field of view of the used camera lens, then the camera sensor size. These specifications are based on the camera type; therefore, it should be quite straightforward to find them the related datasheets. For the unknown camera, the selected properties can be set by a calibration process [6] and derived from the images of the calibration chessboard. Another aspect is the image resolution, which is set from the image frame, and there is no need to include that in the parameters. All the found values are passed to the proposed application by a configuration file at the solution start.

According to DJI Spark Specification sheet [47], the mounted camera uses 1/2.3" CMOS sensor, with dimensions of 6.17 mm $\times$ 4.55 mm. The lens has a field of view angle of 81.9°, and its focal length is 25 mm. An example of the configuration file is shown in Appendix A.2.

3.3.2 Telemetry – Camera positioning

The proposed solution tries to achieve people tracking in a video captured from a drone which implies that the camera is regularly moving. In comparison to a static camera, it is demanded to monitor this movement, and accordingly, the system should be able to adapt to new geometry caused by position changing. Secondly, the positions related to a previous camera's locations should again be taken into consideration the new state for always displaying the relevant information. Thus, the requisite information is longitude and latitude from the radio-navigation system and an azimuth measured by a compass. This defines the location of the camera and which direction it is looking.

For improving the distance estimation of the captured objects, the accurate position based method from Section 2.2.2 requires an altitude of the camera as well as its tilt towards the ground or pitch. The camera should be in a stable position so that the readings from the sensors are as accurate as possible. The drone's camera is normally mounted to the vehicle with a gimbal pivot. For the best recording capabilities, the gimbal can be mechanically controlled. This gives the camera a much-needed image stabilisation and reports the momentary camera rotations to the final application. If the gimbal supports a yaw axis rotation, the final azimuth has to consider this too as the camera's direction is the needed one. To summarise, the positioning data is dynamic during the recording process; hence, it should be streamed together with the frames.

In terms of DJI Spark drone, all this information is listed in the log file and available through DJI MOBILE SDK[15] API too. The desired Spark data includes [47]: a longitude and a latitude from GPS or GLONASS systems, an altitude measured by vision positioning system up to 8 metres then the GPS data is used, a drone's yaw rotation from the compass, a gimbal's pitch rotation value, speed in each of the axes. The altitude measurement accuracy is ± 0.1 m for the vision system, and ± 0.5 m for GPS.

3.4 Additional software – The frameworks and libraries

With an idea of not trying to reinvent the wheel, this section gives a comprehensive overview of all the convenient software solutions which are utilised for the final work implementation. The foundation to every stated choice here was the implementation programming language, it was decided at the beginning of the project to use Python language as this developing environment is broadly used in academia and is well suited for the computer vision projects. The extra benefit is that I have been already familiar with this technology due to its versatility and used it for several small assignments in the past. The support for OpenCV visual data processing library is a matter of course among with an excellent backing for convolutional neural networks frameworks, all briefly discussed in the next few paragraphs. As well as OpenVINO offers its well-optimised APIs for Python, besides the industry standard for fast computations the C++ language. The complete list of every used software with their particular versions is summarised in Appendix B.

Video processing and analysing

So far, the previous sections summarised the hardware requirements and available data, yet another essential point is the assessment of feasible libraries to assemble the final application prototype. It is a good practice knowing which parts need to be worked on and which have been already figured out and are well implemented that they can be purely incorporated in the design.

OpenCV[30] is an open source computer vision and machine learning library. It is cross-platform too with great support for multiple programming languages. The library offers many ready to use algorithms for image and video processing, has convenient interfaces to work with a video stream. Then, the transformations of stream's individual frames, pre-processing, or basically, any needed standard operation upon image data are included. The data representation in Python's OpenCV module is covered by another package, NumPy[16]

[15] https://developer.dji.com/mobile-sdk/
[16] https://numpy.org/

which is typically used for scientific computing. The package supplies additional high-performance array operations.

The OpenCV library from version 3.3 has got its own deep learning inference module the DNN[17]. It supports pre-trained networks from various frameworks and considerably accelerates the forward pass, even can employ a VPU of an accelerator or other chips which offers parallel execution. The process aims to be as simple as possible, the interface allows to load and run the selected network. However, inference outputs demand extra post-processing, especially in the object detection sphere. The model support, to a great extent, overlaps with what is supported in the once covered OpenVINO. The final pool of implemented models is determined by ease of use and availability of the pre-trained model, and secondly by its inference support on the INCS 2 accelerator.

In general, the solution is using the pre-trained models which are proven to work well with resource-restricted hardware. It focuses namely on YOLO models as they have a great ratio of speed and accuracy, then experiments next with SSD models. All are single-pass models with computational complexity as low as possible. The extensive list of all used models is in the experimenting chapter where the models are compared and discussed. As for the designing part, the most important aspect is a fact that models are available and perform well with used hardware. Moreover, the solution architecture should count and be prepared to work with different methods and should also allow their smooth switching.

Exposing and visualisation of the results

The results are sequences of relative locations or coordinates that form trajectories which are updated continuously. Therefore, the overall requirements for visualisation library were flexibility of updates, ease of use, and ideally rendering into a web browser. Bokeh [5] is fulfilling all of these with a good performance and also adding interactive manipulation with the resulting paths. It includes APIs for data handling and management, which was an extra benefit too. The library offers good tutorials, even for the more advance use-cases which show numerous plotting options and high level of customisation. For completeness, there are also the countless other alternatives such as Pycairo[18], Matplotlib[19], or even the OpenCV among its many other functionalities offers the methods to visualise data.

Bokeh serves a web page that contains the resulting visualisation created by the library's interfaces which made the graphs accessible from anywhere after proper configuration. The results could be reshaped for a better view or exported with the built-in tooling, it handles all the zooming and moving at the client's browser. The work with datasets and colour palettes are supplementing the broad range of features.

The solution expects to transmit data from a drone side program for further processing to the second program. This could be achieved by exposing the information through RESTful [11] service at the drone side computer. Python's minimalistic web framework Flask [13] conducts exactly that, it quickly creates a simple service which serves the data. A service client can request the drone's system and obtain the resources by using the predefined addresses.

