TECHNICAL UNIVERSITY "GHEORGHE ASACHI" OF IAȘI

FACULTY OF AUTOMATIC CONTROL AND COMPUTER ENGINEERING

MASTER THESIS

Real-time person re-identification in a robotic solution

by

Eng. **Popovici Mircea-Alexandru**

Coordinated by

Lect. **Ostafi Silviu Florin**

Prof. **Montijano Muñoz Eduardo**

Session: June, 2022

TECHNICAL UNIVERSITY ”GHEORGHE ASACHI” OF IAȘI
FACULTY OF AUTOMATIC CONTROL AND COMPUTER ENGINEERING
DEPARTMENT OF AUTOMATIC CONTROL AND APPLIED INFORMATICS

Real-time person re-identification in a robotic solution

Eng. **Popovici Mircea-Alexandru**

Session: June, 2022

Coordinators:

Lect. **Ostafi Silviu Florin**, Prof. **Montijano Muñoz Eduardo**

Acknowledgements

I would like to thank to Lect. Ostafi Silviu Florin, Prof. Burlacu Adrian, Prof. Montijano Muñoz Eduardo and Prof. Tardioli Danilo for all the time and energy they invested in me and in this project, for sharing their wisdom and for guiding me at every step.

I would also like to express my gratitude towards my colleagues at the University of Zaragoza for welcoming me with open arms, for teaching me about their culture and for bearing with me every time I asked them to act in another video for my testing data sets. Many thanks to PhD. student and colleague Huștiu Sofia, for her sisterly care and for helping me whenever I would be at an impasse writing this thesis, and to BSc. Stoian Victor for frequently checking in on me during this period that has arguably been one of the most chaotic of my life.

Finally, I want to thank my parents for instilling in me the love of knowledge, for their unconditional love and support, and for celebrating with me every victory. Everything I achieve in this life is in no small part their merit as well.

Acronyms

AI Artificial Intelligence 9, 12

AIN Automatically searched Instance Normalisation 17

ANN Artificial Neural Network 12, 13

CNN Convolutional Neural Network 12, 13, 16, 26, 33, 35, 38, 44

COCO Common Objects in Context 15

CPU Central Processing Unit 44

DNN Deep Neural Network 12, 13, 26

FPS frames per second 15, 33

GFLOP giga floating-point operation 15, 17

GPU Graphics Processing Unit 22, 44

HD High-Definition 22, 34

IMU Inertial Measurement Unit 22

IOU intersection over union 14

JPEG Joint Photographic Experts Group 24–26, 33

LiDAR Laser Imaging Detection and Ranging 21

LTS long-term support 21, 23

mAP mean average precision 15

NUC Next Unit of Computing vii, 21–23, 44

OS Operating System 21, 23

OSNet Omni-Scale Network vii, 13, 16, 17, 27, 33, 35, 38, 44, 45

Pascal VOC The PASCAL Visual Object Classes 15

PC Personal Computer vii, 10, 21, 22

PNG Portable Network Graphics 25

RAM Random-Access Memory 22

RGB Red-Green-Blue 22, 33

ROS Robotic Operating System 21, 23–25, 33, 44

SDK Software Development Kit 25

TUIAȘI Technical University of Iași 16

YOLO You Only Look Once vii, 13–15, 25–28, 33–35, 39, 44, 45

Table of contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

A little over a decade ago, *AI (Artificial Intelligence)* and *Computer Vision* still were terms one would hear either in the high echelons of the academic society, or in science-fiction works, with very limited access for the everyday layman. But, being able to interact with machines in similar ways to how we interact with each other, having machines adopt human-like behaviours and being able to trust them with tasks and decisions that would otherwise require the time and patience of professionals have always been very attractive ideas. That is why, since then, the research community banded together and furthered the collective knowledge in this fields, and the results are incredible.

We now use AI everywhere. We use these tools as diagnosticians [4] or psychologists [5], we use them in manufacturing systems [6] too. And when we just want to have a good time, we chat with AI systems [7], we have them beat world champions in classic games [8], or even generate art [9]. For a while now we have started tackling problems of computer vision using AI and the proof is there every time we take a picture on our phones and we apply a filter. Object detection and tracking, object re-identification, these are just some of the classic problems in the field, for which we find increasingly robust solutions with each passing year.

So we keep formulating harder problems all the time, forever trying to push the limits. And in recent years, one such problem, that of multi-camera multi-tracking has become a hot topic in the field. Effectively, the purpose in solving this is to obtain systems capable of robustly tracking a –usually unknown– number of targets using a –normally reduced– number of cameras. Such systems have already emerged, for tracking cars and their trajectories using street cameras [10] or for monitoring wildlife [11]. Of course, a more interesting problem is that of tracking human targets, and most importantly, doing so while also recognising their identities. Significant amounts of work have gone into this, with dedicated datasets being created [12], as well as actual systems to tackle this [13, 14, 15, 16].

However, most existent studies use fixed camera surveillance systems, with all the data from them being processed on one, central computer. And because problems in this field are usually highly computationally taxing, the computers used are equipped with the latest hardware. Some of the more modern systems aim to function in a distributed manner [16], but they are still deployed on highly capable hardware setups. But multi-camera multi-target tracking would arguably be an interesting capability for teams of robots as well, and especially teams of mobile robots, which often lack the powerful hardware found on PCs (Personal Computers).

Towards this idea, this thesis proposes a pipeline for person detection and re-identification (hereafter referred to as *re-ID*), deployed on a mobile robotic platform with limited resources. As the effort to provide more accurate solutions for the problem continues, this project aims to fill a gap, and focuses on proving that existent, modern options can be adapted to work with limited resources in live scenarios, even without excessively sacrificing accuracy.

1.2 Project's context

All of the work done towards this project has taken place within the *Robotics, Perception and Real-time* (RoPeRT) research group, in the *Institute of Investigation in Engineering of Aragon* (I3A), under the authority of the *University of Zaragoza*, in Zaragoza, Spain.

The research group has been, in recent years, working on a multi-robot multi-camera distributed tracking system for human targets. In this direction, some researchers from the group laid the foundation for the system, albeit using a limited, trivial method for the problem of person re-identification [17, 18]. Other researchers in the same group have studied this problem in more detail [16], but have done so in fixed environments that have powerful hardware at their disposal.

The project making the subject of this thesis takes the findings and proposed ideas on solving the re-ID problem and uses them to replace the old method in the multi-robot system solution. The source code is made publicly available on GitHub [1].

1.3 Summary of thesis structure

This thesis is divided into five chapters, with the next ones describing, in order:

1. the theory behind the concepts used in the projects
2. the way in which the theoretical concepts are transposed to reality, as well as the equipment and tools needed for that

[1] https://github.com/mircea98ro/pedestrian_detector_distributed_assignment.git

3. an analysis on the performance of the prototype setup, from multiple points of view

4. conclusions and future directions of expansion for the project

Chapter 2

Theoretical considerations

This project proposes a working pipeline for use in a robotic system towards the purpose of person re-ID. But this problem is not atomic. In fact, its name focuses just on a part of the entire equation. A more detailed look on it reveals the following sub-problems that need to be solved to achieve person re-ID robustly:

1. Person detection in a live-feed
2. Generating descriptions of the detected people based on their features
3. Associating identities to people based on their features and previous knowledge
4. Filtering of the results

For detecting people and generating their descriptions, two specialised CNNs are proposed. The descriptions thus obtained must be associated to the ones previously known, which can translate into a classic assignment problem for which the Hungarian Algorithm is employed. Finally, the results can be processed by a Kalman filter. This chapter will elaborate on how this tools work and why they are fit for the problem we are looking to solve.

2.1 Convolutional Neural Networks (CNNs)

In the field of AI (Artificial Intelligence), CNNs are a particular type of DNNs (Deep Neural Networks), or ANNs (Artificial Neural Networks) with multiple, hidden layers, predominantly used in problems involving image processing. [19, 1]

The key difference to their generic counterparts is that they employ convolutional layers constructed in a very specific manner. Each neuron processes just a localised fraction of the entire available data on its level, as seen in Figure 2.1, thus the neurons in these layers are not fully-connected. The features extracted through convolution are down-sampled using pooling layers.

The data thus obtained can be processed further through more convolution and down-sampling depending on the specific architecture, but they eventually are passed through the fully-connected, ANN-like layers to compile, generally speaking, a descriptor of the initial input. Through this method, CNNs use fewer parameters to process input data than generic DNNs, with the magnitude of this difference becoming increasingly significant as the size of the input data increases.

