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 VIDEO CAPTURED FROM A DRONE

SLEDOVÁNÍ OSOB VE VIDEU Z DRONU

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR **Bc. JAKUB LUKÁČ**
AUTOR PRÁCE

SUPERVISOR **Ing. TOMÁŠ GOLDMANN**
VEDOUCÍ PRÁCE

BRNO 2021

Brno University of Technology
Faculty of Information Technology

Department of Intelligent Systems (DITS) Academic year 2020/2021

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 pedestrians in video using neural networks based object detectors.
3. Assemble a camera unit for video streaming. The camera unit should consist of USB camera and Wi-fi module.
4. Design and implement an application that should be capable of recognize people in the video frame and convert their positions to local coordinates. Perform a visualization of trajectories based on obtained coordinates. The video source for the application should be a video stream from the camera unit.
5. 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

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, 2020
Submission deadline: May 19, 2021
Approval date: November 12, 2020

Abstract

This thesis deals with the problem of determining the location of a person through their distance from camera 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 of the tracking. 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 v grafe. 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 detekovaných osôb v zábere kamery a príslušne ho prezentovať.

Keywords

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

Kľúčové slová

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

Reference

LUKÁČ, Jakub. *Tracking People in Video Captured from a Drone.* Brno, 2021. 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 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. Celý tento pre ľudí jednoduchý proces je možné len veľmi obťažne popísať formálne v podobe nejakého sekvenčného algoritmu. 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 detegovať 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 neskoršieho spracovania.

K tomu, aby bolo možné zistiť polohu osoby v obraze, teda detegovať objekt prislúchajúci zvolenej triede osôb sa využívajú metódy strojového učenia, ako napríklad sofistikované detektory objektov, modely klasifikátorov, detekcia geometrických primitív a ďalšie. Pod povrchom týchto algoritmov sa najčastejšie skrývajú postupy 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 rutinné činnosti akou je aj detekcia a klasifikácia objektov.

Následne pre získanie vzdialenosti nájdenej osoby od kamery sú potrebné vstupy zahŕňajúce informácie o polohe kamery a jej vlastnostiach, ako napríklad konkrétne natočenie a uhol snímania. Všetky tieto premenné parametre udávajú lepšiu predstavu toho, kam kamera smeruje pri vyhotovovaní záznamu. Poloha musí byť založená na použití správnych údajov opisujúcich stav zariadenia, spolu so súčinnosťou už spomínanej konkrétnej pozície objektu vo videu. Ideálny výber z videa 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 robotiky, kde sa určuje rozstup 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 obdobné metódy sa testujú a skúmajú ako vhodné riešenia asistentov vodičov, alebo sa získané údaje z jednej kamery 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 následne odhad založený 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.

Dôležitou časťou sledovania osôb sú už spomínané spôsoby ich nájdenia vo videu a *zaškatuľkovania*. Inšpirácia pre dnešné algoritmy vychádzala z pôvodnej práce, ktorá navrhla kombináciu odhadu potenciálnych 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 naraz ako celok a jeho predpovede o polohe objektov 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 objektov rôznych tried. Táto architektúra je pri svojej vysokej rýchlosti stále dostatočne presná a umožňuje operácie v takmer reálnom čase aj pri obmedzenom množstve výpočtového výkonu. Týmto spôsobom dosahuje výsledky porovnateľné s oveľa zložitejšími modelmi.

Architektúra výsledného systému riešenia definuje tri hlavné časti, a to modul pre poskytnutie potrebných vstupných dát, konkrétne videa a údajov zo senzorov. Ďalej modul zodpovedný za samotné sledovanie osôb, ktorý implementuje a realizuje potrebné metódy spracovania obrazu vrátane odhadu vzdialenosti, a ktorý môže byť umiestnený priamo na malé bezpilotné vozidlo. Takáto aplikačná časť beží na kompaktnom jednodoskovom 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 z kamerového modulu. Spracovanie prebieha začlenením detektora objektov, pričom táto úloha patrí medzi najnáročnejšie, a preto je jej výpočet delegovaný práve čipu akcelerátora INCS 2. Výsledky sa zo zariadenia presunú späť a tvoria vstup pre identifikáciu osôb v čase, kde sa rozpoznajú ľudia z predošlých snímok. V prípade, keď bola osoba neidentifikovaná, nebola skôr videná alebo systém si nie je istý či ju už videl, je jej namiesto toho vytvorená nová identita. Informácie z detekcie taktiež 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. Takto spracované informácie je možné získať a presunúť do poslednej časti navrhnutého systému, všetko za pomoci definovaného REST protokolu. Prezentačná vrstva získa tieto údaje, agreguje ich a aplikuje prípadne posledné úpravy podľa potreby. Poskytuje taktiež možnosť užívateľovi zobraziť dané nájdené trajektórie osôb v prehľadom interaktívnom grafe vo webovom prehliadači. Trajektórie sú tvorené jednotlivými odhadmi vzdialenosti spojené v rámci príslušného profilu rozpoznanej osoby.

Implementované riešenie je schopné spracovať video priamo na kompaktnom hardvéri pričom dosahuje mieru spracovania v malých jednotkách snímok za sekundu. 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 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

May 17, 2021

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 Gathering data from various sensors 8
2.4 Neural networks in image processing 11
2.5 Object Detection – object localisation and classification 17
2.6 Re-identification of person in frames 24

3 System Proposal and Implementation **26**
3.1 Similar work tackling these problems 26
3.2 Platform – The chosen hardware 28
3.3 Inputs – The necessary data for tracking 30
3.4 Additional software – The frameworks and libraries 32
3.5 Solution design 34
3.6 System implementation details 37

4 Experiments and methods evaluation **42**
4.1 Dataset & data sources 43
4.2 Experiments and testing 44

5 Conclusion **52**

Bibliography **54**

A Usage **60**
A.1 Run instructions 60
A.2 Example of a configuration file 61

B Used libraries summary table **62**

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 and decrease the computing power required. Understanding 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 as our primary visual cortex was trained and developed by nature for many years. Our vision is undoubtedly superior to methods which were implemented in computers in the past.

The trend might be changing right now, and computers are getting better and better at the task, enabling 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 mark 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 sectors 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. Optionally, they would adjust the boundary boxes to suit the identified objects, as it seems, it was quite a complex system. In the following chapters, all different improvements to this design are discussed. Especially recently, the more popular single-pass solutions are available and 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. Re-id allows following a subject in the video. 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 a vital 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 such imagery, the proposed system's final output 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 determined by drone sensors.

The drone industry is fast growing and offers fascinating 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, this popularity opens up new opportunities for research as vehicles get cheaper and more and more data is available at a minimal cost, all of which played a major role for me to look into this project. The thesis is trying to build upon those existing systems and comes with a reliable mechanism that solves the task of positioning a person. The resulting application's primary goal is to generate tracking metadata from detected individuals in the video.

There are several complications with tracking objects and estimating their locations, as outlined above. Hence, addressing solutions are split into individual sections in which each part of the final system is analysed from a different perspective. In Chapter 2, distance estimation methods, along with necessary image processing techniques, are explained. It includes a brief introduction to neural networks and their subclass, convolution neural networks, which are the leader in visual analysis in computers nowadays. Following covered areas are algorithms for measuring how far is the object from the camera, Section 2.1. Methods of defining the object's position within the image are 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, and used hardware description. The chapter deals with both the proposed design and its implementation as well. Subsequently, the last chapter covers all the experimenting work which is 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 have to be processed using all the fundamental methods and ideas presented in this chapter. This includes using information about a camera's location, rotation 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 detecting and identifying people in the image.

Computer vision has been improving by leaps and bounds since it could solve 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 [26]. The start of this type 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. The chapter then covers and summarises object detectors, followed by methods that explain re-identification (Re-ID) of a 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 that 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 constantly 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.

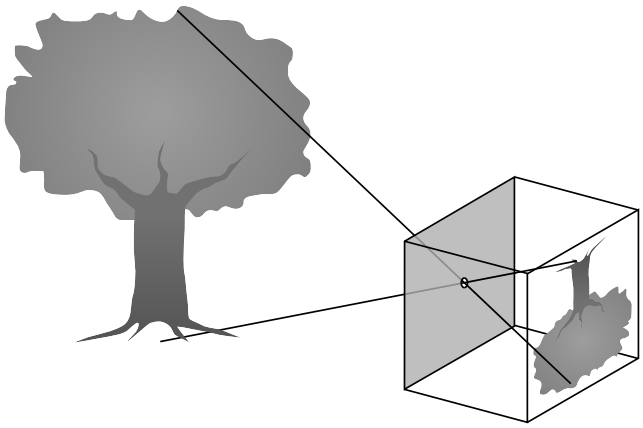


Figure 2.1: A diagram of a pinhole camera. [38].

Generally speaking, a camera as a distance measurement sensor is rather inaccurate and a processing heavy solution. On the other hand, it comes with a low cost for the sensor itself. Robots can be equipped with cameras to avoid obstacles, interact with the right objects appropriately, or navigate in the environment.

The measurements based on camera sensors are likewise favoured within the automotive industry and 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. Collision avoidance has an enormous impact on society which encourage new developments in the field. The eventually presented methods are often tested and researched as the demand for a cost-effective solution for these assistants. Data collected from a single camera is used to estimate a range between two road users, and this knowledge is generalised for determining the object's distance from the camera.

2.2 Distance estimation in a single-camera system

The available vehicle distance estimation methods derive from elemental optics principles 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 [25], which provides the basis for distance estimation methods following in this section. The two methods have each a set of advantages one over the other, as well as having specific restrictions for the object itself or its surroundings.

2.2.1 Size based distance estimation

The size-based approach uses the camera's imaging properties, 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 too, and 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 restrictions for the object of interest.

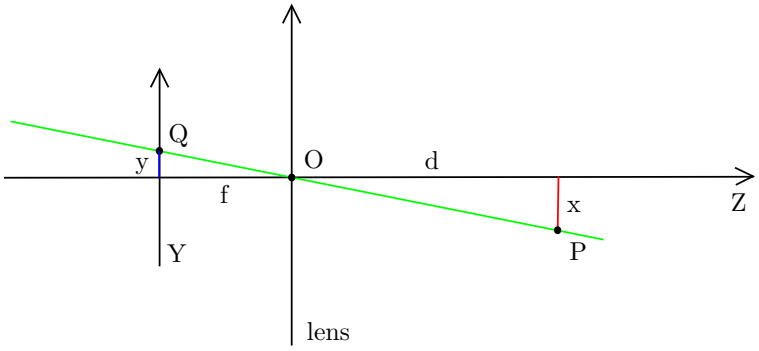


Figure 2.2: Size based distance estimation geometry [38].

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 an actual 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 hand, 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 an image height in pixels, s is a camera sensor height. Both are constant for the particular camera recording. x_p is the height of an object in pixels. Therefore, a way to detect and identify the object in the pixel image representation is required. In Section 2.5, the necessary methods called object detectors are presented. The detectors are based on neural networks, and output the information about the object position in the image needed for the distance estimation.

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. This estimation procedure's main restriction is that it assumes the objects are located on a flat surface. The significant point in the image is where the object's contact with the ground is. 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 relative to the ground, 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. Equation (2.4) assumes that the camera's lateral axis, or pitch axis, is parallel with the flat ground 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

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

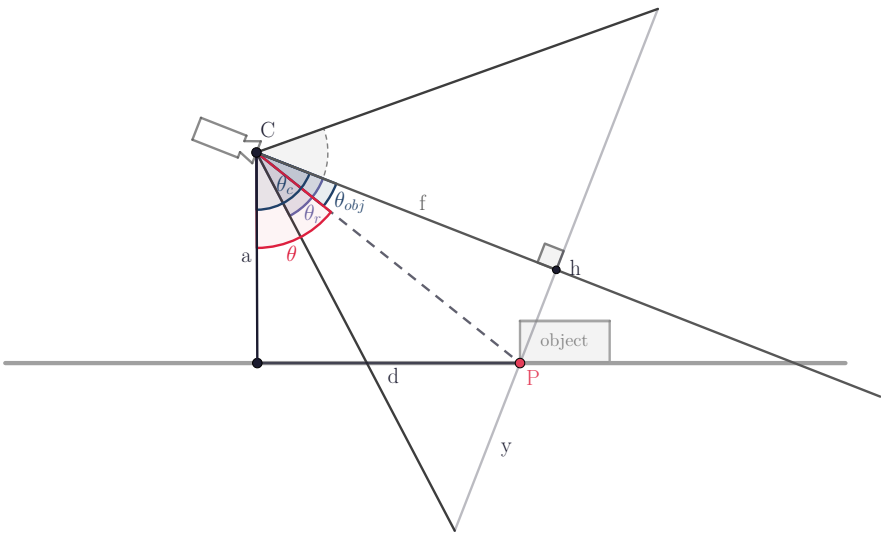


Figure 2.3: Position based distance estimation geometry, created with GeoGebra and based on the figure in [25].