[17] Deep Neural Network Module, https://github.com/opencv/opencv/wiki/Deep-Learning-in-OpenCV
[18] https://pycairo.readthedocs.io/
[19] https://matplotlib.org/

Miscellaneous packages and libraries

Python environment comes with a large standard library of different modules, extending the language by common data structures, algorithms, maths operations, built-in types, time utilities, parallelism interfaces, operating system services and much other functionality the applications might require. Only a few relevant ones are briefly described in this text, their functionality usually covers much more than the number of details stated here. The Python Standard Library[20] is well documented, and all further information could be sourced from the referenced documentation pages.

The application deals with re-identification of the people in time; therefore, the identity has to be represented in a certain unique way. The standard implements UUIDs [50] (Universally Unique Identifier) which can represent the temporary identifications for detected objects. These ids are afterwards replaced by previously recognised persons. In the same way, a logging facility [27] is available from the standard, track event and statistics across the application run. This helps to monitor the implemented parts of the work and gives an inside look into it. Mainly, how the individual submodules or threads communicate and count how much data they exchange, for example, how many persons were detected in the given frame. The format of logging messages contains the elements in this order: *creation time*, *logging level*, *name of the local thread which constructed the message*, *description of the message*.

The communication with the REST data resources is handled by the Requests [39] library. This enables the consumer program to create and send HTTP requests in order to obtain the necessary data. The API is elegant, simple and as obvious as possible, in agreement with its documentation.

3.5 Solution design

The fundamental details and technologies, which the solution is going to be built upon, were all resolved earlier to frame its constraints. By using all the knowledge from these requirements analyses, the solution design proposal is trying to meet the goal of tracking people in the video from a movable camera. This has led to a gradual partitioning of all problems into smaller and smaller fragments which continuously was revealing the solution contours. Finally, the idea is to have two separate programs, one responsible for gathering and processing the inputs, and the second for collection what was prepared and answering the users calls while this architecture tries to accomplish the defined specification of the project assessment.

The presented design here is a foundation for implementing the application, it gives a general outline of the problems which have to be answered in code. It considers the hardware parts and the ways how to employ them into the system, what can run where. The CPU of the used Raspberry Pi board enables concurrent execution so, independent problems can be parallelised. A significant portion of the workload is passed to the mentioned accelerator unit. The second, server part, allows user to view the trajectories of monitored individuals. The implementation then reflects these designing choices which are a blueprint for the realisation, described later on.

[20] https://docs.python.org/3/library/index.html

3.5.1 Software architecture

The software architecture gets into the depth of the tracking people task for its better understanding, resulting consideration can also help you avoid or identify points of failure in the system. Consequently, the architecture blocks should represent the isolated components for which the solution is well known or straightforward to implement from square one.

The two main programs definition has been already mentioned, the *On-vehicle part* or application should run autonomously by using a small computer as a host. The video stream and necessary telemetry measurements flow into the system, then the application should transform the inputs into useful results. This is done by incorporating the object detector, the detection has been identified as the most complicated procedure; hence, it is given to the INCS 2 accelerator. The results from that detector inference are transferred back and passed to a re-identification method where all the people, who appeared in the past frames, are recognised. Alternatively, when the person is unidentified, a new record is made instead for the profiles database. Thus far, the outcome is the bounding boxes of recognised people in the current image with their identifiers. The ids provide essential information to construct trajectories. For clarification purposes, a trajectory is a sequence of past locations and the current location of a person with a certain, uniquely assigned, identity or profile.

The detection boxes are the base for a distance estimation algorithm, and these rectangle coordinates in pixels are converted into the distance from the camera. The estimates are then interpolated into location offsets with respect to a physical position and rotation of the camera, or the drone. The outcome from this system stage is a list of the distinct position data for each person. For better insight, the whole architecture is illustrated in the diagram in Figure 3.2. Lastly discussed segment is the propagation of the list with locations to the second application where the data is cached and accessible through REST API. The server part can, at any time, request the processed output from the REST service.

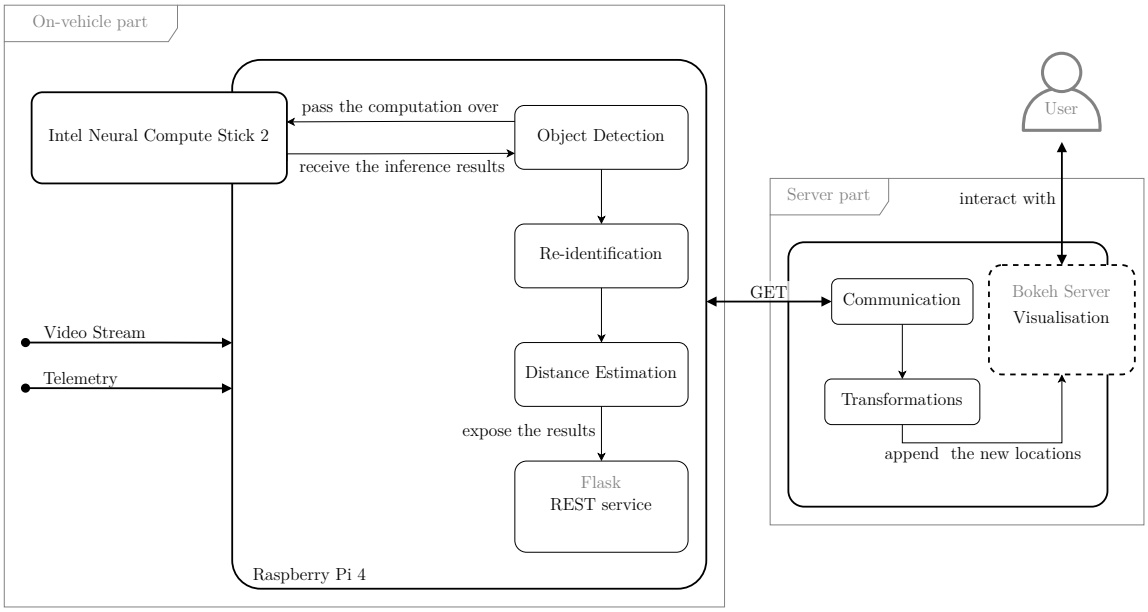


Figure 3.2: Diagram of the proposed solution architecture. Showing the two main application parts along with the relevant internal submodules and their hierarchy.

A close look to the smaller *Server part* shows that its logic is composed of a communication and processing part, and afterwards, the data is suitable for visualisation. The connection to the drone application is handled by the HTTP requests via a predefined REST resource addresses. The used method for the request should be *GET*, according to the best practises. This way, the application can receive the data and apply the translations if required. In case the camera was rotated, the previously detected locations have to be altered as the viewing situation changed and new data are relative to this new state.