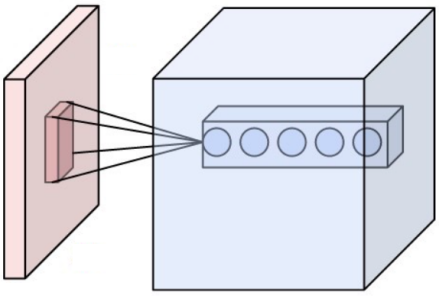


Figure 2.1: Neurons from multiple layers applying different filters through convolution on a targeted window of the input data [1]

After 2012, when Alex Krizhevsky proved the effectiveness of his own AlexNet [20] (Figure 2.2) in computer vision tasks, interest in the field soared, resulting in the creation of various new CNNs, some of which tackle very specific problems. Of these, this project uses YOLO (You Only Look Once) [2] for object detection, and OSNet (Omni-Scale Network) [21] for person re-identification.

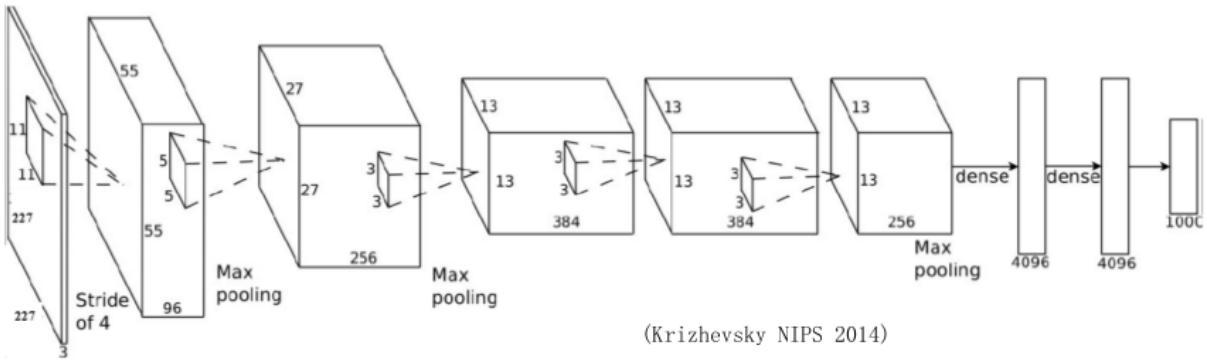


Figure 2.2: Alex Krizhevsky's AlexNet architecture [1]

2.1.1 You Only Look Once (YOLO)

YOLO is the name offered to a family of CNNs specialised in object detection. Currently the project is on its fifth major iteration, with each new generation of models incorporating more and more modern ideas. The project described in this thesis makes use of a pre-trained model available since the third iteration, colloquially known as YOLOv3.

As described in the original paper [2], YOLO offers a completely different approach to what previously available CNNs were implementing. The initially proposed architecture consists of twenty-four convolutional layers, followed by two fully-connected ones. The convolutional layers start processing images at a full scale of 448×448 pixels, with following layers operating on gradually smaller feature spaces, scaled to size by six max-pooling layers placed in between the convolutional ones.

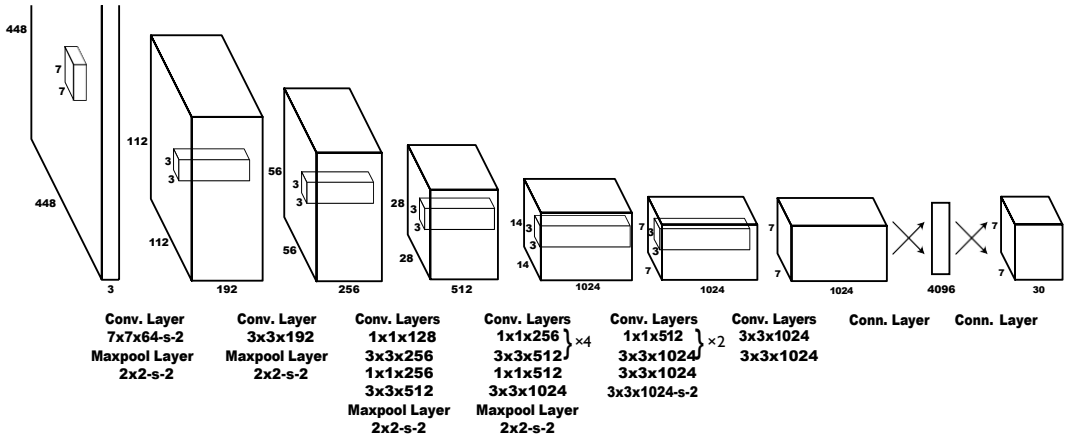


Figure 2.3: YOLO (You Only Look Once) original architecture [2]

YOLO returns results for images as a whole, but processes them by dividing the input into smaller areas in an $S \times S$ cell grid. If a cell contains the center of an object, it will be its job to properly detect the object. The network predicts B bounding boxes for every cell in the grid. These boxes are each represented by five normalised parameters: (x, y, w, h, c), with (x, y) being the coordinates of the center point of the bounding box, relative to the upper-left corner of the grid cell that contains it, (w, h) being the width and height relative to the entire image, and c being the confidence score of the box which is defined as the product of the IOU (intersection over union) metric and the predicted probability that the box actually contains an object.

$$c = Pr(Object) \cdot IOU^{truth}_{prediction} \tag{2.1}$$

Then, each cell —not box—, is assigned probability scores for the classes of objects for which the model used is trained. Each model available offers a number of labeled classes of objects (e.g. "dog", "person", "car", etc.) for which the last, fully-connected layers compute the conditional probability scores $Pr(Class_i|Object)$.

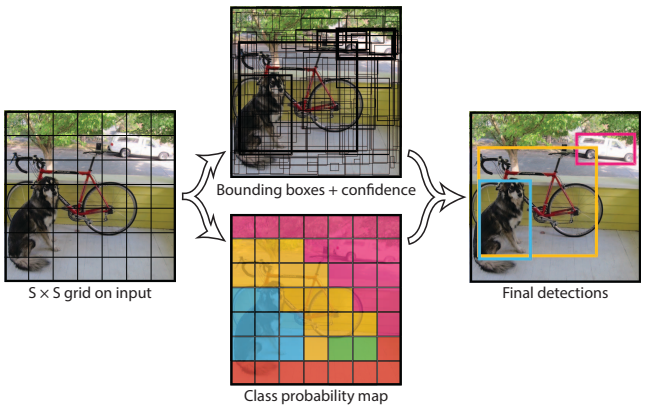


Figure 2.4: YOLO combining box prediction with cell-wise classification for object detection. Thicker means higher confidence score, color means class has the best probability score [2]

In the ideal case, a box encasing an object of type i would be characterised by:

1. a high confidence score c
2. high probability scores for its type's class $Pr(Class_i|Object)$ in the cells over which it spans

as exemplified in Figure 2.4.

The road from the original version to the third iteration of the network brought a number of interesting changes [22, 23] to the implementation of the network, to the metrics it is evaluated against and to the training methods. Some of the more relevant changes are:

- Predicting box coordinates is now done using pre-trained dimension clusters.
- Convolutional layers are resized.
- Multi-scale training is introduced, meaning the network is not resizing the images used in training to one fixed size.
- The Darknet-53 architecture [23], featuring fifty-three convolutional layers is now used. Compared to its predecessor Darknet-19 [22], it has half the FPS (frames per second), but, compared to the state-of-the-art ResNet-101 and ResNet-152 architectures [24], it offers $1.5\text{x} - 2\text{x}$ the speed with similar or better accuracy.
- Multi-label classification is introduced, meaning that now all classification scores are considered, not just the highest two of them. This helps process small objects better, as they often share their cells with other, larger objects.

The training of the network is done in two parts, one for classification, and one for detection. For classification, the author uses the ImageNet 1000 dataset, while detection is trained using datasets annotated in the Pascal VOC (The PASCAL Visual Object Classes) and COCO (Common Objects in Context) styles. The version used in this project, *YOLOv3-tiny*, uses images of $416\text{x}416$ pixels; it was trained on the COCO dataset and boasts a mAP (mean average precision) of 33.1% over the developer's custom testing dataset, with an FPS of 220. While the original *YOLOv3* network needs 65.86 GFLOPs (giga floating-point operations) and almost 62 million parameters, the reduced version needs only 5.56 GFLOPs and a little over 8.8 million parameters to run.