The need for the flat capturing area is essential to the described method. Equally important additional information necessary for the distance estimation can be supplied by various localisation indicators present at the drone's central computer.

2.2.3 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 [29], 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. It used the advanced computational method to enhance the usual optical system. Such works prove that the problem of distance estimation is a critical issue in many applied areas.

Besides the contemporary academic ideas and stand-alone advanced sensors, there are also complete solutions ready to output tracking distance. One example of these new optics and sensors is a package brought to the market by company DJI and its commercial industry drone Matrice 300. The vehicle's new payload[1] option combines multiple sensors, and is capable of determining the distance and effectively the location of the object directly recorded in real-time. The range of the instrument like that can exceed 1 km with reasonable accuracy. However, carrying these extra payloads can be difficult for the much smaller consumer drones, also ease of manoeuvrability and total fly time might reduce significantly.

2.3 Gathering data from various sensors

After describing diagrams and possible ways to calculate the person's location, this section covers the various sensors that supply all the relevant attributes. In addition to the main source of the information – video, the distance estimation depends on the various data. The previous sections assumed that several auxiliary values are defined. They can be available from different drone sensors and technologies embedded in almost all of these aerial vehicles today.

Intended camera placement is one attached to the aerial vehicle and is usually mounted using a gimbal which primarily allows better camera stabilisation. Additionally, the gimbal can also provide extra controllability in selected axes. The principle and terminology are based on typical aircraft rotations [2]. A drone gimbal often allows adjusting a camera in its pitch axis, while more advanced gimbals support rotations in the yaw axis as well. Together with the general localisation data, it makes a clear view of how and where the camera is facing. The following important information is internal camera properties, not only the outputting image specification but also the used lens details, focal length, sensor size, and field of view.

The camera is a dominant source of data, and its position in surroundings is likewise essential for extracting the objects' distances. Supporting sensors are the drone instruments for measuring the precise location in all three dimensions. These onboard systems primarily help to manoeuvre it in an uncharted environment. However, data is also stored as time series among the video frames for further analysis.

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

2.3.1 Positioning systems

The Global Positioning System, or GPS for short, is a well-known and widely adopted satellite-based navigation system. It is one of the many constellations that deliver geolocation along with precise time information to their users. Typical GPS communication is only one way, namely from satellites which send periodic signals to a receiver located on or near the Earth surface. The receiver demands at least four satellites with a straight line of sight for successful localisation.

The principle of such a positioning system is based on finding the distance to each visible satellite. Satellites have predictable and stable orbits. Thus, their positions are known at any point in time. They all carry a source of very accurate time, atomic clocks, and all the clocks are adequately synchronised to common time. The signal that comes from the satellite then contains a timestamp of transmission from the satellite. For simplicity, assume that the user's system is synchronised to satellites' time. When the receiver gets the signal, it can immediately determine the satellite-to-user propagation time [24]. This time value multiplied by the speed of light outputs the range R, distance to the satellite, as shown in Figure 2.4a. Hence, the receiver is located on the surface of sphere with radius R and centre in the given past position of the satellite. By collecting the same information from multiple different satellites simultaneously, the sphere surface can be diminished to a point, the user's exact location. The process is illustrated in Figures 2.4b and 2.4c, respectively.

The problem of required time synchronisation can be addressed as a separate issue. For instance, satellites transmit signals with pseudo-noise code, which the clocks use to synchronise better. Furthermore, the shown fundamentals can be used with what is called *pseudorange* as well. This range is calculated from biased information obtained by non-synchronised satellites' and receiver's clocks. All that leads to a set of linear equations that can be solved by making ranging measurements to four satellites.

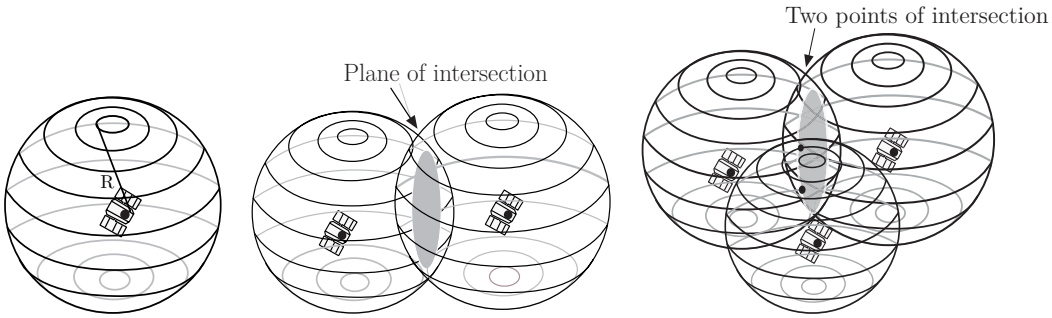


Figure 2.4: Illustration of receiving ranging signals from GPS satellites and showing the simplified steps of localisation based on increasing number of satellites [27]. (a) Receiver located on the surface of sphere in the distance R. (b) Receiver located on shaded circle's circumference. (c) Receiver located at one of two points.

By tracing the position in time, almost all receivers can derive the user's speed and travel direction. Moreover, the facing direction is also supported by an electronic compass in many GPS units for better stationary orientation. The use cases really vary, and good localisation information has major value for many different applications. Modern receivers can also combine the signal from different global navigation satellite systems, for example, from

GPS, GLONASS, or GALILEO. Thus, this created a good redundancy even in environments with a large number of obstacles.

The common GPS offset is usually noted in low ones of metres. However, this value was rather true in the past. New satellites, in combination with modern receiving solutions, claim that they can achieve an accuracy of 30 centimetres. The better performance is due to added transmission band, and nine satellites are usually visible from any place at any given time. Error found by the regular annual report based on the data from various GPS monitoring stations across the world was on average 1.09 m for one set of stations and 1.24 m for a different one [49]. These results are for horizontal positioning. Furthermore, the applied localisation method used data editing techniques for better measurements. Analysis also evaluated vertical average position errors of 2.12 m and 1.45 m, respectively, for the two sets. As the signal is emitted from the satellites strictly above the receiver, this can cause a more significant triangulation inaccuracy in the vertical coordinate. The amount of difference in the position also varies with the number of satellites. Next, the error can be caused by the different composition of the atmosphere, clock bias, or precision loss when manipulating the numbers.

2.3.2 Additional vertical localisation

Subjects locations are relative to the found camera position as it was covered above. The camera elevation from the ground is a crucial property for some distance estimation methods. Accordingly, differences in altitude can cause discrepancies in the whole computation, opposite to just minor shifts introduced by inaccuracy in the camera's latitude and longitude. In aviation, the knowledge of altitude is also fundamental, so drones offer several ways to measure it.

An internal navigation system is often composed of multiple distinct sensors. They can provide relevant elevation information based on the different stage of the flight. During the lift-off and landing, the camera proximity sensors can be given priority over others. However, GPS localisation incorporating data from an accurate barometric altimeter is preferred in mid-air. The reliability and precision of the altitude values from different sources need to be used with caution. All data readings should be taken into account in a well-defined context.

For instance, GPS usually uses elliptic standard, number WGS84, for representing Earth. This shape is somewhat oversimplified and can lead to huge offsets from the actual altitude. However, the method can provide an independent measurement at any time. The coordinate system can also be replaced by more realistic models, such as geoid. Complementary to this, relative altitude is measured from the take-off site by a set of movement sensors [15]. The error then is gradually accumulated over the whole flight.

Another way is to augment the GPS signal by additional systems, often known as differential GPS. Broadcasting ground stations are able to reduce the errors from metres to as much as magnitude of centimetres. Systems are generally used in commercial flying or marine logistic. The subsequent elevation measuring instrument is a pressure altimeter that uses changes in air pressure. The calculated altitude relates to average atmospheric pressure at sea level as a baseline. Barometric pressure can be interpreted in the context of the current weather forecast for better results.

In addition, ground distance sensors based on ultrasound or lasers are generally available and suitable for collision avoidance. Their range is valid only to a certain distance from the ground, within tens metres. The information is, in many cases, more precise than any other

system already mentioned. The modern drone system encapsulates as many as possible of the described sensors. It adaptively chooses the elevation source and improves the flying experience.

2.3.3 Movement and exact camera orientation information

All the sensors and information mentioned in this section can be conveniently gathered from UAVs onboard computers. These computational units incorporate plenty of precise instruments in order to support vehicles flight and simplify its control. The data are widely used also for self-protection of the vehicle, or the autonomous actions such as obstacles avoidance, return home, safe landing, assisted hovering in mid-air.

The distance estimation methods require, among other already described attributes, the exact camera position. The camera's gimbal can provide its actuators states, usually calibrated by one of the limit positions. That translates to the current tilt of the mounted camera. Next, the movement of the drone is constantly monitored with an accelerometer and gyroscope. These sensor readings are preprocessed within the internal Inertial Measurement Unit or IMU. The values describing acceleration and degree of rotation help maintain the position and orientation during the flight. Data is also stored and can be used for other applications. IMU can augment the data and directly calculate useful metrics for future use, such as given speed in all axes.

2.4 Neural networks in image processing

Video as a data-rich source of information needs to be processed automatically. Currently, using neural networks shows the best results to tackle this problem. A neural network is a type of machine learning model. Nowadays, its primary purpose is to understand speech, image data, and support us, humans, at various labour (medicine, science, repeating jobs). The networks' real advantage shows up when the task, to be solved, is easy for people to perform but hard for people to describe it to computers formally. 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. A computing system with the structure like that vaguely mimics the biological neural network, thus the name *neural network*. This section draws mainly from the book *Deep Learning* [18]. 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 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 [52]. The output, as shown in equation (2.7), can vary a lot. Therefore, its value is given to an

activation function, usually to reduce the output codomain. 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 will be discussed 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 [62]. The most important ones, from this work perspective, are feed-forward networks. The information flows from the front to the back, and there are no cycles nor self-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 just 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) [28] is the improvement of machine learning methods mentioned above. This particular type is primarily used for image processing and 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 preprocessing 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 manage to learn these filters with enough training examples. For instance, the layers reduce the image into a structure that is easier to process without losing features critical for getting an accurate prediction. Besides the convolutional layers, CNN also incorporates downsampling layers, called pooling layers. These kinds of layers reduce the level of details for afterwards more unequivocal prediction making. Both layers are often linked using ReLU as an activation function.

An example of underlying CNN architecture is shown in Figure 2.5. All the newly introduced layers are part of the diagram as they would be in real-used network design. The order of operations, as shown, is rather typical for CNNs.

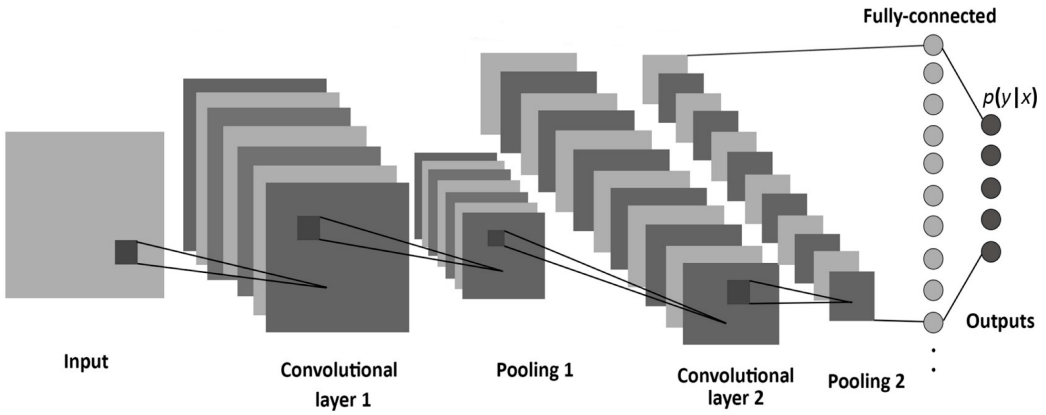


Figure 2.5: Diagram shows the fundamental architecture of the convolutional neural network [59]. It consists of convolutional layers, subsampling layers, also called pooling layers, and a fully connected feed-forward network layer at the most right, which predicts the final category for input.

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 kernel, which is a multidimensional array of parameters.

Finally, an image is a two-dimensional array then, for an input image I and a two-dimensional kernel K of size $m \times n$ convolution [18] 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 cross-correlation. It is the same as convolution but without flipping the kernel, and frequently, it is still referred to as *convolution*. A practical example of a two-dimensional 3×3 kernel operation applied to a two-dimensional array of size 5×5 is illustrated in Figure 2.6.