The positions of people in the actual camera shot are constantly updated in the background of the server application. The underlying data of visualisation framework is streamed with updates, and correspondingly, the user's view is seamlessly changing. The user can at any time access the recorded trajectories in their web browser, the user interface is provided by the framework.

On-vehicle part internal data flow

In a close look at the more complex drone part of the system, the architecture design can give more details about which parts can be executed in parallel. By distinguishing which steps of the system can run at the same time and the flow of the data they demand, the program can complete them in separate threads as the final application runs on the multi-core processor. The proposed architecture defined these four main components: Object Detection, Re-identification, Distance Estimation, REST service which is powered by Flask framework. Furthermore, the data exchange shall be synchronised by utilising a chosen thread-safe data structure.

Initially, the frames are obtained from a camera video stream, for simplicity presume that the next camera image is always available. Only a minimal pre-processing is then performed, for instance, the object detection models might demand a certain size of the input. Hence, only the object detection module can run at this point. After the detections are completed, they can be fed to both re-identification and distance estimation algorithms simultaneously. Their results are subsequently made available by Flask service, for the whole execution refer to Figure 3.3. The service exposes the computed location estimates to a consumer application, Flask creates a new thread for each connection request, in the figure that was simplified to a one continuous thread life-cycle. The re-identification and distance estimation results are merged during the request in order to generate the response payload.

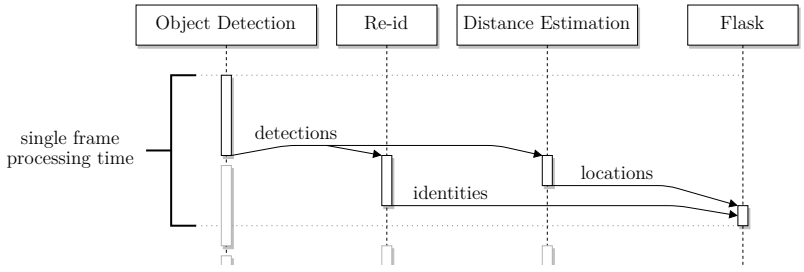


Figure 3.3: Sequence diagram of the On-vehicle application concurrent run. The arrow labels represent the different types of exchanged results from the thread workers, the Object Detection thread assumes the immediate availability of a new frame from the camera stream.

To conclude, just the two submodules can run in parallel, despite that while they are running, the object detection for the next frame can run as well. When the other modules

are done before the detection itself, the final performance should demand just on the detection speed. However, this expects that object detection is the hardest problem within all the components.

3.6 Application implementation details

The implementation details of the proposed solution are uncovered with all their necessary aspects in the following section. It includes the made implementation choices which are reflected in the code base of both parts of the application. The implementation is based on the architecture principals, which still leave a few areas untouched that can be described below.

The first step of the programming part was to configure the developing environment, install all the dependencies and set up a proper communication between Intel's OpenVINO kit and INCS stick. The process can be significantly long as OpenVINO requires to run many preparation scripts, but eventually, it can run on both Raspberry Pi and a standard desktop too. Moreover, Python runs smoothly on both platforms as well, so code is easy to move from one to another, which was exploited during development itself.

3.6.1 The On-vehicle application

The central processing application which can operate directly on the drone has four necessary subroutines. They have been already explained in the architecture. Besides this, the program adds a data provider component at the very beginning of its pipeline. The inputs power the other parts of the application; therefore, this extra abstraction layer creates a unified level of access for the data of all sources. The provider is shielding the application insides as it converts assorted information from sensors and an on-board compute of the drone.

An instance of `StreamProvider` class runs as a new thread within the program. It is initialised with the camera parameters and information source where a video source can be either live camera feed of a connected device or a file. On the other hand, the telemetry is sourced from a recorded flight log file which format is a standard *comma-separated values* or CSV file. All the developing and testing were accomplished with a video file and a CSV file combination. Moreover, the web camera feed was used too but only for the object detection verification.

Next, all the providers accept a particular implementation of the method which they later utilise in the same manner the stream class uses the telemetry data source. That is achieved by a polymorphism, a regular property of Python as an object oriented language paradigm representative. From the architecture section, each provider has its thread which executes the supplied method whenever the new inputs are available. For example, the object detection is handled by `DetectionProvider`, which is given a specific instance of `ObjectDetector` class where it implements one of the specific detectors such as YOLO, SSD, or others. Similarly, `ReIdProvider` and `DistanceProvider` are initialised with their set of methods. This way, the application can be parametrised, and the methods can be swapped easier, which makes experimenting more straightforward process too. Threads are independent and responsible for their tasks, though they have to communicate and exchange the inner intermediate values. The data structure for synchronous data advancing between the tasks is Python's *queue* module, particularly its synchronised `Queue` class. The detailed comparison of used algorithms is made in the next chapter, Chapter 4, where the methods

are confronted with each and evaluated. The threads output their statistics through logging messages so the level of verbosity can also be tweaked.

The application is a standard command line application where common start-up arguments define the methods and options. Its thorough usage guide is given in Appendix A.1. The partial results of the object detection and the re-identification stages can be on-demand visualised within the processed image. The next main stage of distance estimation represents the final results, and the second separate part program visualises them. Although the visualisation module can be used directly in the on-vehicle application as well, all the results of the stages are shown in Figure 3.4.

(a) detections (b) re-id (c) locations

Figure 3.4: An internal visualisation of the individual application stages. The *locations* stage is just an illustration of the real text data which this particular stage outputs.

The featured methods have their limitations too, and they should be considered during interpretation of the results. The found locations are still just estimates and should be treated that way. As in the example figure, a miniature person at the top was not recognised, and the YOLO detector also shows the boxes are not fully aligned with the objects, persons. Accordingly, a trajectory says instead where its owner is heading and at what pace than what is their exact location at a given time. The results confirm the restrictions which are closely summarised for each used method in Chapter 2.