2.1.2 Omni-Scale Network (OSNet)

YOLO does a great job at recognising objects and their type, but it does not go deeper than that, given that it does not allow differentiation between different instances of the same class of objects. For the problem of re-id a particular approach is considered. Proposed in 2019, OSNet [21] is the proof-of-concept network for the

new paradigm it utilises, that of *omni-scale feature learning*, which allows it to consider features of smaller size nested into features of larger scale and aggregate them together, as seen in Figure 2.5. In doing so, the network is able to associate features on all scales which provides more accurate contexts for generating descriptors. For example, where other CNNs that tackle the problem of re-ID would identify a young engineer wearing a white t-shirt and, separately, a TUIAȘI (Technical University of Iași) logo, OSNet would associate the two and identify a young engineer wearing a white shirt *with* the university logo on it.

To achieve this, OSNet uses what it calls *Lite* 3x3 *convolution*, or *Depth-wise convolution* instead of standard 3x3 convolution in its layers, which requires fewer parameters and has lower computational costs than the standard one. More specifically, if the input is $x \in \mathbb{R}^{h \times w \times c}$, with height h, width w and channel width c, considering c' as the output channel width and k as the convolution mask size, then a standard approach would have a cost of $h \cdot w \cdot k^2 \cdot c \cdot c'$ and need $k^2 \cdot c \cdot c'$ parameters. The *Lite* approach reduces these to a cost of $h \cdot w \cdot (k^2 + c) \cdot c'$ and a number of $(k^2 + c) \cdot c'$ parameters.

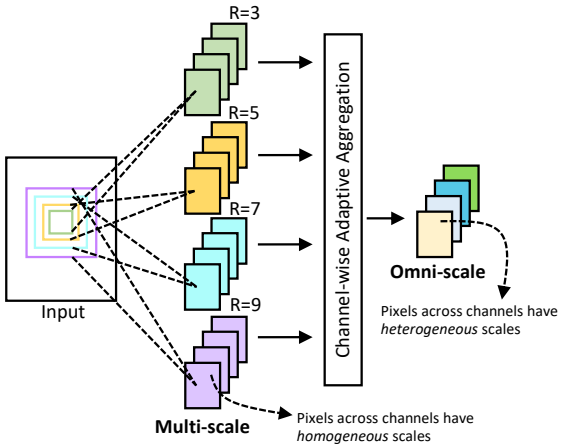
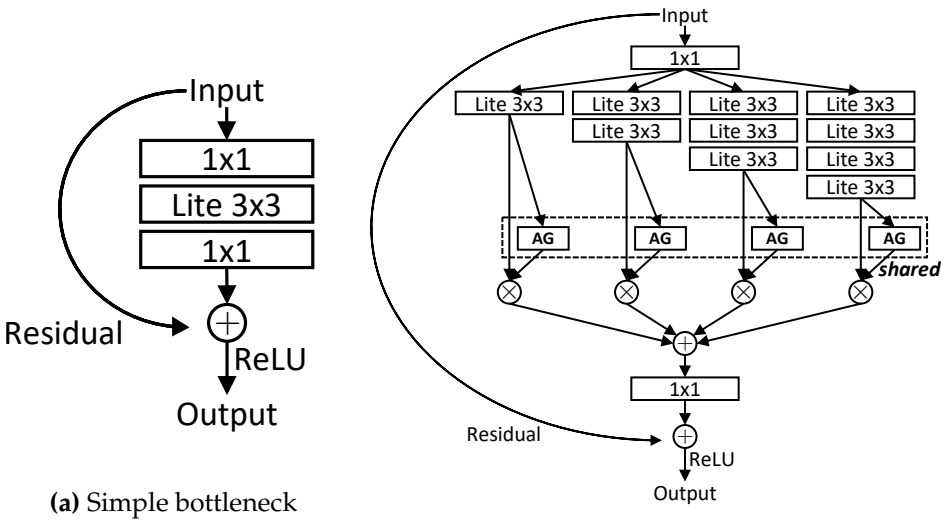


Figure 2.5: OSNet's association of features on multiple scales



(a) Simple bottleneck

(b) OSNet bottleneck

Figure 2.6: Bottleneck type comparisons.

The network uses these *Lite* layers as part of its novel *omni-scale residual blocks* with bottleneck, whose construction, as the name suggests, allows for multi-scale feature learning. A constructive comparison between baseline bottlenecks and the newly

proposed ones is available in Figure 2.6.

The model used in this project is a newer variation of OSNet called *OSNet AIN x0.25* [25], which stands for OSNet with Automatically searched Instance Normalisation, scaled to one quarter of the original size. While the original OSNet architecture, already lightweight compared to its rivals, has 2.2 million parameters and needs 2.7 GFLOPs to process one image, the version used here has a measly 203568 parameters, with less than 0.3 GFLOPs per image, and it still offers competitive results.

2.2 Hungarian Algorithm

The Hungarian Algorithm [26] was first published under this name in 1955 and has since proved to be a valuable tool in solving assignment problems, substantially reducing their computational complexity.

The original assignment problem proposed by Harold Kuhn in his publications considered n available individuals for n jobs, and a rating matrix with positive integers $R = (r_{ij})$, with $i = \overline{1,n}$ and $j = \overline{1,n}$. An assignment can be thought of as a permutation

$$\begin{pmatrix} 1 & 2 & \cdots & n \\ j_1 & j_2 & \cdots & j_n \end{pmatrix}$$

where $j_x = j_y \Leftrightarrow x = y$, such that j_i represents the job assigned to person i. The objective is to find an assignment which maximises the sum $r_{1j_1} + r_{2j_2} + \cdots + r_{nj_n}$, using the available individuals to obtain the highest rating.

Now, the basic approach is to try all of the possible permutations and choose an optimal one. But there are $n!$ such variations [27], which translate into a big-O complexity of $O(n!)$. Using the original version of the proposed algorithm, the problem can be solved in a much faster $O(n^4)$, with more modern variations reducing this to $O(n^3)$ [28].

The problem tackled in this project is slightly different from the original one, and is discussed in Kuhn's second volume on the subject [29]. In this variation, the ratings are replaced by costs, with the new purpose being to find the assignment with the minimum total cost. As such, given a cost matrix $C = (c_{ij}) \in \mathbb{R}^{+}_{n \times n}$ the algorithm proposes the following steps:

Step 1 Subtract row minima
For each row, find the lowest element and subtract it from each element in that row.

Step 2 Subtract column minima
For each column find the lowest element and subtract it from each element in that column.

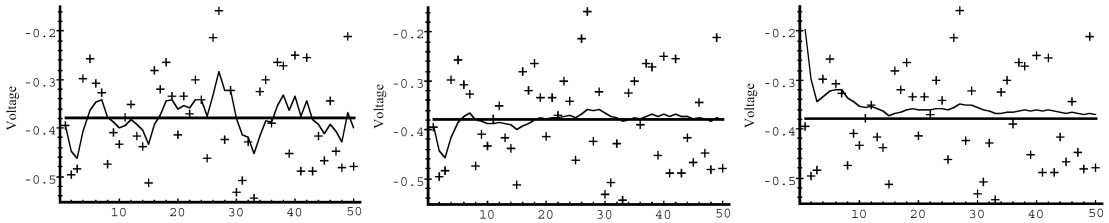
Step 3 Cover all zeroes with a minimum number of lines
In the resulting cost matrix, select a minimum number of lines and column such that all zeroes are selected. If n selections were necessary, then go to the last step. Else, go to the next step.

Step 4 Create additional zeros
Find the smallest element that is not selected at **Step 3**. Subtract it from all elements outside of the selection, and add it to all the elements that are selected both by a row-wise selection and a column-wise one. Go back to the previous step.

Step 5 Return results
If on **Step 3** n selections were made, then proceed either row-by-row or column-by-column to select one null item $c^*_{ij} = 0$ from the obtained matrix on each iteration, thus assigning worker i to job j.

Several improvements have been added to the original algorithm over the years; this project uses a variation that is widely accepted as one of the best optimised ones, called the *Jonker-Volgenant algorithm without initialization* as proposed in [28].

2.3 Kalman Filter



(a) $R = 0.0001$ **(b)** $R = 0.01$ **(c)** $R = 1$

Figure 2.7: Example of Kalman Filter results with different fixed values for the covariance parameter of measurement noise, applied on a set of voltage readings [3]

Kalman filtering is a popular algorithm stemming from the field of statistics, offering a solution to the discrete-linear-filtering problem [3]. More specifically, it is used to estimate the state $x \in \mathbb{R}^{n\times 1}$ of a discrete process defined by the linear stochastic difference equation:

$$x(k+1) = A_k x(k) + Bu(k) + w(k), \tag{2.2}$$

with $w(k) \in \mathbb{R}^{n\times 1}$ as the process noise, $u(k) \in \mathbb{R}^{l\times 1}$ as the control input, $(A)_{n\times n}$ as the state matrix, and $(B)_{n\times l}$ as the input matrix. It also considers measurements $z \in \mathbb{R}^{m\times 1}$, following:

$$z(k) = H_k x(k) + v(k), \tag{2.3}$$

with $v(k) \in \mathbb{R}^{m\times 1}$ being the measurement noise, and $(H)_{m\times n}$ as the observation matrix. The two variables w and v are assumed to be white noise, following normal distributions, and independent of each other, with:

$$w \sim N(0, Q), v \sim N(0, R), \tag{2.4}$$

where Q and R are the covariance matrices for the process noise and for the observation noise respectively.