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

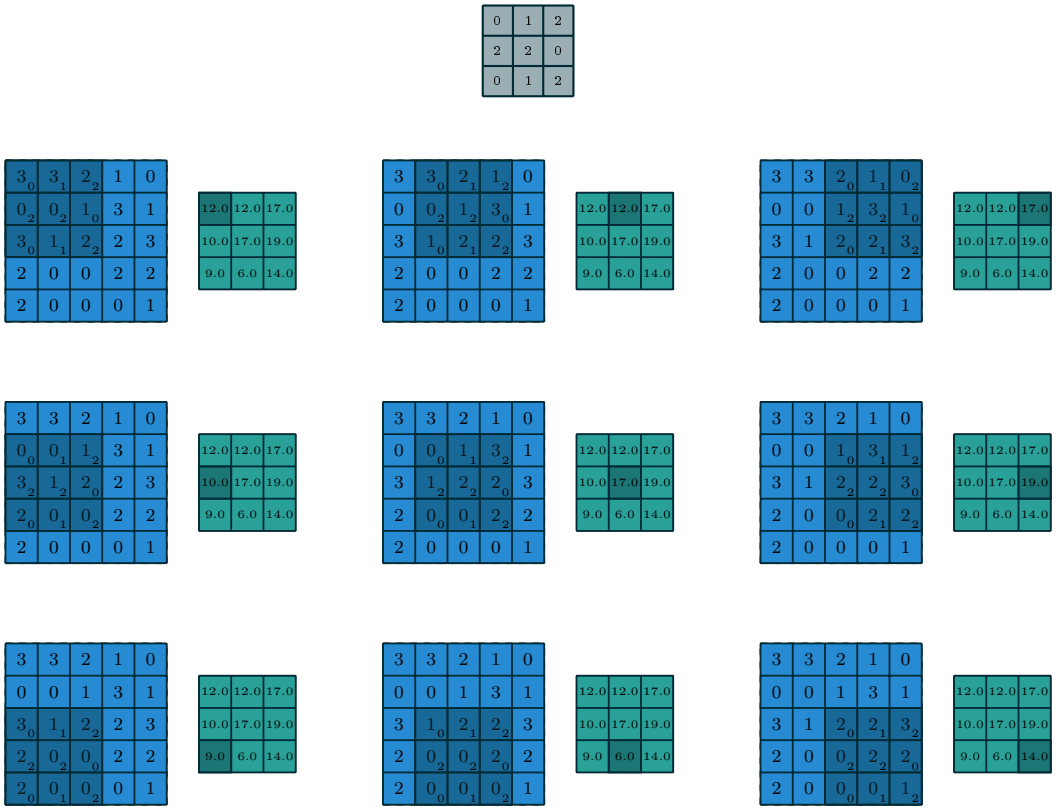


Figure 2.6: An example of a convolution operation from [10], a kernel at the top in grey, an input array in blue, and a result in green.

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 most prominent element from the set feature subregion. Besides that, *average pooling* could take the average of all elements. Analogically, outputting the sum of all elements in the subregion is called *sum pooling*. Two of the types used in CNNs are shown in Figure 2.7.

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

Figure 2.7: Practical example of different pooling operations.

A process of adjusting the pooling layers can reduce the computational power needed as dimensionality decreases for the 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 [54]. Another expected benefit of pooling is persisting the necessary dominant features in a rotational and positional invariant manner.

2.4.2 Activation functions

Activation functions [20] add non-linear transformation, enabling 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 is that the function has to be non-complex to compute. 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 function's main advantage 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 [32], 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, depending 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, and promising replacement of ReLU could be seen in *Swish* [40] 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 over time, which resulted in many popular network architectures. The network is compound of many diverse

layers sizes and incorporates 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 the following Section 2.5. The comprehensive history of CNNs includes all different architectures, and new innovative layer arrangements are frequently researched. They might be intended as general-purpose ones, or narrowly focused on 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 contains just examples of the deep learning frameworks. The applications may run the inference for each one of them using straightforward calls to give programs an extra added value for the user.

Model	Year	Top-1 acc	Top-5 acc	Size	Parameters	Depth
VGG16	2014	0.713	0.901	528 MB	138 357 544	23
VGG19	2014	0.713	0.900	549 MB	143 667 240	26
InceptionV3	2015	0.779	0.937	92 MB	23 851 784	159
ResNet50	2015	0.749	0.921	98 MB	25 636 712	-
ResNet101	2015	0.764	0.928	171 MB	44 707 176	-
ResNet152	2015	0.766	0.931	232 MB	60 419 944	-
ResNet50V2	2016	0.760	0.930	98 MB	25 613 800	-
ResNet101V2	2016	0.772	0.938	171 MB	44 675 560	-
ResNet152V2	2016	0.780	0.942	232 MB	60 380 648	-
Xception	2016	0.790	0.945	88 MB	22 910 480	126
InceptionResNetV2	2016	0.803	0.953	215 MB	55 873 736	572
DenseNet121	2016	0.750	0.923	33 MB	8 062 504	121
DenseNet169	2016	0.762	0.932	57 MB	14 307 880	169
DenseNet201	2016	0.773	0.936	80 MB	20 242 984	201
NASNetMobile	2017	0.744	0.919	23 MB	5 326 716	-
NASNetLarge	2017	**0.825**	**0.960**	343 MB	88 949 818	-
MobileNet	2017	0.704	0.895	16 MB	4 253 864	88
MobileNetV2	2018	0.713	0.901	**14 MB**	**3 538 984**	88

Table 2.1: Comparison of different CNNs architectures in terms of their size and accuracy for image classification [9]. The accuracy (acc) was measured using the ImageNet dataset.

The models are also peeled off their training envelopes, which speed up the weights setting up phase. That extra processing allowed 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. Additionally, the number of parameters, the depth or inference processing time are yet other defining factors for the CNNs practical application. The trend is to make the models with much fewer parameters perform as

good as their vast predecessor or even better. All this while researchers are still constantly improving the accuracy of the standard deep models.

2.5 Object Detection – object localisation and classification

The process of detecting an object in image or video is a fundamental computer vision problem. By solving this problem, machines must find the areas where the object is present and also identify its category. It may sound like the most basic information obtainable from imagery data, but the problem was efficiently resolved only in recent years. With the help of already mentioned deep convolutional neural networks, nowadays detectors are capable of robust and high-level feature extractions. What ultimately leads to describing which objects are part of the image and where in the image are.

Initial work in the object detection field was a detector which name is determined from its authors' names now. Viola-Jones object detector [63] was a big leap in detection when it was published in 2001. It came with a tremendous processing rate and was resistant to many possible image changes. Nevertheless, it was still built around a simple detection idea. A sliding window went through the possible places and scales to look for object occurrences in the scene – architecture composed of three main parts: integral image, Haar filters and AdaBoost classifier for final decision. The integral image was a prevalent image representation at the time, allowed efficient filtering and convolution. That reduced the complexity of using different sliding window sizes. Feature extraction was based on Haar features, selected not by the authors but by AdaBoost machine learning technique. The features were selected from randomly generated feature pools based on the specifics of a desired detected object. AdaBoost is a sophisticated feature classifier that embeds simple weak classifiers into a larger decision cascade. The combination of weak classifiers eventually leads to a solid final inference. Secondly, the classifier is used to combine the weak features, Haar filter outputs, all together into a final detection decision. The Viola-Jones detector was tuned on greyscale images and was purely focused on detecting front-on portraits. By all means, it can also be repurposed as a generic object detector. The concepts in Viola-Jones work set the object detection field's foundations, and the detector itself was the first of its kind with the possibility of real-time processing.

Histograms of Oriented Gradients (HOG) for detecting pedestrians [8] was another key study in the domain. HOG detector can be used for recognising a number of different objects despite its primary goal was pedestrian detection in a static image. The descriptor algorithm aims to discard all the irrelevant parts from the pixel data information. The detection is based on computing the HOG feature descriptors for the individual image blocks. The image is first standardised to a fixed aspect ratio, and then it is cut into a grid of cells. The individual cell is later represented by its histogram of oriented gradients. The input could also be normalised using different colour spaces or gamma correction. Notwithstanding, these transformations have only a negligible effect on the final accuracy when used.

Gradients are calculated using edge detection operators in both x and y image axes. It uses two convolutional kernels with values $[-1, 0, 1]$ of size 1×3 and 3×1, for horizontal and vertical edges. The resulting image gradients are converted to their vector form, effectively by conversion from Cartesian to polar coordinates. Each gradient is now described as two values, its magnitude and direction. Values within every image cell are sorted into histograms, thus the name *histograms of oriented gradients*. The vast pixel values space was reduced to just a few histogram bins. A representation like that is much more suited

for following up classification algorithms, such as SVM or AdaBoost. The histogram definitions can be easily altered, which makes their modifications effortless. Again, many other subsequent detector proposals benefited from these concepts.

The methods described next surpassed the Viola-Jones detector and manual HOG feature extraction methods. For instance, comparison review [36] has shown that the Viola-Jones detector scored at least 18 mAP percentage points behind all others on both evaluated benchmark data sets. That was significantly lower than CNN-based detectors presented in the test, which were in the range of around 5 percentage points between them. On the other hand, Viola-Jones performed well in real-time frame processing rate, its simplicity beating the complex multi-pass detectors by a lot. Therefore, the initial algorithms, especially those not based on neural networks, are not considered in further implementation or evaluation. The credit must be given to these studies, and their many ideas are still present in the later deep learning based detection period after 2014 [66]. CNNs dramatically automated the feature extraction process in the object detection field and became dominant.

mAP (mean Average Precision)

Mean average precision (mAP) serves as a metric to compare the accuracy of object detectors. 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 that the system can recognise. Then, average precision is gathered using IoU over some threshold. If the IoU value is above the threshold, then the prediction is considered as correct.

The state-of-the-art detectors are described in the following 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 [17] 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 these results can be seen in Figure 2.8. A simple and yet logical step could be to create a window N by M pixels and, with a specific offset, go through the input image and classify the newly created sub-images one by one. However, this approach is very inefficient. A fixed window size might also lead to a very low mAP, as the object and the window tend to be misaligned. To summarise, this was also an initial approach of the first available algorithms that integrated convolutional neural networks in detection. R-CNN was demanding much computational power to do so as a result of many redundant operations and corrections. The follow-up architectures fixed one problem after another to gain better performance. This section reviews popular methods to locate multiple objects in a single image.

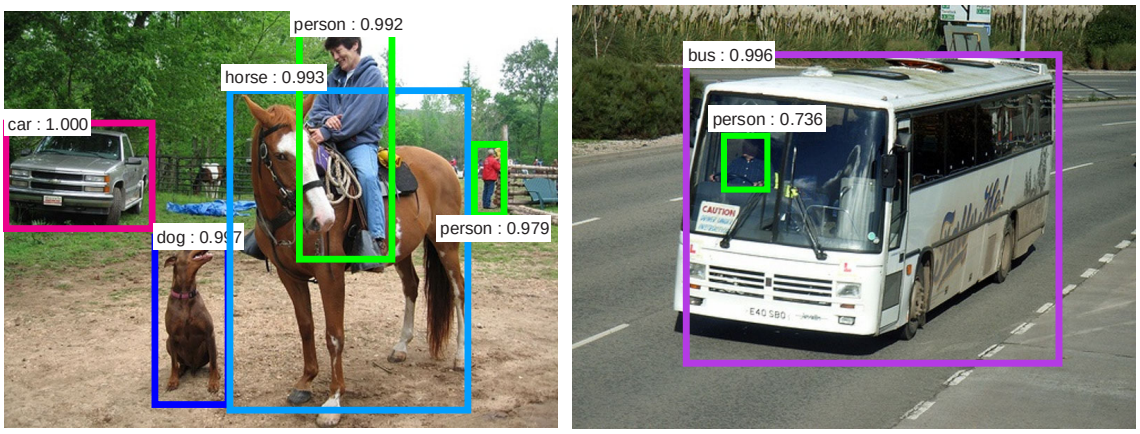


Figure 2.8: An example of output images from an object detector with various detected objects [48]. Showing bounding boxes for each detected object with its type and confidence score (from *not confident* – 0.000 to *very confident* – 1.000)

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

Object detection is finding regions with different objects in the image and classifying them as individual segments, in the same way, R-CNN [17] 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 [26] from the convolutional neural networks field.

An R-CNN detector is organised into three main modules, as shown in Figure 2.9. The first of which extracts region proposals, by using a selective search method without an obligation to know what exactly is in the actual region. The second part wraps the proposed pixels to comply with CNN restrictions and 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 better suit the objects' actual dimensions. The critical drawback of this solution is that the last two parts are executed for each proposed region. However, there are no doubts about R-CNN's genuine accuracy, with an improvement by more than 50 % relative to the previous algorithms.