The data providers are another essential point of the implementation, as the provided input data comes from third systems. The provider should be adapted or supplied for each drone and possibly camera too if that is the case. The implementation defines data objects which unite and simplify the work with drones measurements and camera features. The responsible classes from *settings* are `Camera`, `Gimbal`, `Position` and `Misc` which handles additional data of all sorts. Besides storing the data, the classes offer methods to compare or query the knowledge based on the stored information. A good example is the positions object which can return a distance between two positions in metres. These data objects are then wrapped into a telemetry class that besides the convenience purposes, offers the way to decide whether or not the image is stable. Hence, when the drone in moving from one stationary position to another the image is often undesirably rotated or unstable. The set tolerance set the maximal speed in all axes to half metre per second when the readings are out of the boundaries the frame is not processed.

3.6.2 REST endpoints specification

The next distinguishable part of the system is the communication. Final estimates should be transmitted to the server part for further processing, namely for aggregation and visualisation. The REST service creates a link between these two programs. The on-vehicle application exposes the results at the set route of the used Flask server, `/v1/locations/update`. The service is accessible at port 5000 of the host in the default configuration. The generated response is

```
[
  {
    "position": {
      "lat": 49.217218,
      "long": 16.610695,
      "altitude": 10.4,
      "bearing": 73.6
    },
    "estimates": {
      "0a1bf670-1e15-42ea-999d-fdbf76ee5c15": [0.52, 11.99],
      "f4169b7b-e9a0-4680-b113-c12b5c36b1bf": [3.23, 14.29],
      "..."
    }
  },
  "..."
]
```

where it composes of a list of recently processed frames results. The *estimates* element contains the locations of detected people in the actual frame. The locations object's elements are the personal keys, and the position offsets pairs, then the syntax is as follows `"id-uuid": ["delta x", "delta y"]`. The offsets are represented relative to the *position* element of individual items. The position element is the GPS location of the camera at the time when the image was taken. The information transmitted is also altitude and more importantly, the camera's bearing, which allows the server app to connect together the trajectories even when the camera rotates. Additionally, the whole history of all processed frames is accessible from `/v1/locations`.

3.6.3 The server application

The detached part of the proposed solution is responsible for regularly requesting the data from the drone and offers it to the user. When the batch response is received, each item's distance estimate is appended to the respective current trajectory. Its selection is driven by the unique ids of individual points or profiles. Moreover, the camera's location determines if the view should change before the points inclusion. Therefore, when the vehicle or camera rotates, the past locations compensate for this motion, so the newly appended points are added to the right coordinate system.

The user can access the trajectories plot at port 5006 via HTTP, as the default configuration of Bokeh framework. An example of the final results is illustrated in Figure 3.5, the page includes standard tooling for resizing and dragging the canvas. The trajectories are represented as lines with the detected locations highlighted as circles.

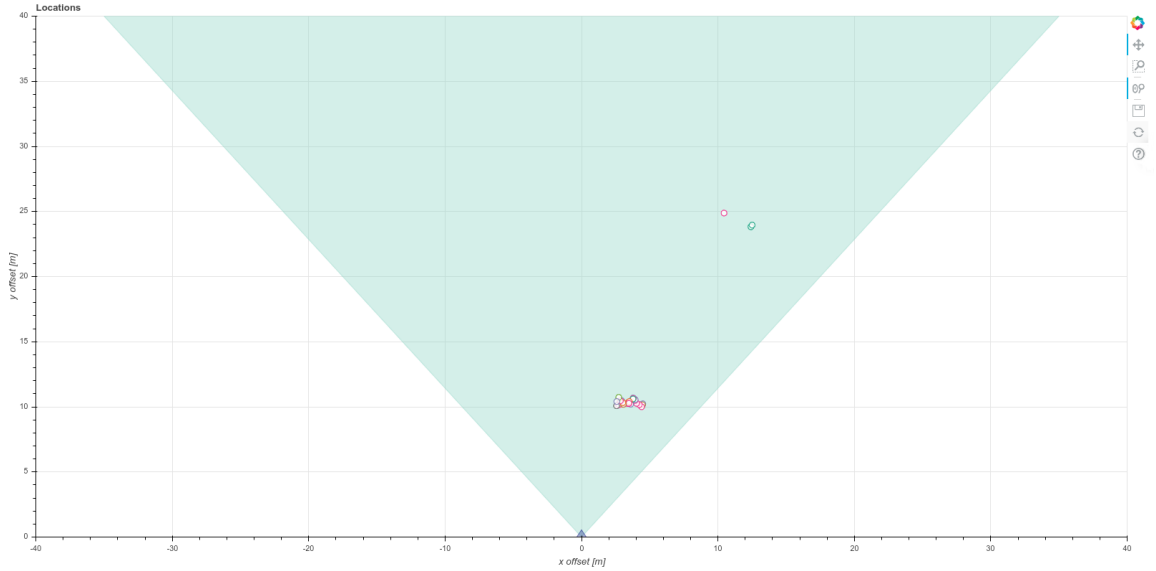


Figure 3.5: UI visualisation – Bokeh UI. The light green triangle represents the camera's field of view, its positions is the origin of the used coordinate system. The circles are detected person's locations.

Chapter 4

Experiments

As stated in the chapter above, the implementation is capable to easily swap its particular methods for each of the described subproblems. The methods are covered in details in Chapter 2, and here the most significant ones are tested and tried with the testing video data. The following part of the offers a view to implementation usability and overall performance. Due to a narrow scope of available data, the experiments are limited to a dataset provided by the supervisor which the details are discussed in Section 4.2. The obtained outcomes, as well as the issues, are then summarised in the subsequent sections 4.3, and then 4.4.

4.1 Methods summary

Object detectors are relatively new models in machine learning, they are only possible with recent discoveries in the field. This was mostly thanks to the growing accuracy of general-purpose classifiers and the novel training datasets. The solution which is needed for this project must work in real-time; therefore, good accuracy is as important as a processing speed – processing frame rate. New specialised hardware accelerators are now available which can deliver desired processing performance and power efficiency for the application even on the compact computing boards.

Most of the models, mentioned above, are not strictly trained on aerial images; hence, they might be less precise on video taken from a drone. Nevertheless, the image processing part of the system should be able to adequately answer two main questions: is a person present in the image and where is it within the image. Moreover, the system should be able to resolve if the person was seen before. Methods for re-identification might be more invariant to the different camera view, which can make looking for the right one an easier. However, imperfections in these parts can lead to poor overall localisation. Only one camera usage can significantly reduce the need to include a sophisticated method of re-id in the system. These methods are commonly used in multi-camera non overlapping setups where it is hard to id a person just based on the multiple independent images. Person's features are extracted and compared along with his or her movement constraints if possible. This might result in the usage of much simpler algorithms which can offer the corresponding accuracy for this very case. Furthermore, the methods like that come with a lower computational cost that is always advantageous.