The algorithm that this filter implements can be seen as a cycle of two main operations, namely *prediction* (or *time update*) and *correction* (or *measurement update*). Then, (2.2) can be thought of as a *prediction equation*, and (2.3) can be considered a *correction equation*.

In the prediction phase, the predicted, or *a priori* estimation of the state is

$$\hat{x}(k+1|k) = A_k\hat{x}(k|k) + B_k u(k), \tag{2.5}$$

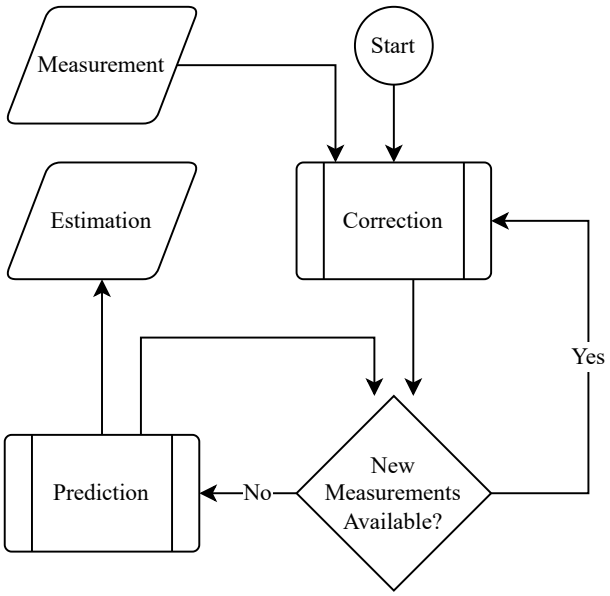


Figure 2.8: Kalman Filter process loop

and the *a priori* estimation of the error covariance is updated to

$$P_{k+1|k} = A_k P_k A_k^{\mathsf{T}} + Q_k. \tag{2.6}$$

The correction phase begins with computing the optimal Kalman gain

$$K_k = \frac{P_{k|k-1}H_k^{\mathsf{T}}}{H_k P_{k|k-1} H_k^{\mathsf{T}} + R_k}, \tag{2.7}$$

then goes to generate an *a posteriori* state estimate which incorporates the newly available measurement

$$\hat{x}(k) = \hat{x}(k|k-1) + K_k(z(k) - H_k\hat{x}(k|k-1)), \tag{2.8}$$

and finally ends by obtaining an *a posteriori* error covariance estimate via

$$P_{k|k} = (I - K_k H_k)P_{k|k-1}. \tag{2.9}$$

The steps of each Kalman filter phase are summarised in Figure 2.9.

To provide useful results, the filter needs to be adapted to the context it is used into, by setting appropriate values for the covariance parameters Q and R.

The effect of the method is that it filters noises, creating smoother results than those generated by raw measurements. Lastly a Kalman Filter can compensate for a temporary lack of new input measurements and provide useful predictions of what the state of the system should be.

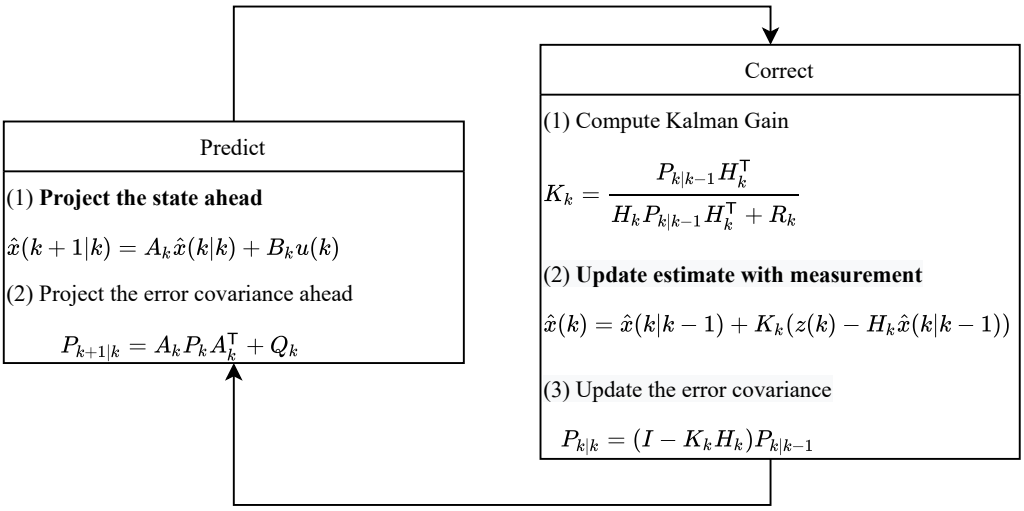


Figure 2.9: The Kalman Filter phases detailed

Chapter 3

Implementation

As initially stated, the aim of this project is to build a functioning pipeline, capable of processing live feed from a camera and provide information on the number of people seen in the images. This chapter describes the pipeline step-by-step, with details on the technologies used and the implementation of several facilities.

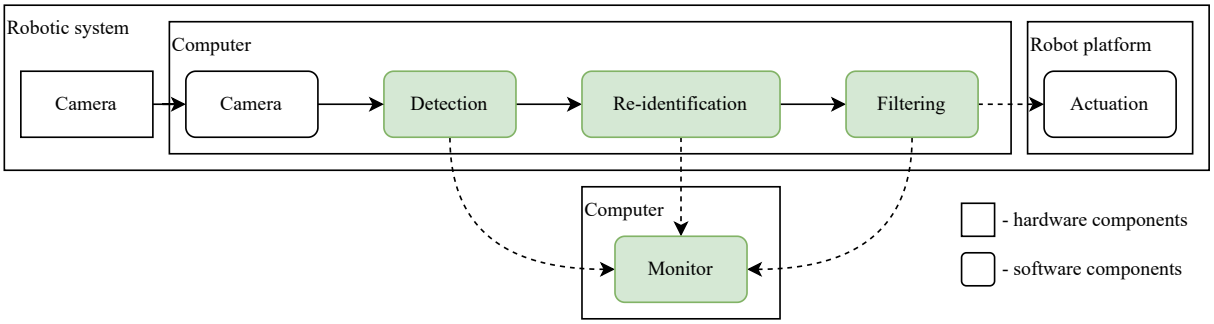


Figure 3.1: Proposed system architecture. Coloured components represent this thesis' contribution

Firstly, as this project is part of a bigger one aiming for distributed, multi-camera, multi-target tracking using a team of mobile robots, the hardware, as well as some software components have to be the same or similar to those used throughout all parts of the project. More specifically, the proposed pipeline is deployed on Intel NUC (Next Unit of Computing) Mini-PCs (Personal Computers), which are fitted to Turtlebots and receiving sensory data through LiDAR (Laser Imaging Detection and Ranging) sensors and Intel RealSense D435i Depth Cameras. The OS (Operating System) of choice is *Ubuntu LTS (long-term support) 20.04*[1], with *Python 3*[2] based *ROS (Robotic Operating System)[30] Noetic*[3] running on it. Running on the ROS framework are four nodes responsible for particular sub-tasks of the main problem, as well as an additional node, running externally to the robotic system, used for monitoring and performance analysis purposes.

[1]https://releases.ubuntu.com/20.04/
[2]https://www.python.org/
[3]http://wiki.ros.org/noetic

The software packages through which these nodes are implemented, alongside with the analysis of the system's performance, represent the contribution of this thesis. Taking a look at the larger project, it is important to notice that the output of this pipeline serves as the input for other nodes, using the results of the identification to help a team of robots move around following a number of human targets.

3.1 Hardware

3.1.1 Intel NUC Mini-PC

The core hardware on which the project is tested is Intel's NUC6i5SYH NUC (Next Unit of Computing) Mini-PC (Personal Computer)[4]. The entire range offers barebone, low-power, small form-factor computers, enough to allow them to be fit to mobile robot platforms. However, the trade-off is that NUCs have limited capabilities compared to full-fledged modern computers, the biggest drawback being the absence of a dedicated GPU (Graphics Processing Unit). The chosen model in particular, provides 8GB of RAM and relies on the low-power, 14nm Intel Core i5-6260U silicon, a modest dual-core processor from 2015, running nominally at 1.8GHz, with a maximum frequency of 2.9GHz, 4MB of level 3 cache, 512KB of level 2 cache and 128KB of level 1 cache. The processor boasts an integrated GPU, but as will be detailed upon later on, it is not used at any step.