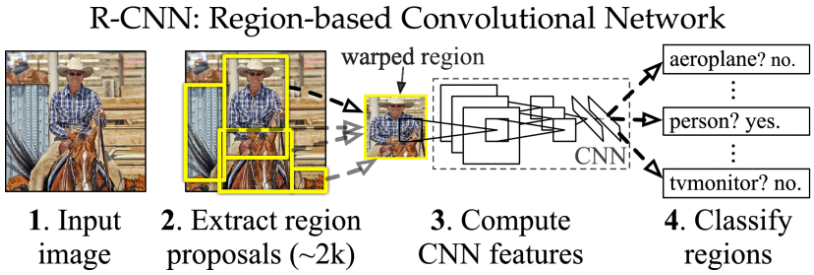


Figure 2.9: R-CNN object detection method proposed in [17], shows the outcomes of the input image processing during the detection stages.

The number of generated regions for the method is around 2000 category-independent proposals [17] for an average input image. Every one of them requires a forward pass of

the CNN, which 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. As a result, a new design of Fast R-CNN [16] 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 2.10. Specifically, for each object proposal, the RoI pooling layer extracts a fixed-length feature vector. Furthermore, the number of models is also reduced to simplify the training process and fine-tuning of each. The original SVM is replaced by a softmax classifier where their both performed equivalently. That led to more unified training, rather than having the three training stages of the first R-CNN.

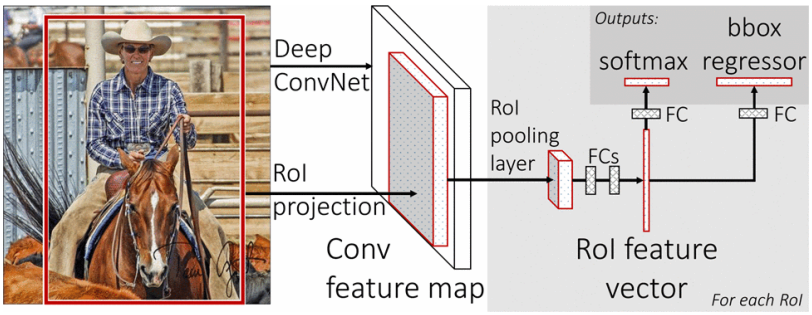


Figure 2.10: Fast R-CNN object detection method, shows the improved architecture with the shared convolutional neural network (*Conv feature map*) and a straightforward training process, proposed in [16].

Both proposals are still suggesting to use of selective search, which 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 [48]. Selective search is replaced by the already in place convolutional network, reusing it to search for region proposals. A single CNN, which helps find an object and classifies it, also enabled training only a single model. The improvement is pushing the 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. At the same time, it is proposing the bounding boxes and their scores. 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, hence these are also called anchor boxes.

2.5.2 Mask R-CNN

The previously covered methods detect an object, and the result is a rectangular bounding box, more specifically its coordinates. Mask R-CNN [21] added to the mix a complete object's binary map as a more granular result. For each pixel in the input image, the detector predicts 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. The mask is created by another convolutional network added in parallel with the original design's second stage. The extra branch predicts segmentation masks in order to separate the object from its 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 uses quantisation when downsampling a feature map causing the misalignment between regions in the input image and the extracted features. The classification is robust enough to compensate for these minor translation errors thus were not discovered sooner. 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 [30] 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 while using neural networks.

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 for different scales of feature maps as well as for their various aspect ratios. This variability 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 levels of far more complex models.

The architecture details are gathered mainly from the original paper *You Only Look Once: Unified, Real-Time Object Detection* [44]. 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 mainly a problem of how to train the model as such. Authors of YOLO came up with new methods to tackle

the dilemma. First, they had to define a structure of the single-pass output, which consists of many predictions and accordingly, their confidence score. Neural networks can output these many values without any problems, for instance, ImageNet works with predictions for hundreds of different classes. To summarise, YOLO transforms the problem of detection into a problem where an input image outputs the corresponding tensor. This tensor encodes all the possible object predictions as separate sets of values representing 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 which centres fall into 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 [11], has two bounding boxes, of which each has the 5 values, and holds 20 different class probabilities. The used grid size is 7, final tensor is $7 \times 7 \times 30$, which prompts the 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 network's training process is more critical than the further details about the structure 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 the confidence loss of a box that only contains background. The labelled images are converted to tensor representation accordingly. The suitable class is assigned to a cell that includes the centre of a particular object. The cell's box, including the object and the highest IoU, gets its confidence increased, and all other boxes decreased. The coordinates of the box, which confidence is being increased, are also adjusted to match the ground truth. This high-level description of the loss function demonstrates the ideas behind the method, and the equations describing the whole error arithmetic can be found in [44]. The network training process also 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 a high enough confidence score, then the box class is based on its cell's 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, ignoring the others. 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 slight mismatch in a large box is generally negligible, although the same shift in a small box has a much more eminent effect on the IoU metric. 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 real-time processing capabilities. Despite these facts, YOLO's authors also tested the final model on artwork images. The network outperforms other object detectors, and that 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 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* [45]. 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 better image resolution. The effect was an increase in features extraction and better overall accuracy compared to the original design, which was pre-trained with just 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 ten 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 frequent boxes' sizes and aspect ratios from the training data by the K-Means Clustering method. After all, the anchor boxes resulted in a slight decrease in detection quality. The complete list of all incremental enhancement is nicely summarised in the paper, and 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 into 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 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 changes accordingly with the image resolution. This way, the original dataset can be extended even more, it might prevent the over-fitting to some extend during a large number of training epochs while looking at the images repeatedly.

The 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 can 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 metadata includes a class label alongside the object's coordinates, then both errors are back-propagated. The practical implication could be creating an object detector that can detect, for example, dogs, but the enhanced classification marks 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 full pictures with classes alone. That was mainly possible with the defined structure of words and

hierarchical classification. The words are sorted in a tree structure which goes from the abstract root of *physical object* label to more and more specific labels down to specific leaf 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 the 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 [46] object detector was another incremental update and followed well the original message of the architecture: being extraordinarily fast and accurate. It came with just minor improvements and changes in the design that 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, it achieves an akin accuracy but much faster, authors claim about two times speeding up. The accuracy comes with a price though, as version 3 is slightly slower than its predecessor. The interesting improvement was to extend the number of classes within one bounding box, meaning multiple class confidence values are outputted for a single box. In general, the improvements helped YOLO achieve similar accuracy as Faster R-CNN and therefore, finally overcame the SSD detector as well. Nonetheless, it is still a reasonable option when speed matters.

The recent popularity and effectiveness of CNN accelerate the growth of many new published solutions every year. The list shown in this section of available object detectors models is just a snapshot of the current state. Furthermore, this thesis tries to focus on the single-pass ones with the proven applications in resource-restrictive environments. Detectors' research helped to understand particular models from the YOLO family supplemented by few other single-pass options.

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 monitor a person's movement. A person re-identification tries to identify identical people across images in time or across images from multiple cameras. 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. 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 significant flaw. The system would create a new identity for a person when the match is not confident enough, which results in the trajectory interruption.

The re-id method extracts and compares persons' features 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 indeed applied for the solution. Moreover, video recordings are usually taken from a distance that makes the video very similar to the drone's camera view. In this work

context, identification is a vital part of the tracking mechanism that allows the system to match multiple individuals within camera frames in time.

For instance, face recognition is not a good approach as the people are captured from a significant distance. Instead, it is necessary to use information about how their whole body looks. Therefore, methods that allow the use of features such as body figure proportion, clothing, or walking pattern, are better suited for this. The resulting algorithm needs to consider 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* [33], [65], *SORT* algorithm [3], [64], or primary image features comparison methods for example based on histograms, or even already covered Histograms of Oriented Gradients.

Chapter 3

System Proposal and Implementation

The solution of tracking people in video can be described as estimating each individual's distance in a processed image. This estimate is further used to locate the individual relative to a camera's positioning system. This chapter consists of both hardware and software implementation details of the proposed solution. The first few sections aim to describe the targeted use case and its hardware, including its limitations. The hardware choices also define the used software toolkit, especially OpenVINO[1] software to take advantage of Intel Neural Compute Stick 2. The USB stick features a manycore vision processing unit. Its brief description and relevant usage are demonstrated 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.

Almost 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 expectation of getting all these calculations done is a fraction of a second for a single frame, in other words achieving at least frames per second (FPS) rate in the order of ones. That is covered more comprehensively in Chapter 4, which immediately follows.

3.1 Similar work tackling these problems

All the comparable systems are briefly presented in this section. The crucial aspect of the discussed solutions here is using a single camera sensor as the primary source for examining the surroundings. As proposed earlier, the monocular camera setup is quite popular in the automotive industry, because 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 [39], [53], [56], and especially [25] which provides the basis for distance estimation methods, Section 2.2. They all use a camera to detect approaching vehicles and estimate their distance to assist a driver. The detection methods vary and are aimed, of course, at 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 as they are. For example, finding the position of vanish line or the vanishing point in an image

[1] Open Visual Inference and Neural network Optimisation

could be an essential clue based on road lines. By knowing its precise position, it gives this approach much-needed accuracy over the traditional pinhole camera principle. Another widespread technique across previous works was an object's behaviour prediction. The 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 complexity. In addition, papers include many well-covered past work surveys that make them valuable information sources. Besides vehicles centred works, several more general ones deal with distance estimation [7], [55]. Both articles relate to depth or distance estimation, respectively. In the case of the second one, the distance is studied as 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. The drone itself provides another similar system, and manufacturers often include better or worse tracking systems to capture improved aerial footage. The DJI is a world-leading manufacturer of these small aerial vehicles, later in this chapter, the data from the DJI drone is exploited as well. The early solutions used a GPS beacon worn by a tracked subject, and 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 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. In short, the so far briefly described built-in intuitive flying action functionality is shipped with almost every DJI drone and is called ActiveTrack system [13].

The next available sources are examples of partial problems solutions that could be the most influential for the proposed system. Mainly, the independent projects that show ways to track either people or other objects in the image, not necessarily taken from a higher viewpoint. They use various object detection methods and run them 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 [19][50]. There is undoubted performance improvement using the accelerator chip than running 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 [23], 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 is necessarily the detected entities. Like 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 bulkier USB stick: Colar[2] USB Accelerator by Google, PLAI[3] PLUG USB artificial intelligence accelerator chip by Gyrfalcon Technology, or a pocket-size board module Jetson Nano[4] by NVIDIA. And the list is completed by perhaps

[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/

the 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. To elaborate more, 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 stated as an alternative for the overall picture, it is not considered any further for this solution.

3.2 Platform – The chosen hardware

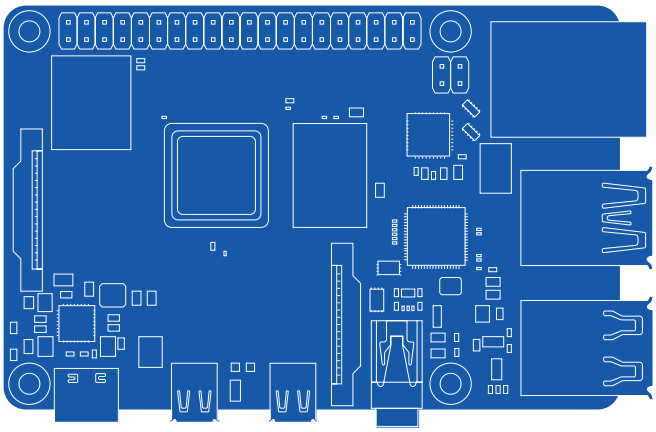
The whole work's initial idea meant to create a flexible and lightweight positions tracking system for aerial footage. Requirements for such a system are that it should be compact and yet still powerful to handle advanced computer vision tasks. As mentioned in the previous section, there are a few proven 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 extra 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 for this case is Raspberry Pi 4 small computer with Intel Neural Compute Stick 2 accelerator unit, both shown in Figure 3.1. Additionally, the carried camera unit must be able to supply the necessary data over the network, or the unit can be directly integrated with the processing hardware. The supervisor of this thesis lent the recommended computational devices in order to test the final solution.

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 module implementation responsible for utilising the previously used functionality. Finally, the presentation 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 just 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 that is dedicated to educational purposes in schools. Despite that, it is pretty famous for prototyping and research work or even used in robotics. All the technical details in this section are sourced from [41], 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 insertable 64 GB Micro-SD card as internal storage. The board can be controlled by the officially supported operating system (OS) called Raspbian [42].

[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 [58]

(b) Intel Neural Compute Stick 2 [22]

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.

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 communities.

The 4th generation includes wireless connection, and after the initial configuration of a Wi-Fi adapter, the development could 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 boards' 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 the 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 various computer vision tasks, often seen in robotics applications. The Intel Movidius VPU is also embedded in many smart devices on the market for automated analysis or for 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 are optimised for the models' inferencing, such as convolutional neural networks. In general, it can be a local 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 application through Intel's OpenVINO Toolkit. 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 lets them focus on the application.