Speaking of re-identification, two main approaches are tried. The first is the more advanced feature extraction using an OpenCV's KAZE key-points detector. The detector

can find and extract the necessary image description, which is later used to match the next image. However, the method did not perform that well and is not used in the final comparison. Instead, a truly simple histograms comparison is incorporated to get the matches of person's profiles. The histogram composed of 32 bins per each colour channel is extracted as a profile template and stored. Then, the matching process uses the histograms correction comparison, with a threshold of 0.7. When the correlation coefficient is below the threshold, a new profile is created for a person as non of the templates in the database get a strong level of similarity.

As for the object detectors, three main ones are tested, the SSD detector based on MobileNet, and then YOLO second and third iteration. The YOLO models are used as their *tiny* modifications for better performance speed on Raspberry Pi and INCS 2.

The last part of the used algorithms are the ones for distance estimation, both proposed methods in Section 2.2 are implemented and tested. The size based estimation assumes the average height of $1,74$[1], the average for both genders adult height. The position based method is strictly defined by method equations.

4.2 Dataset & data sources

The provided data from a DJI Spark recording session are generally good quality footage. The video covers a couple of different situations which are also considered during the design and implementation of the solution. A quite standard positional capturing is included with a bit of wind that is causing a minimal shaking. Overall the available data were taken under very good conditions, and this might mean the provided telemetry is quite accurate. Although there might be a slight de-synchronisation while using the offline data, the provided CSV logs are not synced to the video feed, and this process was taken manually.

Drone positions vary from approximately 5 to 10 metres of height from the ground, the GPS signal is strong. The scene is mostly clear, but the samples also include a section where people are partially occluded by a treetop. The camera moves around to get static views of the standing, or the walking supernumeraries. The shots also contain a few where the camera rotates while maintaining its location on a map. This particular case is essential to test all features of the proposed system.

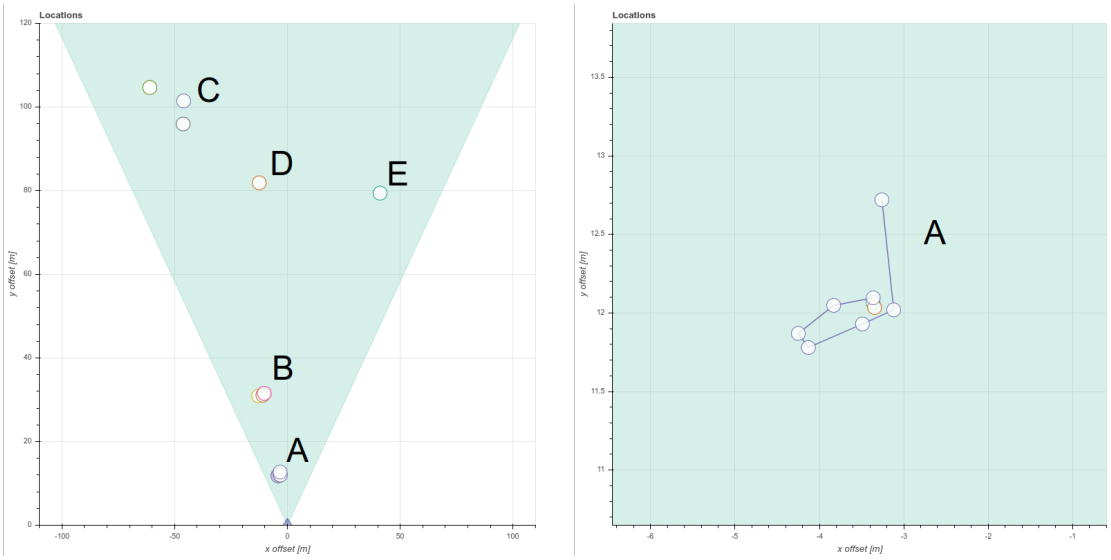
4.3 Experiments and testing

The test environment was the Raspberry Pi 4 with connected INCS 2, and the board was connected to a local network. The server application ran on the separate device within the same network and was collecting the data of the individual test runs. One of the obtained results is described in Figure 4.1 and is analysed in further details below.

The figure shows the tracking solution results for the Tiny YOLO detector, the histogram comparison method, and the pixel position based distance estimation. The results in 4.1a are captured for about 3 seconds from a static camera position[2]. As the real locations of the persons are not available; however, the environmental clues give an estimate where they stood. From the map, the assessed distance from the camera of the individuals: Person A stood on a visible pathway intersection, so its distance from the camera is about 16 m. Person B is about 25 m from the camera, then Person C – 35 m, Person D – 30 m,

[1] https://ourworldindata.org/human-height

[2] The camera position was $49.2172369N$, $16.6108169E$ and bearing of $84.4°$.



(a) detected locations overview

(b) zoom-in view for person A

(c) real view at the scene

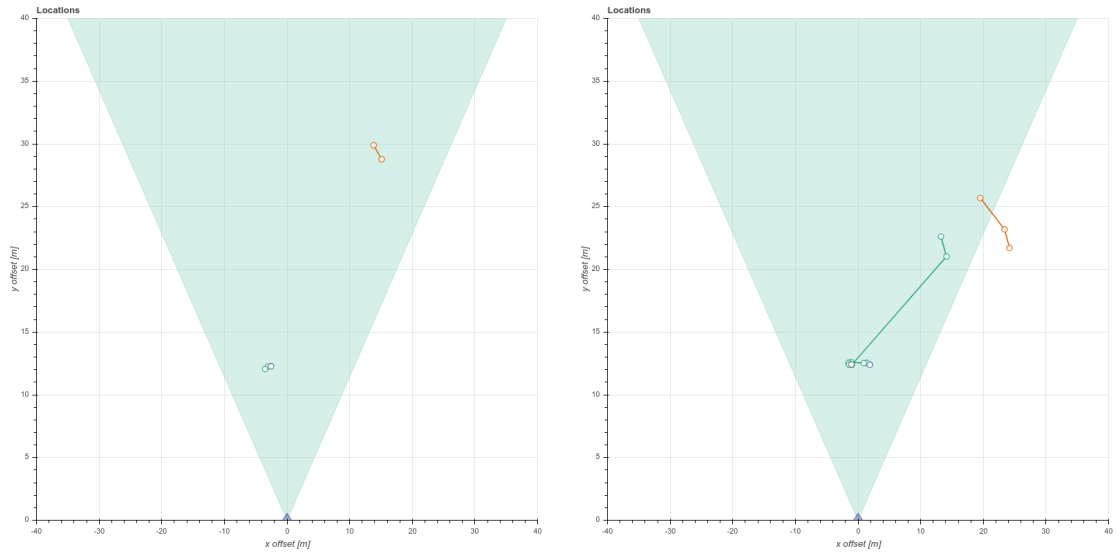
Figure 4.1: Testing result example, including the summary view with the persons' locations, a zoomed-in for person A, and the real view of the captured scene, people remained mostly stationary.