Figure 3.2: Intel NUC Mini-PC

3.1.2 Intel RealSense Depth Camera

The robotic system includes a stereo camera, particularly an Intel RealSense D435i Depth Camera[5]. The device is fitted with a Full-HD (1920x1080) RGB camera as well as a HD (1280x720) stereoscopic depth-measuring system, and lastly with a full 6-axis IMU (Inertial Measurement Unit). The system described in this thesis uses only the RGB information, which the camera offers at a maximum frequency of 30Hz.

Figure 3.3: Intel RealSense D435i Depth Camera, front view

[4]https://ark.intel.com/content/www/us/en/ark/products/89190/intel-nuc-kit-nuc6i5syh.html
[5]https://www.intelrealsense.com/depth-camera-d435i/

3.2 Software tools

3.2.1 Ubuntu Linux

The Intel NUC computer fitted on-board the robot uses Ubuntu 20.04 as its OS (Operating System). Ubuntu is a highly stable, and more importantly popular distribution of Linux, but with LTS (long-term support) version 22 just recently being released, version 20 still remains preferable. Its popularity makes it the default choice of most developers, which translates into a plethora of tools being readily available for use on it, including the vital ones as described in the next parts. Lastly, Unix-based systems such as this one are usually lightweight and offer highly-granular control, with certain configurations running with minimal user interface and background services. For this particular application, that is important as it frees up more resources to be used by the image processing pipeline.

3.2.2 Robot Operating System

As its creators admit themselves, ROS is not really an operating system, but rather a deployable framework that offers a backbone for developing robotic applications [30]. The main facilities and advantages of ROS are:

- being free and open-source, with a huge community publishing packages that add to its range of functionalities
- being multi-lingual, using Python, C++, LISP, Octave, or any combination of them at the same time
- offering a simple-to-use, standardised, peer-to-peer communication protocol which can be easily built upon to cover any application's needs
- organising code projects into modular packages linked through hierarchies of dependencies
- offering robust simulation environments
- offering support for a wide range of devices, enabling developers to focus on high-level problems rather than on having to write their own drivers

ROS divides the elements of the workflow into *nodes*, *messages* and *services*, with several other auxiliary components such as the parameter server, which offers an easy way to share parameters between multiple nodes and update them on-line. The main elements, the nodes, are the executable scripts which implement application-specific functionalities. They can publish messages on *topics*, or subscribe to existent topics to receive streams of messages in turn. Finally, services offer the option to interact with

the nodes during execution, for example to trigger a reset of a device or somehow alter the behaviour of a specific part of the program.

While in some applications devices need not interact, in this one, at least for testing purposes, the monitoring of performance across the proposed pipeline is ran on a computer other than the ones fitted on the robots. Similarly, in the larger project to which this application pertains, teams of robots work together in a distributed manner, but to do so they must be configured on the same ROS network. For this, the framework offers a central *name service*, or rather it uses a master node which helps all nodes establish their peer-to-peer connections.

Finally, the version used here, *Noetic*, is a mature distribution of ROS, and it is the first one to fully migrate from Python 2 to Python 3, which most of the packages used in this project require as well.

3.3 Pipeline implementation

As seen previously in Figure 3.1 and presently in Figure 3.4, the software nodes running on a robot are connected to one another in a daisy-chain fashion. This means that each node is dependent on the input provided by the previous one by means of ROS messages. The detection node receives live frames from the camera node through standard messages available in the framework, but it outputs customised messages with a specific structure. The same format will be kept for both the inputs and the outputs of the re-ID and filtering nodes, each one updating the data existent in their inputs with specific new information as per their functions and passing everything further down.

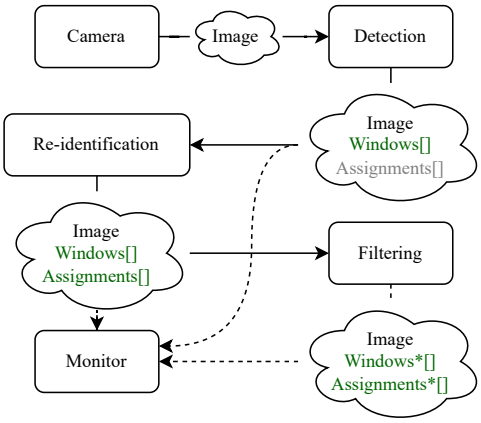


Figure 3.4: Message flow through the pipeline

These custom messages are built to ensure that all necessary data is provided, that all nodes process the same information (e.g. the re-ID node considers the same image as the one the detection node provides boxes for), and that there is no time or processing power expended on unnecessary data being moved around. These messages, whose type will henceforth be referred to as *ProcessData*, follow the structure presented below:

- **Header** ***h*** - Standard metadata container that informs on the properties of the message. Contains a time-stamp field used in the performance analysis.
- **CompressedImage** ***img*** - Contains a live-feed image compressed into a standard JPEG format.

- **ProcessWindow[]** ***data*** - Contains an array of structures with details about the detected windows.
  - **Window** ***window*** - Contains a structure with the coordinates and size of a detected window in the scene. It includes four fields: x (horizontal coordinate of window center point relative to frame), y (vertical coordinate of window center point relative to frame) , h (window height), w (window width).
  - **Int** ***assignment*** - A positive integer whose value $a = x$ signifies that the box it's associated with is identified as the x-th person in the available database.
- **Time** ***stamp*** - A standard time field that stores the time-stamp of the moment on which the image contained in the *img* field was received by the pipeline (so by the detection node).

Using the same message format for all nodes' outputs does not only make things easier to understand from a human perspective, but also simplifies the development of monitoring mechanisms, since data extraction is done in the same way from any level of the pipeline.

3.3.1 Image acquisition

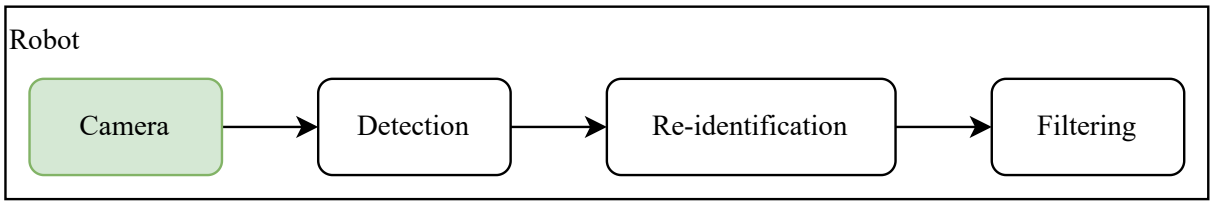


Figure 3.5: Camera node in the pipeline

Before processing anything,the system needs input data. To this end, ROS provides a package that acts as a driver, bridging the gap with Intel's RealSense 2 SDK[6], in turn enabling access to the full facilities of the D435i Depth Camera. Through this users can set all of the parameters of the camera at run-time, including its image acquisition frequency and its output's resolution. The live feed of the camera becomes available in raw or compressed (JPEG, PNG or Theora) through ROS topics to which any node can subscribe.

3.3.2 The detection node

The purpose of the detection node is to process live-feed images by using YOLO, returning information about the placement of people in the scene. The framework

[6]https://www.intelrealsense.com/sdk-2/

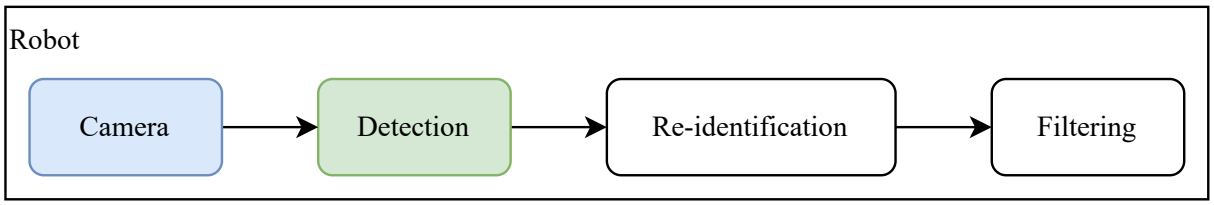


Figure 3.6: Detection node in the pipeline

which allows running images through YOLO is offered by the OpenCV[7] [31] package, through its DNN module.

As seen in Figure 3.4, the detection node takes input from the camera node. Because JPEG images are much smaller in size to those uncompressed, while maintaining good quality, the node requests compressed images from the camera node, eventually converting them to the "bgr8" encoding used by OpenCV. This helps speed up transfer times and decrease latency between the camera and the detection node.