3.2.3 OpenVINO

OpenVINO is a set of tools to orchestrate the whole development 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 support 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 as well. 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 good support for Raspbian, and the toolkit includes guidelines on how to install and configure everything on a resource-constrained device such as this. Through 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 practical examples 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 distinct data types include what is necessary for providing the application results. Many times discussed and analysed sensor is a camera, which gives the rich data that demand a high level of processing to extract the knowledge of it. Telemetry data is essential too; it defines the camera's rotation angles, position, and other miscellaneous yet still crucial information. 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. Although the following analyses are demonstrated on this device, a considerable amount of the principles would apply to other UAVs, especially those manufactured by DJI. The less portable parts are the descriptions of gathering the pieces of data from the drone's specific logs. 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 valuable required sensor data are covered back in Section 2.3. The information and necessary metrics described below can easily be mapped to different, completely unknown, vehicles' systems.

The drone has its own controller connected to a smartphone, and the operation is administered by installed DJI GO[14] application. The application is responsible for video

[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

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 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 sorted and filtered. Only the essential and the most accurate values are then used for reliable people tracking.

3.3.1 Camera

A camera is an excellent sensor to capture the environment around us. Nowadays, it is a primary source of information for countless applications. The camera attached to a drone is the most significant input for people tracking in the proposed solution here. Its image data is crucial for object detection and consequently for the re-identification of previously detected subjects. As a result of the past research, covered in Chapter 2, and primarily Section 2.4, the image pixels can be well interpreted and understood by machines. The individual frames of video are processed to obtain the positions of objects, persons, within 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. The dedicated camera unit features a recording capable optical instrument connected to the Raspberry 4 board. The data is captured and read from the camera, processed and streamed over the network into the tracking system itself.

Auxiliary variables about the camera are required as well in order to convert the object's positions from the pixel domain to their actual locations relative to the camera placement. The sensors and technologies on board, responsible for a drone's intelligent flight assisting, are used to define accurate camera position and rotation. The telemetry data reflects the drone's and effectively also the camera's physical positioning, more on that in the following section. 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, a field of view of the used lens, and the camera sensor size. These specifications are based on the camera type, and therefore it should be quite straightforward to find them in the related datasheets. For the unknown camera, the selected properties can be set by a calibration process [6] and derived from the calibration chessboard images. Another aspect is the image resolution, which can be derived from the currently processed 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 start.

3.3.2 Telemetry – Camera positioning

The proposed solution tries to achieve tracking capability in a video captured from a drone which implies that the camera is regularly moving. In comparison to a static camera, it is required to monitor this movement, and accordingly, the system should be able to adapt to new geometry caused by position changes. Additionally, the positions related to a previous camera's locations should again consider the new state to display correct and relevant information consistently. Thus, the requisite information is longitude and latitude

from the radio-navigation system and an azimuth measured by a compass. This all 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 sensors' readings are as accurate as possible. The drone's camera is typically mounted to the vehicle with a gimbal pivot. For the best recording capabilities, the gimbal can be mechanically controlled. It gives the camera a much-needed image stabilisation and reports the momentary camera rotations to the terminal application. If the gimbal supports a yaw axis rotation, the final azimuth has to consider that too, as the camera's direction is the needed one. To summarise, the positioning data is dynamic during the recording process, hence should be streamed together with the frames.

In terms of the DJI Spark drone, all this information is structured in the log file and available through DJI MOBILE SDK[15] API. The mobile application can also embed the telemetry data to the video recording by adding subtitles with the most used values. The desired Spark data includes [57]: 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. According to specification, the altitude measurement accuracy is noted as ± 0.1 m for the vision system and ± 0.5 m for GPS.

3.4 Additional software – The frameworks and libraries

With the idea of not trying to reinvent the wheel, this section gives a comprehensive overview of all the convenient software solutions utilised for the final work implementation. The foundation to every stated choice here was the implementation programming language, which was decided at the beginning of the project. Python language was a go-to option as its developing environment is broadly used in academia and is well suited for computer vision projects. The extra benefit is that I have been already familiar with the 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 following few paragraphs. 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 appraisal of feasible libraries to assemble the final application prototype. It is a good practice to know 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[37] 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

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

work with a video stream. It also includes the transformations of the stream's individual frames, pre-processing, or basically, any needed standard operation on image data. The data representation in Python's OpenCV module is covered by another package, NumPy[16], 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 – DNN[17]. It supports pre-trained networks from various frameworks and considerably accelerates the forward pass, and even can employ a VPU of an accelerator or other chips that offer parallel execution. The process aims to be as simple as possible, and the interface allows to load and run the selected network. However, inference outputs demand extra post-processing, especially in the object detection domain. To a great extent, the module support overlaps with what is already in the once covered OpenVINO kit. 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 uses pre-trained models that are proven to work well with resource-restricted hardware. It focuses namely, on YOLO models as they have a great speed and accuracy ratio, and then also experiments with SSD models. All are single-pass models with computational complexity that is 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 crucial aspect is the fact that models are available and perform well with suggested hardware. Moreover, the solution architecture should count and be prepared to work with different methods and should also allow their smooth switching for better testing.

Exposing and visualising the results

The results are sequences of relative locations or coordinates that form trajectories that are updated continuously. Therefore, the overall requirements for the visualisation library were flexibility of updates, ease of use, and ideally rendering into a web browser for convenient presentation. Bokeh [5] fulfils all of these with a good performance and also adds interactive manipulation with the resulting paths. It includes APIs for data handling and management, which is an extra benefit too. The library offers good tutorials, even for the more advanced use cases, which show numerous plotting options and a 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. The alternatives were tried out and served relatively well while not always fully covering the mentioned necessities. However, the Bokeh framework stood out and is used as the final presentation layer.

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, and it handles all the zooming and moving in the client's browser. The operations for datasets and colour palettes are supplementing the broad range of Bokeh's features.

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

The solution expects to transmit the final processed data from the main program for further manipulation to the presentation program. This could be achieved by exposing the information through RESTful [12] service at the processing unit site computer. Python's minimalistic web framework Flask [14] manages exactly that. Flask quickly creates a simple service that serves the data as needed and encapsulates all the network-related challenges. A client of the service – *the presentation layer* can query the processing unit at any time and obtain the resources by using predefined addresses.

Miscellaneous packages and libraries

Python environment comes with an extensive 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 many other functionalities the applications might require. Only a few relevant ones are briefly described next as their range of capabilities 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 [61] (Universally Unique Identifier), representing the temporary identifications for detected objects. Previously recognised persons' identities afterwards replace these temporary ones. Similarly, a logging facility [31] 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 [47] library. It enables the consumer program to create and send HTTP requests in order to exchange 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, are all resolved earlier and also delineate their constraints. By using all the knowledge from these requirements summaries, the solution design proposal is trying to meet the goal of tracking people in the video from a movable camera. That has led to a gradual partitioning of all problems into smaller and smaller fragments which were continuously revealing the solution contours. Finally, the idea is to have several separate modules that try to solve the tracking problem. First, the data have to flow into the system through the assembled camera unit over the network. The second one is responsible for gathering and processing the inputs, and the third for collecting what is prepared and answering the users' calls.

The presented design (Figure 3.2) is a base for implementing the application, and it gives a general outline of the problems which have to be answered in code, such as network

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

communication, process synchronisation, or data serialisation. 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 last unit is the presentation layer and allows user to view the trajectories of monitored individuals. The implementation then reflects these designing choices, a blueprint for the realisation, described later on, together with its known flaws.

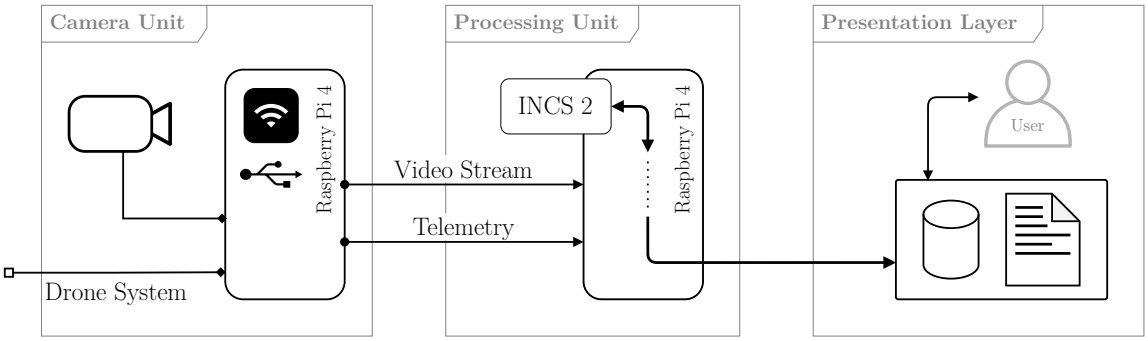


Figure 3.2: Diagram of architecture overview, including the three main units of the system. The data are recorded and transferred from the Camera Unit, the tracking information is deducted by the Processing Unit. The Presentation Unit stores and provides information further to the user.

3.5.1 Software architecture

The main modules definition has been already highlighted, and the *Processing Unit* should run autonomously by using a small computer as a host. The video stream and necessary telemetry measurements flow into the system from the *Camera Unit*, then the application should transform the inputs into desired results. The overall processing can be described in a few steps, and these roughly correspond to the concluding implementation as separate interconnected modules within this unit. Accordingly, object detection has been identified being the most complicated procedure, and indisputably, the solution should offload it to the accelerator hardware. The detector model inference is transferred back to the general processing device and passed to the re-identification method. People who appeared in the video already are assigned their past identities. Alternatively, when the person is not recognised, the profiles database makes a new record for them. Thus far, the pipeline's outcome is bounding boxes of recognised individuals in the current image with their particular identifiers. The ids provide essential information to connect and extend the past trajectories for each profile in the database.

Before the trajectory is updated, the location must be converted from the image domain to the real world one. Detection boxes are the base for distance estimation algorithms, and these rectangle coordinates in pixels are transformed 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 interchangeably. The outcome from this system stage is a list of the distinct position data for each person. The position extends a sequence of past locations of a person with a certain, uniquely assigned identity or profile. For better insight, the whole architecture is illustrated in the diagram in Figure 3.3. The lastly discussed segment is the propagation of this list of locations through REST API to the

second application, where the data is cached and displayed to a user. The presentation layer can, at any time, request the processed output from the REST service that runs on the processing unit.

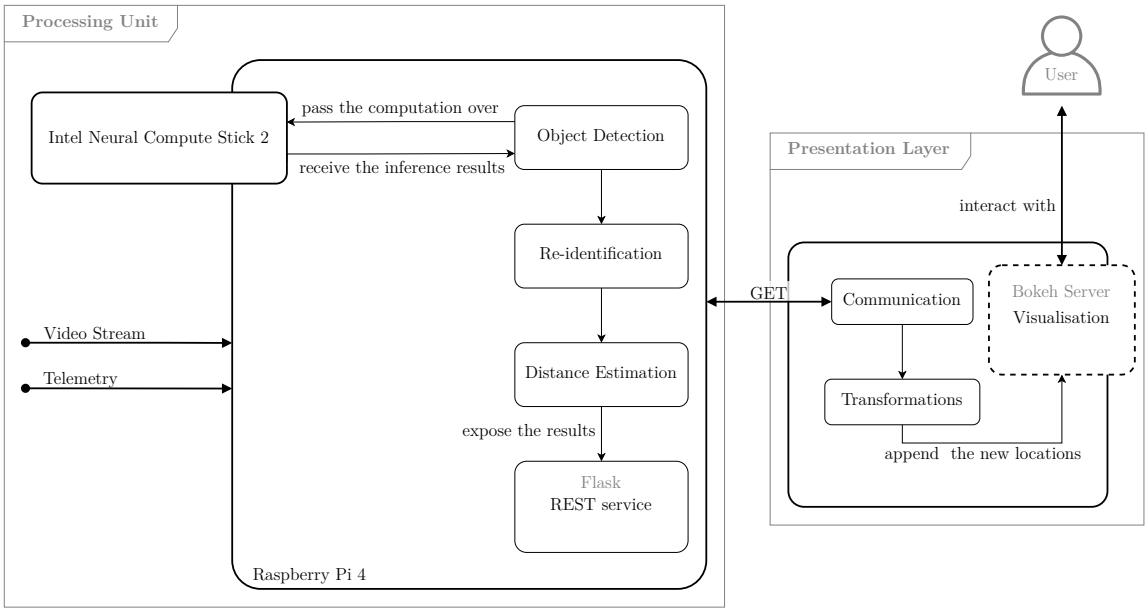


Figure 3.3: Diagram of the proposed solution architecture. Showing the two main application parts along with the relevant internal submodules and their hierarchy.