Person E – again about 35 m. The person in the white top in 4.1c was not recognised by the detector, or the detection was not confident enough. Figure 4.1c adds the perspective of how even the stationary person and get the locations estimates within an about metre radius as the bounding boxes fit the person differently every time. That can be triggered not just by moving from place to place but also by simple posture changes.

The system's estimate for persons A and B are quite accurate, namely 12 m and 28 m, respectively. The results are reflecting the true locations, especially in the lower part of the image. Then, the error increases as the persons are detected in the middle and the top half of the frame, it is caused by the uneven and slightly uphill terrain. Moreover, the objects get smaller, and the detected bounding box misalignment has a much significant impact. That can also explain the widespread of person C's locations. Accordingly, the camera tilt is 90 degrees with the vanishing line of flat surface in the middle of the image.

4.4 Overall system performance

The results coming off the system shall deal at least with the minor changes in the view. The rotation compensating capability of the server application was tested as well, see Figure 4.2. Prior to the rotation, the test captured just two people from the video sample for these particular results. The tracing methods were the same as in the previous test case, besides the detector used the SSD model this time. From the testing, this object detector had more precise bounding boxes around detected persons but struggled with detecting all of them. The video sample composed of a short 4-5 s footage where two persons walking and one stayed almost stationary, others are not detected. The camera was rotated for almost 30 degrees anticlockwise (locations are rotated clockwise). As can be seen, the detections in the bottom of diagram 4.2b are pushed towards the centre of the camera's view. The orange trajectory is also transformed accordingly. In 4.2b, the third person was recognised; however, its identity was matched with someone's else. The mismatch in the histograms comparison resulted in the unwanted connection line between the two, see the long trajectory line.



(a) before the rotation occur

(b) after the rotation

Figure 4.2: Results before and after the camera rotated while capturing the analysed video.

An interesting observation was made during the implementation, where all the frames of video were processed. The frame rate was about 30 FPS which led to analysing almost identical frames one after another. This way, the trajectory was constructed of many points really close to each other, and the results were not satisfying. Additionally, these many points slowed down the UI rendering, the final implementation is, therefore, filtering them, and showing indistinguishable locations as one for the same identity profile. That might also help for static people in the image as they could be visualised as a single point. The analysis is done on the server part when the data is already processed.

The tests partially confirmed that the models still struggle with small-scale objects, especially on the less powerful hardware. With the objective to address the issue, the frame can be split into several pieces, and they can be processed on their own. This can

increase an absolute number of pixels per person when the image is downsampled, a major disadvantage of the approach is the needed processing power. On the contrary, the split image creates a problem with a person at the cutting edge.

One of the comparing cases was dealing with difference in size based and position based distance estimation methods. The results of size based method were way too inaccurate, resulting in up to 150 m distance from the camera for the same test case as in 4.1. The variety in bounding box size and the actual person size is way out of the proportion for this method to be used, especially with YOLO detector as the result of this experiment. On the other hand, the implementation for size based estimate does not compensate for the altitude, but the difference of the results is so significant that this would just make a small change for the final estimates. The drone altitude for the experiment case was about 10 metres which would make a slight difference.

Chapter 5

Conclusion

The implemented solution is capable of processing the video on board of the drone, using the compact computer with the accelerator. Then, the results are presented to a user in the form of a graph which includes the data points representing locations. The points are position estimates of the individuals found in the video. They are gradually matched with their past self, and the points finally build the trajectories. The system is tested with YOLO and SSD object detectors models which give mixed results as used. Therefore, there I see, together with the re-identification, room for improvements. The absolute error in location is caused by the inaccurate persons' detections and their misaligned bounding boxes, combined with the insufficiency of distance estimation methods. The distance evaluation creates substantial demands on the correctness of the detection.

Next, a source of the input data was a DJI Spark's recordings and telemetry. However, the proposed application can be further easily extended to accept other data sources of particular drones while the core of the methods remains intact. Tests on these video recordings included the complete pipeline verification and getting the final locations of the people in the video.

The implemented system got satisfactory results when the conditions were ideal, though when the input is pushing the system and its used methods to their limits, the decrease of accuracy is more than significant. The next work shall focus even more on the models for object detection. Maybe, it can consider the more advanced re-identification methods with neural networks which can run a separate INCS 2 unit too. The models can be adapted for the aerial video input by taking advantage of transfer learning of the current state-of-the-art detection networks. Moreover, the object detectors are multi-purpose detectors, inferring all sorts of objects, the idea of post-training on just people samples can also better adapt the model for the solution.

The drone industry is growing rapidly with its business potential too, and number of new applications will start to be noticeable in our daily life. The proposed solution also omitted the increasing capabilities of drone sensors, in particular, the extra multi-cameras mounted to them for better usability. The manufacturers themselves are incorporating object tracing to the products as the collision avoidance systems. Additional cameras can give a better sense of depth to the system.

Bibliography

[1] Adaimi, G., Kreiss, S. and Alahi, A. Rethinking Person Re-Identification with Confidence. *CoRR*. 2019, abs/1906.04692. Available at: http://arxiv.org/abs/1906.04692.

[2] Aircraft Rotations – Body Axes. *The Beginner's Guide to Aeronautics* [online]. [cit. 2020-05-10]. Available at: https://www.grc.nasa.gov/WWW/K-12/airplane/rotations.html.

[3] Bewley, A., Ge, Z., Ott, L., Ramos, F. and Upcroft, B. Simple online and realtime tracking. In: *2016 IEEE International Conference on Image Processing (ICIP)*. 2016, p. 3464–3468. DOI: 10.1109/ICIP.2016.7533003.