As mentioned before, a reduced version of YOLOv3 is loaded into the DNN module . This particular version can categorise its detections into eighty types of objects, with the first class being that of "person". So, each processing done by the CNN returns a feature vector $f \in \mathbb{R}_{+}^{85\times 1}$. Of these, the first five elements contain the parameters (x, y, w, h, c) as described in subsection 2.1.1. The remaining eighty values represent the conditional probability scores $Pr(Class_i|Object)$, and the only interest in them is that the probability for the class "person" is the highest of them all, and above a fixed threshold.

After being identified by YOLO, the boxes are sorted and selected in an attempt to keep exactly the best n of them, with n being the known number of targets the robot expects to find in each image. This is done by first removing boxes that do not contain persons, then by sorting the remaining ones by their confidence score, in descending order. The boxes with lower confidence scores than the set threshold are also removed, and one more filtering eliminates lower confidence boxes overlapping over higher confidence ones.

(a) 2/3 targets detected **(b)** 3/3 targets detected **(c)** 2/3 targets detected

Figure 3.7: Three consecutive frames processed by the detection node

[7]https://opencv.org/

Notice! Frames are analysed independently from one another by the detection node, which means that there is always a chance that **a person that was correctly found by YOLO in one frame, might not be identified in the next one**, or the other way around, as exemplified in Figure 3.7.

3.3.3 The re-identification node

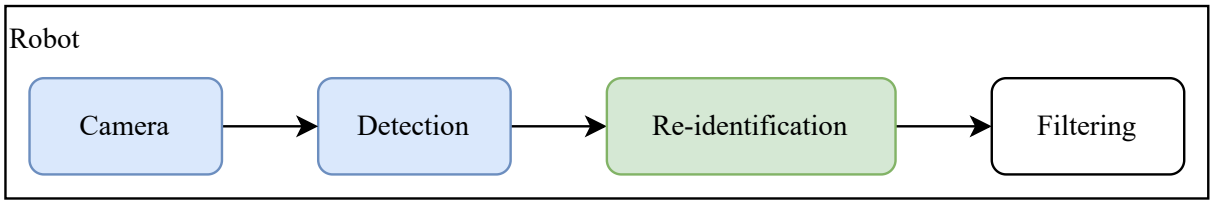
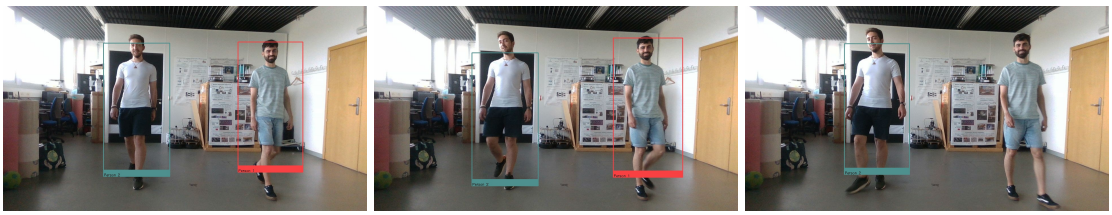


Figure 3.8: Re-identification node in the pipeline

The re-ID node is responsible for recognising the people in the windows provided by the detection node. It uses OpenCV for image processing, PyTorch[8] [32] and TorchReID[9] [33] for providing access to OSNet and lastly, the hungarian algorithm, as implemented in SciPy, to process OSNet's results.

Each window provided by the detection node is cropped from the original image and passed through the neural network. The node stores a gallery of descriptors for each recorded target person, and the descriptors obtained live are compared to the recorded collection, by means of computing the cosine distance between them and each of the pre-recorded descriptors. Having all the results, we consider the distance between a live-obtained descriptor and a stored gallery as the minimum distance between the live-obtained descriptor and any of the members of the gallery. Repeating this for every window in the input leaves us with a matrix of costs $C = (c_{ij}) \in \mathbb{R}_{+}^{n \times g}$, where n is the number of windows to identify, g is the number of recorded galleries/identifiable persons, and c_{ij} is the distance between the descriptor of the i-th window and the j-th gallery. Since each window contains a different person, and each gallery describes a different person, this turns into the assignment problem problem discussed in the second chapter, so the job of finding an optimal assignment falls to the hungarian algorithm.

Notice! The re-ID node only identifies persons detected by the detection node. Therefore, a fault in the results of the detection node will be noticeable in the results of the re-ID node too.



(a) 2/2 targets identified **(b)** 2/2 targets identified **(c)** 1/2 targets identified

Figure 3.9: Three consecutive frames processed by the detection and re-ID nodes

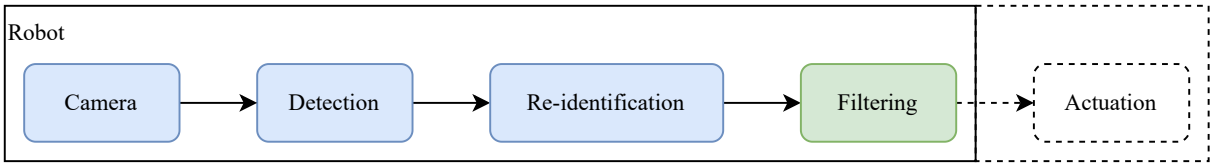


Figure 3.10: Filtering node in the pipeline

3.3.4 The filtering node

While the detection and the re-identification nodes are enough to implement the basic functionality, we see in Figure 3.7 and Figure 3.9 that their results are not entirely reliable. To address this, a new node is introduced, which implements Kalman Filters. It takes in account the results of the previous nodes and creates predictions based on them, with two objectives:

- Smooth the outputs from YOLO, as in the movement of the detected boxes across time
- Compensate for the absence of known targets from the input data by placing boxes for them artificially based on predictions

Kalman filters are always adapted to the properties of the system they predict upon. In this instance, the node uses several instances of the filter, one for each target person, and they predict the position of the center point of each target's bounding box. Considering all instances of the filter conceptually identical, it follows then naturally that the measurements used in the correction phase, as well as the output of the filter, should be a pair (x, y) of coordinates. Then we define the states of the filter as $[x, y, \dot{x}, \dot{y}]$. Finally, the components of the system follow:

$$\mathbf{x} = \begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \end{bmatrix}, A = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \mathbf{z} = \begin{bmatrix} x \\ y \end{bmatrix}, H = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

[8]https://pytorch.org/
[9]https://kaiyangzhou.github.io/deep-person-reid/index.html

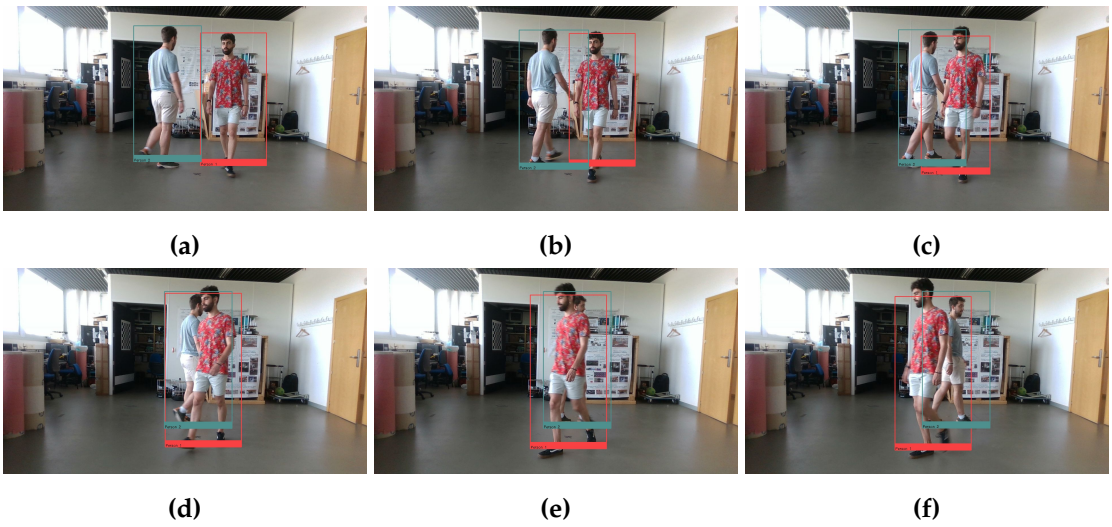


Figure 3.11: Crossing sequence, processed by the detection, re-ID and filter nodes

We set one more parameter, the assumed covariance of the process as $Q = \mathbf{I}_2 \cdot q$. The scaling factor q is chosen so that the filter can compensate for process uncertainties, and its tuning in this application is discussed in the next chapter.