Presentation Layer shows that its logic is composed of a communication and data transformation part, and afterwards, the data is prepared for visualisation. The connection to the main processing application is handled by the HTTP requests via predefined REST resource endpoints. The used method for the request should be GET, according to the best practices. This way, the application can receive the data and apply the translations when required. Trajectories are sequences of past locations and the current ones queried from the processing unit. In case the camera is rotated, the previously detected locations have to be altered as the viewing situation changed and new data relates to this new state.

People's positions in the actual camera shot are constantly updated in the background. The visualisation framework's underlying data is supplied 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, and the presented user interface is built by the framework.

Processing part internal data flow

In a close look at the more complex processing part of the system, the architecture design can give more clues about which parts can be executed in parallel. By distinguishing which steps of the system can run simultaneously and the flow of the data they demand, the program can complete them in separate threads as the final application runs on the multicore processor. The work can also be offloaded to the accelerator, which opens opportunities for parallelisation too. The proposed architecture defined these four main components: Object Detection, Re-identification, Distance Estimation, REST service powered by Flask's web components. 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, such as the object detection models might demand a fixed 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 final results are made available by Flask service afterwards; for the whole execution, refer to Figure 3.4. The service exposes the computed location estimates to a consumer application. Flask creates a new thread for each connection request, but in the figure, that is 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.

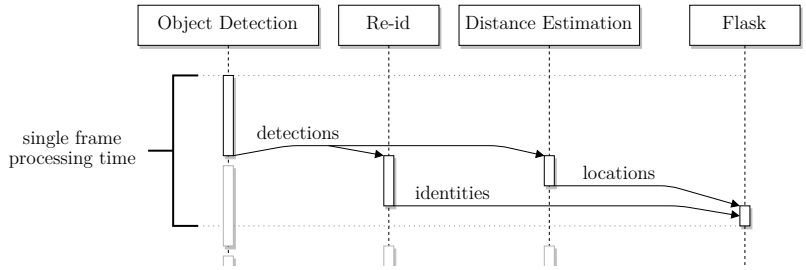


Figure 3.4: Sequence diagram of the Processing Unit application concurrent run. The arrow labels represent the different types of exchanged results from the thread workers, the Object Detection thread assumes the immediate availability of a new frame from the camera stream.

To conclude, just the two submodules can run in parallel, despite that while they are running, the object detection for the subsequent frame can run as well. When the other modules complete their task before the detection itself, the final performance should depend just on the detection speed. Based on the fact that object detection is the most challenging problem within all the components. Modules can also run on shared hardware with the camera unit. Both units can be attached to the drone and run on the same Raspberry Pi board while exchanging the data via the local loopback network.

3.6 System implementation details

The following section uncovers the implementation details of the proposed solution. It includes the made implementation decisions, which are reflected in the code base of all application parts. The implementation is based on the architecture principles, which still left a few aspects unclear, and the intention is to describe them below.

The first step of the programming part was to configure the developing environment, install all the dependencies and set up proper communication between Intel's OpenVINO kit and INCS stick. The process can be significantly long as OpenVINO requires running many preparation scripts, but eventually, it can run on both Raspberry Pi and a standard desktop computer. Moreover, Python runs smoothly on both platforms, so code is easy to move from one to another, which was also employed during the development process. The camera & data gathering part provides data for the tracking processing unit. Raspberry Pi serves again as an operating unit with the necessary connectivity to transfer data and allows attaching the camera hardware and the telemetry source. The camera can be easily

connected and set up within Raspbian OS [60] due to the widespread support of the various USB devices. When the camera is connected, the data can be streamed to the desired location using the implemented script. It handles all the necessary video manipulation with `ffmpeg` converter.

3.6.1 The vision processing part

The central processing module, which can operate directly on the drone, has four necessary subroutines. They have been already explained in the architecture, together with how to parallelise them. 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. The provider is consuming the data from the camera unit and shielding the application insides as it extracts only necessary information from assorted sensors values.

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 a live camera feed of a connected unit or a file. On the other hand, the telemetry is sourced from a pre-recorded or streamed flight log file, which format is a standard *comma-separated values* or CSV file. All the development and testing were accomplished with a video file streaming and a CSV file combination. The camera feed was used as well, but only for verification of this implemented feature.

Next, object detection is handled by `DetectionProvider`, re-identification by `ReIdProvider` and distance estimation by `DistanceProvider`. All the providers accept a particular implementation of the method, which they later utilise in the same manner the stream class uses different data sources with common interfaces. That is achieved by a polymorphism, a natural property of Python as an object-oriented language paradigm representative. Each provider has its thread, which executes the supplied method whenever the new inputs are available. For example, the object detection provider 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 on demand, which makes experimenting a more straightforward process. 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 compared with each other and evaluated. The threads output their statistics through logging messages, so the level of shown verbosity can also be tweaked.

The application is a standard command-line application where start-up arguments define the methods and options. Its thorough usage guide is given in Appendix A.1, the help guide. The partial results of the object detection and the re-identification stages can be on-demand visualised within the processed image. The OpenCV library optionally creates a window that automatically fits the image. The displayed partial results include the highlighted bounding boxes and also identity labels. The next main stage of distance estimation represents the final results, visualised to the user by the second separate program. Although, the visualisation module can be used directly within the processing application as an alternative, the data are passed inside the python interpreter instead of using the REST service. Figure 3.5 shows all the results after each of the stages.

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

Figure 3.5: An internal visualisation of the individual application stages. The *detections* show the inference bounding boxes with generic class labels, and the *re-id* stage shows its results with particular identities defined by unique id and colour. The *locations* stage is just an illustration of the actual text data which this particular stage outputs.

The featured methods have their limitations too, and they should be considered during an interpretation of results. The found locations are still just estimates; as seen in the example figure, a faraway person at the top is not recognised, and the used YOLO detector also responds with not fully align boxes and their objects. 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 for *settings* are `Camera`, `Gimbal`, `Position`, and `Misc`, which handles additional data. 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 convenience purposes, offers the way to decide whether or not the camera is stable. Hence, when the drone moves from one stationary position to another, the image is often undesirably rotated or unstable. The fixed tolerance set the maximal speed in all axes to a half metre per second by default. When the readings are out of the boundaries, the frame is not processed.