[4] Bochkovskiy, A., Wang, C.-Y. and Liao, H.-Y. M. YOLOv4: Optimal Speed and Accuracy of Object Detection. *ArXiv*. Apr 2020, abs/2004.10934.

[5] Bokeh Development Team. *Bokeh: Python library for interactive visualization* [online]. 2020 [cit. 2020-05-28]. Available at: https://bokeh.org/.

[6] *Camera Calibration* [online]. OpenCV [cit. 2020-05-28]. Available at: https://docs.opencv.org/master/dc/dbb/tutorial_py_calibration.html.

[7] Croon, G. C. H. E. de. Monocular distance estimation with optical flow maneuvers and efference copies: a stability-based strategy. *Bioinspiration & Biomimetics*. IOP Publishing. Jan 2016, vol. 11, no. 1, p. 016004. DOI: 10.1088/1748-3190/11/1/016004. Available at: https://doi.org/10.1088%2F1748-3190%2F11%2F1%2F016004.

[8] Documentation for individual models. *Keras Documentation* [online]. [cit. 2020-01-18]. Available at: https://keras.io/applications/.

[9] Dumoulin, V. and Visin, F. A guide to convolution arithmetic for deep learning. *ArXiv*. 2016, abs/1603.07285.

[10] Everingham, M., Eslami, S. M. A., Gool, L. V., Williams, C. K. I., Winn, J. et al. The Pascal Visual Object Classes Challenge: A Retrospective. *International Journal of Computer Vision*. 2015, vol. 111, no. 1, p. 98–136. DOI: 10.1007/s11263-014-0733-5. ISSN 0920-5691. Available at: http://link.springer.com/10.1007/s11263-014-0733-5.

[11] Fielding, R. T. *Architectural Styles and the Design of Network-Based Software Architectures*. 2000. Dissertation. University of California, Irvine. Available at: https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm.

[12] Film Like a Pro: DJI Drone ActiveTrack. *DJI Guides* [online]. 2017 [cit. 2020-05-14]. Available at: https://store.dji.com/guides/film-like-a-pro-with-activetrack/.

[13] Flask. [online]. Pallets Projects [cit. 2020-05-29]. Available at: https://palletsprojects.com/p/flask/.

[14] Girshick, R. Fast R-CNN. In: *2015 IEEE International Conference on Computer Vision (ICCV)*. Dec 2015, p. 1440–1448. DOI: 10.1109/ICCV.2015.169. ISSN 2380-7504.

[15] Girshick, R., Donahue, J., Darrell, T. and Malik, J. Region-Based Convolutional Networks for Accurate Object Detection and Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Jan 2016, vol. 38, no. 1, p. 142–158. DOI: 10.1109/TPAMI.2015.2437384. ISSN 1939-3539.

[16] Goodfellow, I., Bengio, Y. and Courville, A. *Deep Learning*. MIT Press, 2016. Available at: http://www.deeplearningbook.org.

[17] Győri, A. Turning a Raspberry Pi 3B+ into a powerful object recognition edge server with Intel Movidius NCS2. *Towards Data Science* [online], 19. May 2019 [cit. 2020-05-23]. Available at: https://towardsdatascience.com/turning-a-raspberry-pi-3b-into-an-object-recognition-server-with-intel-movidius-ncs2-8dcfebebb2d6.

[18] Haykin, S. *Neural networks : a comprehesive foundation*. 2nd ed. Upper Saddle River,: Prentice-Hall, 1999. ISBN 0-13-273350-1.

[19] He, K., Gkioxari, G., Dollár, P. and Girshick, R. Mask R-CNN. In: *2017 IEEE International Conference on Computer Vision (ICCV)*. Oct 2017, p. 2980–2988. DOI: 10.1109/ICCV.2017.322. ISSN 2380-7504.

[20] *Intel Neural Compute Stick 2* [online]. [cit. 2020-05-25]. Figure. Available at: https://www.bgp4.com/wp-content/uploads/2019/01/dims-neural-compute-stick.jpeg.

[21] Jain, S. How to add Person Tracking to a Drone using Deep Learning and NanoNets. *Nanonets Blog* [online]. 2018 [cit. 2020-05-23]. Available at: https://nanonets.com/blog/how-to-add-person-tracking-to-a-drone-using-deep-learning-and-nanonets/.

[22] Kim, G. and Cho, J.-S. Vision-based vehicle detection and inter-vehicle distance estimation for driver alarm system. *Optical Review*. 2012, vol. 19, no. 6, p. 388–393. DOI: 10.1007/s10043-012-0063-1. ISSN 1340-6000. Available at: http://link.springer.com/10.1007/s10043-012-0063-1.

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

[24] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E. et al. Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*. Dec 1989, vol. 1, no. 4, p. 541–551. DOI: 10.1162/neco.1989.1.4.541. ISSN 0899-7667.

[25] Lee, S., Hayes, M. H. and Paik, J. Distance estimation using a single computational camera with dual off-axis color filtered apertures. *Opt. Express*. OSA. Oct 2013, vol. 21, no. 20, p. 23116–23129. DOI: 10.1364/OE.21.023116. Available at: http://www.opticsexpress.org/abstract.cfm?URI=oe-21-20-23116.

[26] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S. E. et al. SSD: Single Shot MultiBox Detector. *CoRR*. 2015, abs/1512.02325. Available at: http://arxiv.org/abs/1512.02325.

[27] Logging facility for Python. *Generic Operating System Services* [online]. The Python Standard Library [cit. 2020-05-29]. Available at: https://docs.python.org/3/library/logging.html.

[28] Lu, L., Shin, Y., Su, Y. and Karniadakis, G. Dying ReLU and Initialization: Theory and Numerical Examples. Mar 2019. Available at: http://arxiv.org/abs/1903.06733v2.

[29] Luo, H., Jiang, W., Zhang, X., Fan, X., Qian, J. et al. AlignedReID++: Dynamically matching local information for person re-identification. *Pattern Recognition*. 2019, vol. 94, p. 53 – 61. DOI: https://doi.org/10.1016/j.patcog.2019.05.028. ISSN 0031-3203.

[30] *Open Source Computer Vision Library* [online]. OpenCV team [cit. 2020-05-29]. Available at: https://opencv.org/about/.

[31] *Pinhole camera model — Wikipedia, The Free Encyclopedia* [online]. Wikipedia contributors. 2020 [cit. 2020-05-12]. Figures. Available at: https://en.wikipedia.org/w/index.php?title=Pinhole_camera_model&oldid=942392219.