With the filter instances initialised, the filter node reads data from the re-ID node. If a box in the received data is assigned to person i then the i-th filter instance goes through a correction phase using the available measurement. After all of the received data is used in this way, **all** of the filters go through a prediction phase, and the results are packed and sent further to serve as the final result of the pipeline. We expect predictions to compensate for errors over short spans of time, but if a filter does not get any data to go through a correction phase over a period longer than a fixed threshold, it goes inactive until new data is made available to it, which can be interpreted as the target being temporarily out of the frame.

3.3.5 The external monitor node

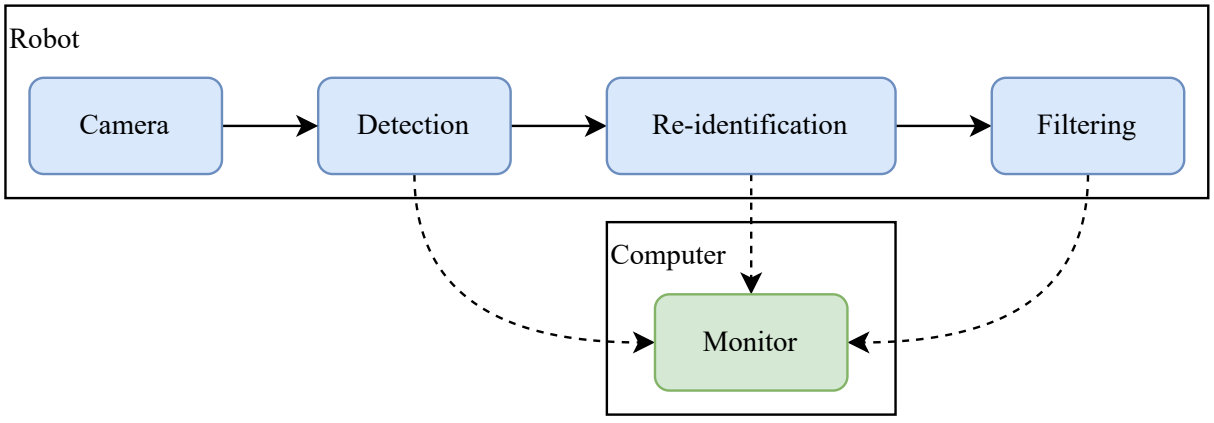


Figure 3.12: Monitor node

Although not serving a direct purpose in solving the problem of person re-ID,

the monitor node is important for debugging and testing purposes. This node can subscribe to any nodes from the pipeline and process their results, providing a way for human users to visualise them. At the same time, the monitoring node uses metadata from the messages it receives to publish both live and post-factum statistics on the functioning of the pipeline.

(a) Detection node output **(b)** Re-ID or filter node output

Figure 3.13: Pipeline node outputs processed into images by the monitor node

Live statistics are plotted using PlotJuggler and they provide the live evolution of stats. When a test ends and the monitor is stopped, it exports the average processing time and the quality metric in a number of file formats, for later review.

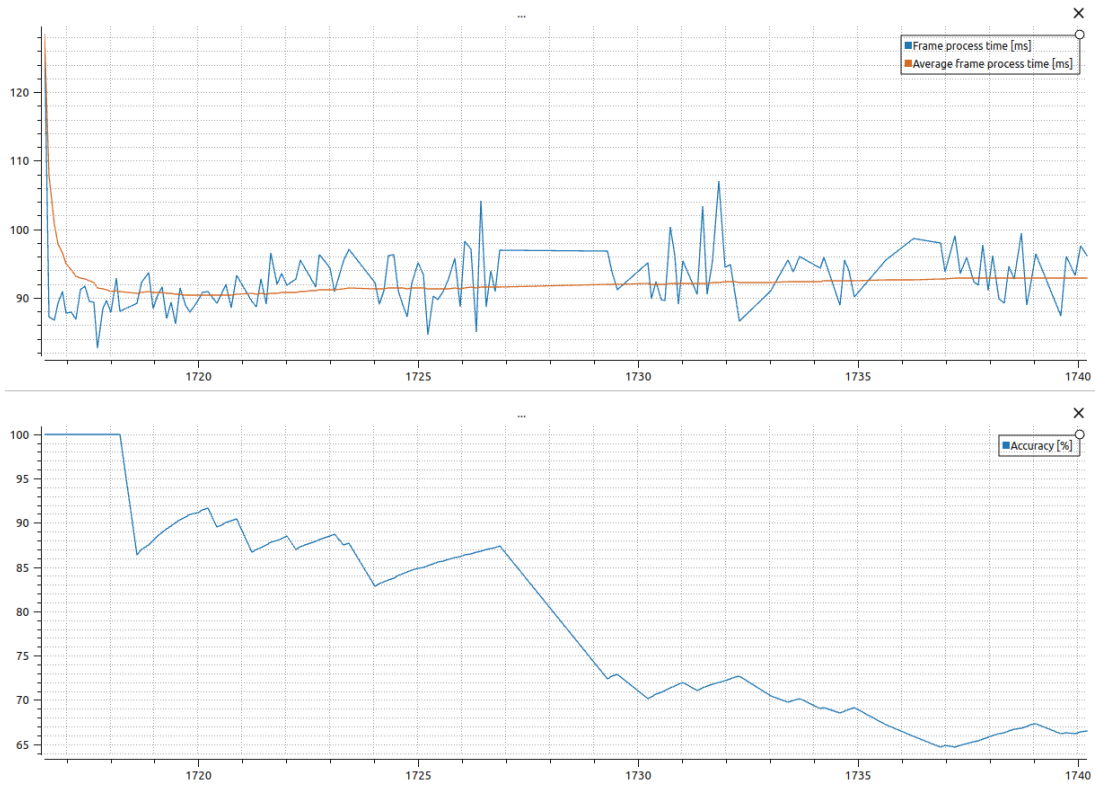


Figure 3.14: Live statistics plotted from monitor data

Chapter 4

Results

This chapter presents experimental results and the way they are processed. It gives an interpretation of the results, pointing to the strengths and weaknesses of the pipeline along the way.

4.1 Testing methodology

In the process of evaluating the pipeline, the logic step to begin with is that of establishing the metrics to be used and their relevance. With the metrics established, one can start making empiric or logic observations based on which to decide on the values of the system parameters to be used in the testing process. The following subsections will follow the order of this steps.

4.1.1 Metrics

To decide on the metrics, the focus of this project must be considered once more; that is to build a pipeline that can be deployed on a system with limited resources and perform as fast as possible.

To this effect, for each test, the individual processing time of each frame that goes –at least partially– through the pipeline is considered. A frame is processed by maximum three nodes. On each of these levels the time it takes to process the frame is taken as the interval between the image being received on **the detection node** and it being sent forward by **the node being tracked**. This means that the time measured on the re-ID includes the one measured on the detection node as well, with an analogue behaviour for the time measured on the filter node. Eventually, knowing all of these individual times at each stage, the **mean processing time** (μ) and the **standard deviation** (σ) can be computed automatically for each level of the pipeline.

But this is not enough. While it is important that the system runs fast, it is equally important that the system correctly detects and identifies people. To quantify this, four criteria are defined:

1. **For the detection node**. Consider t being the number of detected targets, and T the actual total number of targets in the frame.

   (a) **Average detection rate** - This is a quantitative measure of how many of the targets are detected –not identified–. It is computed as the average of the detection rates obtained on each frame. For a frame $i \in \{1 \dots n\}$, the detection rate is computed as:

   $$d_i = \frac{t_i}{T} \tag{4.1}$$

   The average rate is then:

   $$D = \frac{\sum_{i=1}^{n} d_i}{n} \tag{4.2}$$

2. **For the re-ID and filter nodes**. Four values are used to define the metrics:

   - **True positives** T_P - Number of people correctly detected and identified
   - **True negatives** T_N - Number of detected boxes that are correctly identified and thus **not** assigned to any person (e.g. detection points to a dog, so the box is not identified as containing any of the targets)
   - **False positives** F_P - Number of detected boxes that are incorrectly identified and assigned to people (e.g. a box containing Target 1 is assigned to Target 3, or a box containing a dog is assigned as one of the human targets)
   - **False negatives** F_N - Number of people either incorrectly **not** detected or **not** identified (e.g. target is not contained in any box, or it is contained but their box is not assigned to anyone)

   With these known, the metrics are:

   (a) **Accuracy** - This is the classic metric of correct results out of the total results obtained:

   $$A = \frac{T_P + T_N}{T_P + T_N + F_P + F_N} \tag{4.3}$$

   (b) **Precision** - This metric considers the proportion of correct identifications compared to all identifications made:

   $$P = \frac{T_P}{T_P + F_P} \tag{4.4}$$

   (c) **Recall** - This considers the correct identifications made compared to all the identifications that *should* have been made:

   $$R = \frac{T_P}{T_P + F_N} \tag{4.5}$$

The *average detection rate* is computed automatically and validated manually, because, not accounting for the rare bad detection, this is simply a matter of counting the number of boxes in the frame. The *accuracy*, *precision* and *recall* values are processed

entirely manually. That is due to the fact that the pipeline is tested on the real robotic platform, either on live video feed or on replays of ROS bag files containing live in-house recordings. Therefore, no labels are available to be used as the ground truth.