3.6.2 REST endpoints specification

The following part of the system covers the communication protocol. Final estimates should be transmitted to the presentation module for further processing, namely for aggregation and visualisation. The REST service creates a link between these two programs. The processing unit's application exposes its 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 such as

```
[
  {
    "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 comprises a list of recently processed frames results, and each *estimates* element contains the locations of detected people in the actual frame. The transmitted object's locations are the personal keys, and the position offsets pairs, then the syntax is as follows `"id-uuid": ["delta x", "delta y"]`. All the offsets are represented relative to the positioning element of the individual frame. The position element is the GPS location of the camera at the time when the image was taken. The communicated data also includes altitude and, more importantly, the camera's bearing that allows the client application to join the trajectories even when the camera rotates. Additionally, the whole history of all processed frames is accessible from `/v1/locations`.

3.6.3 The client application

This detached part of the proposed solution is responsible for regularly requesting the data from the processing unit and offers it to the user. When the batch response is received, each item's distance estimate is appended to the respective current trajectory. The unique id of individual points links a person's path. Moreover, the camera's location determines if the view should change before the inclusion of the points. Therefore, when the vehicle or camera rotates, the past locations compensate for this motion in the shown final graph, so the newly appended points are added to the correct coordinate system.

The user can access the trajectories diagram at port 5006 via HTTP, as the default configuration of the Bokeh framework. Figure 3.6 illustrates an example of the final results, and the page includes standard tooling for resizing and dragging the canvas. The trajectories are represented as lines, with the detected locations highlighted as circles.

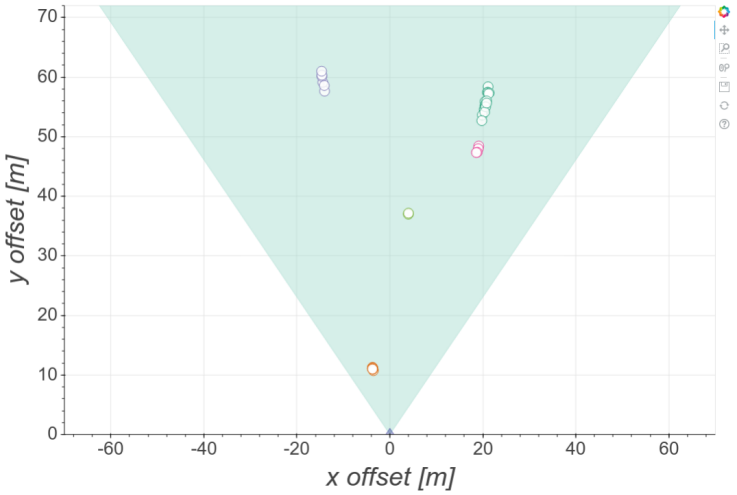


Figure 3.6: UI visualisation – Bokeh UI. The light green triangle represents the camera's field of view, and the camera position is the origin of the used coordinate system. The circles are the detected people's locations and connected with the lines of the same colour.

An interesting observation was made during the implementation while all the frames of video were processed. The video frame rate is 30 FPS which led to analysing almost identical frames one after another. The saved video data enabled this way of thorough offline processing of every frame. The trajectory was then composed of many points close to each other, and the results were not satisfying. These many points slowed down the UI rendering of the final implementation. Therefore, the presentation layer adds smoothing filtering and shows only indistinguishable locations for the corresponding identity profile. That might also help stationary subjects in the image as they could be visualised as fewer points when there is only a minor difference in the consecutive estimates. The analysis is done on the presentation layer only, and all the data points are still processed and stored for further use as needed.

Chapter 4

Experiments and methods evaluation

As stated in the chapter above, the implementation is capable of easily swapping the particular methods for each of the described subproblems. The different methods are covered in more details in Chapter 2, and the most significant ones were tested and tried with the available sample video data. The following part offers a view of implementation usability and overall performance. Due to a narrow scope of available data, the experiments are limited to a dataset provided by the supervisor of this work. The dataset specifics are discussed in Section 4.1, and then the obtained outcomes and issues are summarised in the subsequent Section 4.2.

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

Most of the models mentioned above are not strictly trained on aerial images, and as a result they might be less precise with video taken from above or from a drone. Nevertheless, the image processing part of the system should adequately answer two main questions: is a person present in the image, and where is within the image. Moreover, the system should be able to resolve if the person has been seen before. Methods for re-identification are more invariant to the different camera view, making looking for the right one easier. However, imperfections in these parts can lead to poor overall localisation. Only one camera usage can significantly reduce the need to include sophisticated methods 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. This results in the usage of a much simpler algorithm that can offer the corresponding accuracy for this very case. The coloured histograms features serve this purpose and use correlation comparison with a set threshold limit. When the correlation coefficient is below the threshold, a new profile is created for a person as none of the profiles in the database has achieved that solid level of similarity needed. The histograms similarity addresses the two main problems for the tested dataset. Namely, it is not generating too many new profiles for the same person and secondly creates valid chains of successful matches in time. Furthermore, a method like that comes with a lower computational cost that is always advantageous.

The last part of the range of used algorithms is the one 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 m[1], the average for both genders adult height. It is an average of males (1.80 m) and females (1.68 m) mean height at age 18 in Czechia between years 1896 and 1996. On the other hand, the position based method is strictly defined by its equations.

4.1 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 relatively standard positional capturing is included with a bit of wind that is causing minimal shaking. Overall the available data were taken under excellent conditions, and this might mean the provided telemetry is accurate. The lighting conditions were also satisfactory, and the image is sharp with minimum noise.

Drone positions vary through the recording, its altitude is from approximately 5 to 10 metres from the ground, and the GPS signal is constantly strong. The people in the frames are clearly visible most of the time, but the samples also include a section where a treetop partially occludes them. The camera moves around to get static views of both standing and walking around subjects. The recordings 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.

(a) drone and people positions (b) marked distance is 31 m

Figure 4.1: Manually estimated positions where the observed people in one of the data sample are located. People in the video are mostly stationary, numbered from 1 to 6, and the drone position is marked with the grey pointer. Images are taken from aerial map [34].

The recorded people are positioned between the visible pathway and nearby trees, this surroundings features are the main clues for estimating positions manually. Based on that, the people are always located at a distance of 10 to 40 m from the camera. The data are originally split into video samples and a separate log archive file of the entire flight session. The data preparation required synchronising these two parts for working with the provided dataset offline in a repeatable manner. The exploring and stitching of the data are done

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

with the help of Flight Log Viewer[2], which can visualise the recorded information and convert it to other formats for next use. Thus, there might be a slight misalignment of the offline CSV data logs and its video feed.

According to DJI Spark Specification sheet [57], the mounted camera uses 1/2.3" CMOS sensor, with dimensions of 6.17 mm × 4.55 mm. The lens has a field of view angle of 81.9°, and its focal length is 25 mm. The frame rate of the recorded samples is 30 FPS. An example of the used configuration file is shown in Appendix A.2.

4.2 Experiments and testing

Each stage covered in the implementation section can accept different methods explained in the previous chapters. The goal here is to compare and evaluate the methods first separately within their intended use case and then the system as a whole. All the mentioned analysis was done on the available pre-recorded data. In order to better mimic the actual use case, the camera unit is simulated locally on the processing unit. The video and flight log are streamed from the files and processed further on a small computer device with a connected accelerator stick. The test environment was a Raspberry Pi 4 with connected INCS 2, and the board was connected to a local network. The presentation application ran on a separate device within the same network and collected the data of all individual test runs.

Any more precise details are described as needed, otherwise the testing and performed measurements used this common setup to obtain all the presented values below. The used data had the required balance of high-quality video and broad sensors readings history related to each video frame. This is a principally important piece of information in the last stage of the system, and other public open datasets commonly used in CNN applications lack this piece.

4.2.1 Object detectors

The models were chosen based on their declared performance and characteristics. Factors to look for were the industry standard mAP metric and the fact that they were previously applied in resource-restrictive environments, mainly preferring the less complex single-pass ones. The choice also included the ease of use within the INCS toolkit. `ObjectDetector` class is implemented three times to encapsulate the SSD detector and two detectors from the YOLO family. This specific SSD [35] (ssd_mobilenet_v1) uses MobileNet and is trained on Pascal VOC Challenge data from years 2007–2012. YOLO models use the Darknet network, and the experiments are done with the Tiny YOLO models [43]. They were trained on the COCO dataset, while one (tiny_yolo_v3) is a purely general object detector able to recognise up to 80 classes. The other one (frozen_darknet_tinyyolo) was used in a similar people detection application [51] and was able to detect more pedestrians in the test recording, see Table 4.1.

Additionally, only the frames that relate to a stable camera state are accepted by the detector for subsequent better distance estimation. This restriction helps to prevent wasting the limited resources on the data recorded when the drone sensors might output unstable measurements. Speed in any direction cannot exceed the set limit of 0.5 m/s, so this helps to filter out blurred or unstable imagery. The sensors also give much more accurate telemetry data when the camera hovers almost still, as the higher speed usually influence its tilt. In all the test scenarios, the excluded frames were up to about 10 % of the total.

[2] https://www.phantomhelp.com/logviewer/upload/

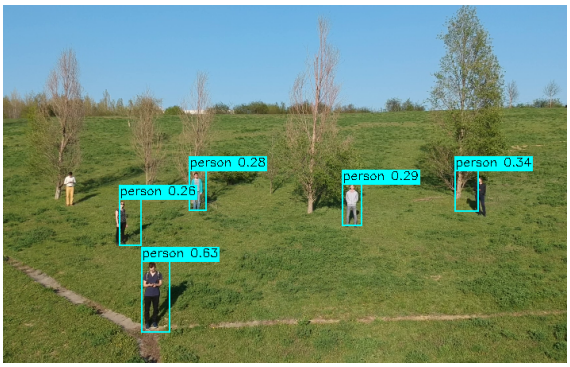
Table 4.1 summarises the carried out trials of different models and the number of found objects – people relative to the defined minimal match confidence limit. The used video was about 40 seconds long, and in the camera view, there are six fully visible persons. Three of them are moving and walking, and the rest three are standing still most of the time, even barely moving. Moreover, one person's body is not entirely present in the frame for about 5 seconds. Thus, the number of people who can be detected in the test sample is about $900 * 6 + 160 * 5$, and that is 6200. This is just a reference value, meaning in every frame, every person is detected.

	Declared model values				
Model	mAP	FPS on GPU	FPS	confidence	# detected people
ssd_mobilenet_v1	0.727	46	15.1	0.20 0.10 0.05	913 913 913
tiny_yolo_v3	0.331	>200	7.5	0.20 0.10 0.05	288 380 517
frozen_darknet_tinyyolo	0.237	>200	7.1	0.20 0.10 0.05	1 122 2 211 3 646

Table 4.1: Comparison of tested object detection models and their number of successful detections. The total of analysed frames in the test video was 1060, most of the frames included six people. Declared model values are sourced from [35][43]. FPS is measured for only the detection inference, which runs on the INCS 2 accelerator.

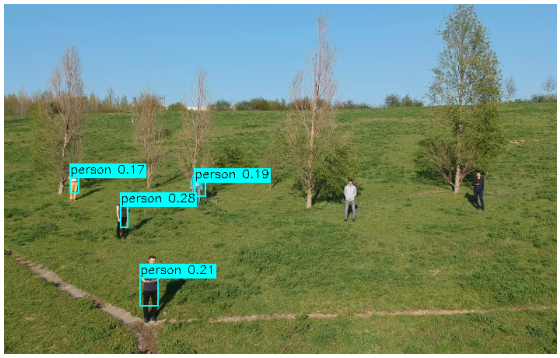
The SSD detector gives stable, well-positioned boundary boxes with decent confidence. The lowering of the confidence limit did not have any impact on this particular model. The number of found people is relatively low, and rarely all the people are found in the frames; one of the exceptions is shown in Figure 4.2a. The person in yellow positioned far back left is not moving during the whole video and is not recognised at all in any frame. The model can output multiple boxes for only one person, or when two people are above each other, it can blend them as one result, but both cases occur very occasionally. However, a slight horizontal misalignment of a detected person and their box can be seen in the few results. These left and right shifts although have minimal impact on distance estimation. Figure 4.2b shows the second model results, which are very low in terms of the number of successfully detected people. Just one person is sufficiently recognised, but on the other hand, the model's boundary boxes are well positioned around them. Due to the low number of detections, this model is not considered for further evaluation. The next model from the same YOLO family performed in the opposite way, and it was able to recognise all six persons and even from quite a distance. The detected matches mark the people in the image nicely, but it can output infrequently duplicate boxes close to each other while detecting a single person. This outcome was more present when the confidence limit was too low; a good value was around 0.1–0.2. This may be a good candidate for the next system analysis, yet a further manual examination of the produced boundary boxes led to the conclusion that this TinyYOLO detector infers boundary boxes that often cover just persons' upper body

or only their torso (Figure 4.2c). This would cause significant inaccuracy and inconstancy in the following stages. To conclude, the SSD detector provides good and stable matches of recorded people, capturing them from feet to head, and the detections of this model are the most optimal for the testing of the next stages.



(a) ssd_mobilenet_v1

(b) tiny_yolo_v3



(c) frozen_darknet_tinyyolo

Figure 4.2: Visualisation of the typical successful inference of the models. The shown images represent the single frame analysis; results are generated during the models' comparison.

4.2.2 Re-identification in time

The primary method for the detected person re-identification should be simple enough to be easily executed on a small computer such as Raspberry Pi 4. Histograms correlation of the found persons' boundary boxes is used and assessed in Table 4.2. The histograms are extracted from the cutouts, resized to a standard size of 64×128 with all three RGB channels. Profile of a person contains histograms values and additionally added the centre location of this cutout image. In order to keep the profile matching quick, even for many stored profiles, they save the histograms only for their most recent match. Profiles create links between found location estimates so the trajectories can be adequately connected.

The evaluating is done on the 331 detection matches extracted by the SSD detector. The used testing data includes six different people in the video, of which just five are detected, and this was also confirmed by manually inspecting the gathered boundary boxes. The detection ran on the trimmed video section from the object detectors test; that section included multiple suitable matches in each frame and avoided ones without any match. Ideally, the number of profiles should be 5, respectively, slightly higher as a few double detections are outputted by the SSD model. There are 3 cases where the person had two

boundary boxes around. As each profile can only be used at most once per bounding box in a single frame, the total number of *unique* profiles in the analysed video should be at least 8.

Later during the testing, the previous match location in the image was added to the profile comparison. Even simple feature descriptors such as histograms can perform relatively well boosted with this extra added information. The results using locations are significantly improved while still maintaining the low computational complexity. The previous match centre coordinates help maintain the profile continuity, resulting in one constant profile of the person with the most matches (person A – closest to camera).

	settings				
Method	corr	loc	# total	# short	max len
histograms – 32 bins	0.5	–	244	236	17
histograms – 16 bins	0.5	–	18	2	35
histograms – 8 bins	0.5	–	14	3	59
histograms – 16 bins	0.6	–	35	19	31
histograms – 8 bins	0.7	–	14	3	59
histograms – 16 bins	0.5	∞	11	2	95
histograms – 16 bins	0.5	0.3	12	3	95
histograms – 16 bins	0.5	0.1	17	7	64
histograms – 8 bins	0.7	∞	9	1	95
histograms – 8 bins	0.7	0.3	10	2	95

Table 4.2: Comparison of different settings of re-identification and their effect on the matching of previously seen persons. *corr* refers to the correlation coefficient threshold, and *loc* is the maximal distance threshold for feature vectors extended with the person's last location context. The number of short profiles is defined as the number of profiles with a length less or equal to 5.

After observing the produced profiles' series, the main challenge for the method is to assign the right profile to the detected person if not all of them are currently detected. A different subset of people can be found between frames' succession, and it can lead to a problem of purposely not using the particular profile. Another essential point that can adulterate the re-identification is the background in the recording which is very similar for all subjects. That might cause a high histograms' similarity between unrelated profiles. When one of the people is not detected in the couple subsequent frames, the location closeness is less effective when overcoming the issue. Such problem of *profile stealing* is partially corrected by adding an extra threshold for the distance measurement of the final profile feature vector that consists of $\big((centre\_coordinates\_of\_the\_match) \quad correlation\_value\big)$. The threshold defines the maximal allowed distance of the two feature vectors and increased the number of profiles, but prevented almost all occurred profile swaps.

4.2.3 Distance

The distance estimation is based on only the current image and the information from the object detection. At this stage, the system works with camera properties that are well known for the specific recording, and for the first time, also works with the telemetry data. Hence, the computation is less dependant on the overall movement of subjects and only computed for successful boundary boxes detections – for found people. The video recording