[32] Qi, S. H., Li, J., Sun, Z. P., Zhang, J. T. and Sun, Y. Distance Estimation of Monocular Based on Vehicle Pose Information. *Journal of Physics: Conference Series*. IOP Publishing. Feb 2019, vol. 1168, p. 032040. DOI: 10.1088/1742-6596/1168/3/032040. Available at: https://doi.org/10.1088%2F1742-6596%2F1168%2F3%2F032040.

[33] Ramachandran, P., Zoph, B. and Le, Q. V. Searching for Activation Functions. *CoRR*. 2017, abs/1710.05941. Available at: http://arxiv.org/abs/1710.05941.

[34] Raspberry Pi 4 Computer Model B. *Raspberry Pi 4 Tech Specs* [online]. [cit. 2020-05-25]. Available at: https://static.raspberrypi.org/files/product-briefs/200206+Raspberry+Pi+4+1GB+2GB+4GB+Product+Brief+PRINT.pdf.

[35] *Raspbian* [online]. [cit. 2020-05-25]. Available at: https://www.raspberrypi.org/downloads/raspbian/.

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

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

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

[39] Reitz, K. Requests: HTTP for Humans. [online]. Python Software Foundation [cit. 2020-05-29]. Available at: https://requests.readthedocs.io/.

[40] Ren, S., He, K., Girshick, R. and Sun, J. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Jun 2017, vol. 39, no. 6, p. 1137–1149. DOI: 10.1109/TPAMI.2016.2577031. ISSN 1939-3539.

[41] Rosebrock, A. Real-time object detection on the Raspberry Pi with the Movidius NCS. *PyImageSearch* [online], 19. Feb 2018 [cit. 2020-05-23]. Available at: https://www.pyimagesearch.com/2018/02/19/real-time-object-detection-on-the-raspberry-pi-with-the-movidius-ncs/.

[42] Russell, S. and Norvig, P. *Artificial Intelligence: A Modern Approach*. 3rd ed. Prentice Hall, 2010.

[43] Ružička, M. and Mašek, P. Real Time Object Tracking Based on Computer Vision. *Mechatronics 2013: Recent Technological and Scientific Advances*. Jan 2014, p. 591–598. DOI: 10.1007/978-3-319-02294-9-75.

[44] Saha, S. A Comprehensive Guide to Convolutional Neural Networks. *Towards Data Science* [online]. [cit. 2020-01-17]. Available at: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

[45] Shang-Hong Lai, Chang-Wu Fu and Shyang Chang. A generalized depth estimation algorithm with a single image. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 1992, vol. 14, no. 4, p. 405–411.

[46] Sivaraman, S. and Trivedi, M. M. Looking at Vehicles on the Road: A Survey of Vision-Based Vehicle Detection, Tracking, and Behavior Analysis. *IEEE Transactions on Intelligent Transportation Systems*. Dec 2013, vol. 14, no. 4, p. 1773–1795. DOI: 10.1109/TITS.2013.2266661. ISSN 1558-0016.

[47] *Spark User Manual* [online]. 2017-10 [cit. 2020-05-28]. 53–54 p. V1.6. Available at: https://dl.djicdn.com/downloads/Spark/Spark_User_Manual_v1.6_en.pdf.

[48] Specifications. *Raspberry Pi 4 Tech Specs* [online]. [cit. 2020-05-25]. Figure. Available at: https://www.raspberrypi.org/products/raspberry-pi-4-model-b/specifications/.

[49] *TensorFlow Conv2D Layers: A Practical Guide* [online]. [cit. 2020-01-06]. Available at: https://missinglink.ai/guides/tensorflow/tensorflow-conv2d-layers-practical-guide/.

[50] UUID objects according to RFC 4122. *Internet Protocols and Support* [online]. The Python Standard Library [cit. 2020-05-29]. Available at: https://docs.python.org/3/library/uuid.html.

[51] Veen, F. The neural network zoo. *THE ASIMOV INSTITUTE* [online]. [cit. 2020-01-05]. Available at: https://www.asimovinstitute.org/neural-network-zoo/.

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

[53] Zhang, X., Luo, H., Fan, X., Xiang, W., Sun, Y. et al. AlignedReID: Surpassing Human-Level Performance in Person Re-Identification. *CoRR*. 2017, abs/1711.08184. Available at: http://arxiv.org/abs/1711.08184.

Appendix A

Usage

A.1 Run instructions

Quick run manual:

```
usage: person_tracking.py [-h] -o DETECTOR -m MODEL
                          [-l LABELS] [-r LOG] [-v VIDEO] -s CONFIG
                          [-c CONFIDENCE] [-d DEVICE]

optional arguments:
  -h, --help            show this help message and exit
  -o, --detector        detector model choice
  -m MODEL, --model MODEL
                        path to pre-trained model, directory should
                        contain all other necessary model files
  -l LABELS, --labels LABELS
                        path to labels file, each label should be on
                        a new line
  -r LOG, --log LOG     path to csv log file of flight record
  -v VIDEO, --video VIDEO
                        path to video recording of the flight
                        which should be analysed
  -s CONFIG, --config CONFIG
                        path to used camera configuration file
  -c CONFIDENCE, --confidence CONFIDENCE
                        minimum probability to filter weak detections
  -d DEVICE, --device DEVICE
                        device to run inference on (typically "MYRIAD" or
                        "CPU")
```

```
usage: viewer.py [-h] -l HOST [-p PORT]

optional arguments:
  -h, --help            show this help message and exit
  -l HOST, --host HOST  set server part host name
  -p PORT, --port PORT  set server part port number
```

A.2 Example of a configuration file

- Configuration file for FC1102 camera type, which is mounted to DJI Spark drone:

```
[camera]
type = FC1102
sensor_width = 6.17          # in mm
sensor_height = 4.55         # in mm
focal_length = 25            # in mm
fov = 81.9                   # in degrees
```

Appendix B

Used libraries summary table

- Python 3.6
- Bokeh 2.0.1
- Flask 1.1.2
- NumPy 1.18.2
- OpenCV 4.2.0
- OpenVINO 2020.1.023
- requests 2.23.0
- simplejson 3.13.2
- tornado 5.1.1

OS versions:

- Raspbian GNU/Linux 10 (buster)
- Ubuntu 18.04.4 LTS