selected for the experiments below is identical to the one from the previous Section 4.2.2. The drone altitude for the experiment case is about 8 metres. The histograms method uses 16 bins, correlation threshold of 0.5, and maximal distance of the final profile descriptors is 0.3, which provided a decent distance estimations series. Distance estimation also filters out the detections which share one or more boundary boxes' sides with the edge of the frame, as this case implies that not the whole person is detected.

One of the comparing cases is dealing with difference in size based and position based distance estimation methods. The size based method results are way too inaccurate, resulting in up to three times the real distance from the camera for the same test case where the position based method outputs a maximum distance of 63.4 m (Table 4.3). If position based results take into account the uneven terrain of the recording, the distance is estimated to almost 40 m maximum. The variety in bounding boxes' size and the actual person size is interfering and pushing the size based method to its limits, especially with the detector that does not produce optimal boxes. On the other hand, the implementation of position based estimates does not manage the changes well in the area topology.

Distance estimation method	A [m]	B [m]	D [m]	E [m]	F [m]
Size based (set height 1.74 m)	53.19	85.48	91.44	82.33	86.90
Size based (set height 1.80 m)	54.18	87.92	94.39	85.02	90.49
Size based (set height 1.68 m)	50.91	83.14	88.65	79.60	83.76
Position based	14.00	26.99	63.37	37.26	59.53
Position based with altitude compensation	14.00	22.03	39.74	27.25	33.55
Altitude offsets based on topographic map	0.0	+2.0	+3.0	+2.5	+4.0
Manual estimates based on environment clues	17	24	31	28	33

Table 4.3: Comparison of different distance estimation methods, person C is not detected in the used video; hence it is excluded from the table. Supplemented by real positions estimates based on the environment clues of the scene. The people are named A to F respectively to the numbers 1 to 6 labelling in Figure 4.1a. All reported distance values are averaged across all analysed frames since people were standing still.

Dist. estim. method + detector	A [m]	B [m]	C [m]	D [m]	E [m]	F [m]
Size based (1.74 m) + YOLO	122.19	170.79	243.14	205.78	162.83	201.09
Size based (1.74 m) + SDD	53.19	85.48	–	91.44	82.33	86.90
Position based + YOLO	15.91	41.46	115.64	111.50	62.01	83.68
Position based + SSD	14.00	26.99	–	63.37	37.26	59.53
Manual estimates	17	24	36	31	28	33

Table 4.4: Comparison of different distance estimation methods in combination with both YOLO (frozen_darknet_tinyyolo) and SSD (ssd mobilenet_v1_caffe) object detection models. The SSD model is also used in the previous table; thus the values are identical for these rows.

Table 4.3 compares the final results of the system and shows the degree of how accurate it can be even under challenging conditions. Another essential point shown in the table is an error rate of results which is in the magnitude of few metres for the close subjects. The error increases with the distance, although the recording's area terrain mostly causes this. The evaluation of the position based method with altitude rectification tries to show

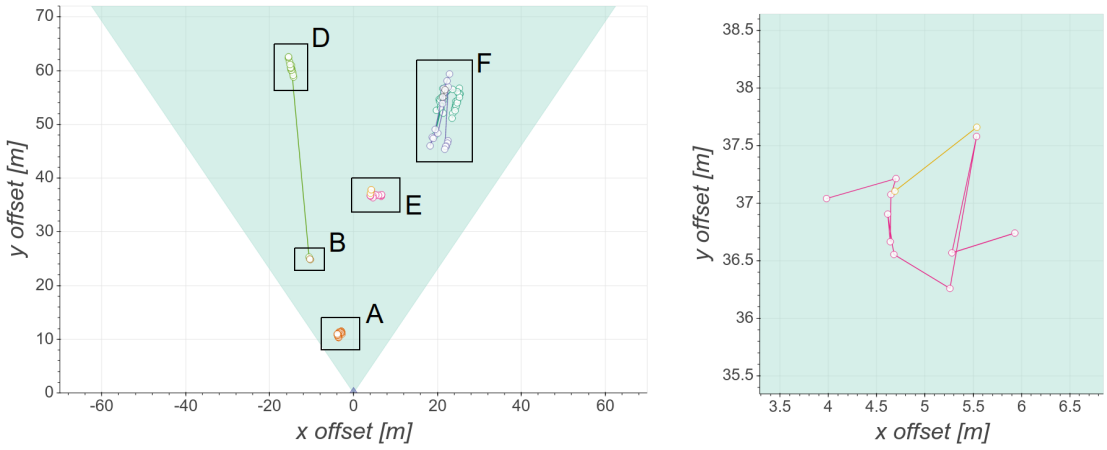
the real difference in the results. It isolates from the unexpected variety in the topology of the captured surface. The compensation simulates the ideal flat ground plane and demonstrates the true potential of the used method. Such a real-time altitude adjustment is not incorporated into the final system, and the calculations were based on the manual terrain assessment and contour lines of a topological map. For example, person E actual location (manually estimated) is about 2.5 m higher than the ground under the camera. Thus, this difference can be used to re-calculate the person's E distance from the camera, which improves the final results of the method. In this particular case, the method's estimate is 37.3 m which is only 27.3 m after applying the altitude amends (the real estimate is around 28 m). In the case of the altitude compensation, the system would need an accurate height map of its surroundings for such enhanced calculations.

Contrarily, the size based method is unpredictable, with very little room for any improvements. The results for this method carries a considerable error rate. The method can eventually operate in controlled conditions where the heights of detected people are well-known for each profile. Moreover, perfect object detection is essential too, and all people must be fitted with accurate boundary boxes covering them from their feet to head. Demonstration of how much size based method is dependant on good object detection is highlighted in Table 4.4. In the YOLO model, the inferred bounding boxes include mainly the torso of the detected individuals, as already shown in Figure 4.2c. The position based method is impacted significantly less by the imperfect detections – the deviation is about 50 % of added distance compared with huge deviation margins in size based results. The offset is caused but detecting the person's lower edge at their knees instead of the point where they touch the ground.

4.2.4 Overall system performance

All the stages are compared and the missing part is to obtain the estimated data and try to visualise them. The final example of results presented in Figure 4.3 uses the short video section of the drone's close approach when recorded people are just standing. It is the same recording as in the previous two sections. The used videos are more focused on people detection and movement rather than on the movement of the drone itself. The camera recorded from a stable stationary position and rotated a few times. This allows obtaining the optimal sensors' data. However, the locations always relate to the current drone's GPS position, and thus they can be transformed accordingly even when the drone moves. The scene does not change drastically, which help to maintain the constructed identities. Moreover, it was easier to synchronise the telemetry log with the stable drone positions when working with the already captured data. A slight misalignment should not affect the results as the sensors readings are changing minimally.

The figure shows the tracking solution results for the SSD detector, the histogram comparison method including previous location context, and the pixel position based distance estimation. The object detector has the most precise bounding boxes, while the position based estimations also outperformed its competing size based alternative. The detector did not recognise person C, and thus there are only five clusters of points in the results (Figure 4.3a). Distances correspond with the averaged values from the previous Table 4.3, row *Position based*. The long green line on the left is caused by the last persisting profile stealing as person B is not detected in the subsequent frame, and their profile is picked by person D. When the B is detected again, the new profile is issued for this person. The similarity in the background and the fact that the profile is vacant for the frame allows this



(a) detected locations overview

(b) zoom-in view for person E

Figure 4.3: Testing results visualisation. On the left, the summary view from implemented presentation layer with the found locations, people mainly remained stationary. On the right, a zoomed-in view of person E estimated trajectory. The camera position is $49.2172369N$, $16.6108169E$ with bearing of 84.4°.

wrong re-identification. Figure 4.3b adds how even the stationary person can get the locations estimates within about a metre radius as the bounding boxes fit the person differently every time they detected. That can be triggered by simple posture changes or variance in the pixel data between consecutive images. Moreover, this particular case shows that a few detections are assigned a different profile. They marked with two colours indicating the person was assigned the second profile, which might be caused by some misaligned bounding box that did not correlate enough with the established long-term *pink* profile.

The system's estimates for persons A and B are quite accurate, namely 14 m and 27 m. The results are reflecting the true locations, especially in the lower part of the image. The error increases as the individuals are detected in the middle and the top half of the frame, which is caused by uneven and slightly uphill terrain. Moreover, the objects get smaller, and the detected bounding box misalignment has more significant impact. That can also explain the widely spread cluster of points of person F. This person gets double detections in several frames as both the person and their shadow are detected, explaining the two colours in the points. In general, other factors can also contribute to a certain portion of the error rate, such as optimisations or limited precision of the floating point operations. Then, the error in some sensors data accumulates over the flight session and can also negatively affect the distance estimation.

The tests partially confirmed that the models still struggle with small-scale objects, especially on the less powerful hardware where only neural networks with a limited number of parameters can run. 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, but a notable disadvantage of the approach is the needed extra processing power. The efficient power boost can be using multiple accelerators and offload the work to each of them. On the contrary, the split image creates a problem with a person at the cutting edge.

In addition, the used re-identification's added previous location improvement worked well for the available analysed dataset, but much more crowded public areas with lots of

overlapping pedestrians paths might bring this method to its limits. Histograms would likely include similar background when the people try to avoid one another. This might result in a high correlation rate, and as they would be quite close, the past location context feature would also be less effective. A situation such as this could eventually break the correct person's profile tracking. So, there is still room for further experimenting with other published methods.

The most crucial limit of distance estimation methods is the environment, the used methods need lots of information to perform as intended. The amount of accessible telemetry data is sufficient enough for the majority of modern drones. However, the two presented methods struggled with more contextual data. The size based method, as its name implies, requires knowledge of the size of the recorded objects, which might be difficult to get and link with each pedestrian. Complementary to this, the position based method requires flat terrain where the people are captured, and as shown in the experiments above, this can be problematic to guarantee. Additionally, the bottom edge of the particular detection bounding box is considered as the contact with the ground and a person. This might raise several problems, and if the person is on the uneven terrain of the recorded area or stand on a platform, the final estimates could be significantly different from the reality.

Chapter 5

Conclusion

The implemented solution is capable of processing the video onboard the drone, using the compact computer with the accelerator. The results are then presented to a user in the form of a graph that includes the data points representing locations. The dots are position estimates of the individuals found in the video. They are gradually matched with their past self, and the connected dots finally build the trajectories. The system is tested with YOLO and SSD object detectors models, which give mixed results as used, and therefore there I still see room for improvements. YOLO has detected significantly more people in the video, while SSD precisely fits the people with rectangular boxes. The absolute error in location is caused by the inaccurate persons' detections and their misaligned bounding boxes, combined with the insufficiencies of distance estimation methods. The distance calculation creates substantial demands on the correctness of the detection and recording environment.

Next, a source of the input data is rather an entry level consumer drone but is loaded with the necessary sensors to help the described distance estimation methods. Nevertheless, the proposed application can be further easily extended to accept other data sources of particular drones while the core of the methods remains intact. This is all due to well proposed and implemented the modular system with straightforward data transfers. Tests on these video recordings included the complete pipeline verification and getting the final locations of the people who appeared in the video.

The implemented system got satisfactory results when the conditions are ideal, though when the input is pushing the system and its used methods to their limits, the decrease of accuracy is more noticeable. Future work should focus even more on the models for object detection, even try the new ones which are yet to be published. It should consider the more advanced re-identification methods with neural networks, which can run on a separate accelerator unit similarly to the object detection in this system. Alternatively, 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. The object detectors are multi-purpose detectors, inferring all sorts of objects, and the idea of post-training on just people samples can also better adapt the model for the solution. However, this may mean the need for a new dataset as such.

The drone industry is growing rapidly with its business potential too, and the 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 control and usability. The manufacturers themselves are incorporating object tracing to the products for collision avoidance.

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

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

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

[11] 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.

[12] 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.

[13] 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/.

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

[15] *Flight Altitude – Basic Concept* [online]. DJI Developer SDK, 2020-03-27 [cit. 2021-02-22]. Available at: https://developer.dji.com/onboard-sdk/documentation/guides/component-guide-altitude.html.

[16] 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.

[17] 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.

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

[19] 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.

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

[21] 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.

[22] *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.

[23] 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/.

[24] Kaplan, E. and Hegarty, C. *Understanding GPS/GNSS: Principles and Applications*. Third editionth ed. Artech House, 2017. GNSS technology and applications series. ISBN 978-1-63081-058-0.

[25] 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.

[26] 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.

[27] Langley, R. The Mathematics of GPS. In: University of New Brunswick. Available at: http://gauss.gge.unb.ca/gpsworld/EarlyInnovationColumns/Innov.1991.07-08.pdf.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

[32] 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.

[33] 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.

[34] Mapy.cz, Aerial Map, Brno – 49.2172361N, 16.6108150E. [online]. [cit. 2021-05-13]. Available at: https://en.mapy.cz/turisticka?x=16.6109617&y=49.2172238&z=19&base=ophoto&source=coor&id=16.6108169%2C49.2172369.

[35] MobileNet-SSD. *A caffe implementation of MobileNet-SSD detection network* [online]. chuanqi305 [cit. 2020-05-13]. Available at: https://github.com/chuanqi305/MobileNet-SSD.

[36] Nguyen-Meidine, L. T., Granger, E., Kiran, M. and Blais-Morin, L. A Comparison of CNN-based Face and Head Detectors for Real-Time Video Surveillance Applications. *CoRR*. 2018, abs/1809.03336. Available at: http://arxiv.org/abs/1809.03336.

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

[38] *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.

[39] 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.

[40] 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.

[41] 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.

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

[43] Redmon, J. Performance on the COCO Dataset. *YOLO: Real-Time Object Detection* [online]. [cit. 2020-05-13]. Sources at: https://github.com/pjreddie/darknet/tree/master/cfg. Available at: https://pjreddie.com/darknet/yolo/.

[44] 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.

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

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

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

[48] 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.

[49] Renfro, B. A., Stein, M., Reed, E. B., Morales, J. and Villalba, E. J. *An Analysis of Global Positioning System (GPS) Standard Positioning Service Performance for 2019* [online]. The University of Texas at Austin, May 2020. Available at: https://www.gps.gov/systems/gps/performance/2019-GPS-SPS-performance-analysis.pdf.

[50] 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/.

[51] Rosebrock, A. YOLO and Tiny-YOLO object detection on the Raspberry Pi and Movidius NCS. *PyImageSearch* [online], 27. Jan 2020 [cit. 2021-05-13]. Available at: https://www.pyimagesearch.com/2020/01/27/yolo-and-tiny-yolo-object-detection-on-the-raspberry-pi-and-movidius-ncs/.

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

[53] 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.

[54] 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.

[55] 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.

[56] 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.

[57] *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.

[58] 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/.

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

[60] Using a standard USB webcam. *Documentation, Usage – Webcams* [online]. [cit. 2020-05-25]. Available at: https://www.raspberrypi.org/documentation/usage/webcams/.

[61] 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.

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

[63] Viola, P. and Jones, M. Rapid object detection using a boosted cascade of simple features. In: *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001.* 2001, vol. 1. DOI: 10.1109/CVPR.2001.990517.

[64] 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.

[65] 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.

[66] Zou, Z., Shi, Z., Guo, Y. and Ye, J. *Object Detection in 20 Years: A Survey*. 2019. Available at: https://arxiv.org/pdf/1905.05055.pdf.

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
                          [-t MAXSPEED] [-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
  -t MAXSPEED, --maxspeed MAXSPEED
                        maximum allowed speed tolerance for the frame to be
                        processed
  -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
- FFmpeg 3.4.8
- 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