Together, the timing and performance metrics can offer a clear image on what the system is capable of, and they offer hints on the relationships between the nodes and how they influence each other.

4.1.2 Parameter tuning

The architecture proposed in this project contains three original nodes, and uses a third-party developed camera node as well, with each component introducing their own specific parameters. The large number of parameters and their intricate relationships with one another complicate the task of testing the system. For this thesis, most of the parameters are given fixed values and are omitted from the discussion.

Before going into detail about the specific parameters of each node, it is right to first mention the meta-parameters used by the three original nodes in the pipeline:

1. **Level** {raw, yolo, osnet, kalman} - Sets how far along the pipeline images should get for processing.

2. **Targets** $t \in \mathbb{Z}_+$ - For the synthetic tests presented in the thesis this parameter informs the nodes about the number of targets they system is tracking.

When there is only one target, the pipeline can be reduced to the level of detection as the problem of re-identification becomes trivial. Even for more than one target, the filter node is optional, the problem or re-ID being essentially solved without it. For the purpose of this thesis, the results presented are all computed with the entire pipeline running.

The number of targets is of great importance in the analysis of the performance. Both the processing time and the accuracy vary depending on this parameter, and the discussion on the results will be focused on this aspect.

Camera

The pipeline first solves the problem of image acquisition. The camera has several modules, but only the RGB one is used, so most of the parameters are of no importance. For all the results presented in the following parts, the device is set to provide compressed, JPEG images at a resolution of 1280x800 pixels, at a speed of 30 FPS. Testing multiple configurations has shown that changing these parameters does not produce relevant changes in the performance, and that is unsurprising considering the architecture of the pipeline.

The two CNNs used –YOLO and OSNet– are the only ones concerned by the dimensions of the live-feed, and they each resize the images they process to fixed sizes.

Then, the size of the original image affects virtually just the transfer times between nodes, but that is mitigated by the combination of high transfer speeds and the use of compression. So, the decision to use HD images over lower-resolution ones is mainly due to them offering more clarity to human users monitoring and visualising the results of the pipeline.

The frequency of the camera also has little influence. The detection node is the one who introduces images in the pipeline, and it ignores all received frames while it is busy, so that it always processes the most recent frame available. Then, the only concern is to avoid bottle-necking the detection node by providing images with too low a frequency. Essentially, the speed of the camera must be at least as high as that of the detection node. But just to stay on the safe side, similarly to the principle of Shannon's sampling theory, it is better to sample the camera at twice the speed of the detection node. Because the frequency on the detection component is variable, and one of the performance indices being studied, the most scenarios are covered by setting the frequency of the camera to the maximum value its hardware supports.

Detection

The detection node, which uses YOLO has a different story. This one sets three thresholds, related to the properties discussed in subsection 2.1.1:

1. **Minimum class score** 0.0/1.0 - Any box with a lower score is considered not to be a part of the required class 'person'. In this situation, setting it to 0 disables the check.

2. **Minimum box confidence score** 0.2/1.0 - Any box with a lower score is considered to be a false detection.

3. **Maximum overlap percentage** 0.8/1.0 - Any box that overlaps with another box (of higher confidence score) in a proportion above this threshold is considered redundant.

These are tuned based on empirical observations, to provide the best results, but there is a logic to choosing them so.

Each box has class probability scores, and the natural way of classifying a box is by assigning it to the class for which it presents the highest score. Therefore, as described in subsection 3.3.2, the detection node eliminates all boxes not classified as people. By setting the first parameter to 0, the algorithm allows all 'person' boxes to pass the first filter.

The second parameter is supposed to filter out those boxes that might be false alarms. But, testing has shown that using *yolov3-tiny*, as the number of targets increases, the average confidence score gets lower. While a value of 0.2 does not do enough to filter all false detections, it is arguably better to detect all the targets and

have some extra hits, than to only detect some of the targets. The job of refining this detection can fall to the re-ID node.

The overlap parameter is the one doing the heavy lifting. As seen in Figure 2.4, YOLO can return an immense number of boxes. But most of these overlap one another, so they can be filtered and eliminated. The parameter is set at 0.8, a high value, in order to help detect targets close to one another or the ones crossing one in front of the other. This helps in scenarios such as the one previously seen in Figure 3.11.

To sum up the observations, a strict detection node creates more *false negatives*, but eliminates false positives on the re-ID node, while a relaxed detection node reduces dramatically *false negatives*, but increases the chances of having *false positives* later down the line. We favour the latter option.

re-ID

The re-ID node uses OSNet which is pretty straightforward. Each window is cropped from the image, resized and passed through the network, returning a feature vector that is then compared by means of cosine distance with the available galleries. So, for OSNet this nodes uses:

1. **Window size** 128x256 - This fixes the size to which the windows are resized before processing.

2. **Resize windows** {false, true} - If set, the node first expands detected windows to match the wanted aspect ratio.

3. **Maximum distance** $0.6/1.0$ - Hard threshold that dictates the maximum distance between a feature vector and a gallery for which they can still be considered associated.

The window size is the same as that proposed by the authors of OSNet, for the simple reason that the authors provide proof of the efficiency of the CNN with this setting. The chosen size translates into an aspect ratio of 1:2.

Naturally, if a box has an original aspect ratio different to the one that OSNet is set to use, resizing it in preparation of going through the CNN will morph its features. And since the boxes in each frame can have different sizes, it follows that the features looked for here might morph in different ways every time. By setting the second parameter, each window can first be expanded to the desired aspect ratio, and then the scaling does not morph the features anymore. This usually helps the re-ID node do a better job, but also brings complications. Sometimes, the node might try to expand a window out of the bounds of the entire image, and even if that does not happen, this step still takes time and computational effort. Testing has shown that good results can be obtained even without this, and since the focus is on the speed of the pipeline, this auxiliary pre-processing step is omitted for the tests presented in the next subsection.

The maximum distance threshold overrules the Hungarian algorithm, when the latter associates galleries with feature vectors that are too different from them. This is set entirely based on observations from experiment, but following logical assumptions. A higher threshold allows more assignments to be made, which might decrease the number of *false negatives* obtained, but might increase the number of *false positives*. A lower threshold would have the complete opposite effects.

One issue that was not yet discussed is the nature of the descriptor galleries. The re-ID node has a number of parameters that dictate how these galleries are obtained:

1. Galleries can be loaded from pre-generated files.

2. Galleries can be generated at run-time from the first x detections, with x being a fixed number.

3. Galleries can be generated dynamically at run-time, with the oldest feature vector always being replaced by the new vector associated with the gallery. This keeps the number of feature vectors in each gallery fixed to x, while their content is continuously updating.

Testing has unsurprisingly shown that overall, the best results are obtained with pre-loaded galleries. But with a strict detection node and a small number of targets, the same performance is obtained even if the galleries are generated from the first few frames. In the synthetic testing environment used for this thesis , the galleries are pre-loaded, but it is exceptionally important to note that the pipeline up to this point can also offer useful results without having any prior knowledge of the environment.

Filter

The Kalman Filter node, has a difficult task, despite it being optional in solving the re-ID problem. It is implemented for the purpose of compensating for errors emerging from the first part of the pipeline, but there are many kinds of errors, and the filter cannot help with all of them at once. As discussed in subsection 3.3.4, this node uses:

1. **Covariance zoom factor** 0.03 - This multiplier is the q used for defining the process covariance matrix as $Q = \mathbf{I}_2 \cdot q$.

While different ways of tuning this parameter have long since been established [34, 35, 36], in this project, the parameter is set manually. This is based on several implementations of Kalman Filters on OpenCV, towards the purpose of mouse position smoothing. Higher values of the parameter tend to increase the aggressiveness of the correction phase. That means that the influence of each measurement obtained from the re-ID node is greater. In turn, that means erroneous measurements disturb the filter quickly, but it also means that correct measurements have strong effects.

4.2 Analysis

4.2.1 Time performance

As previously stated, the focus of this thesis is to provide a system that runs fast enough when deployed on hardware with limited capabilities. To that effect, the most important results to be observed are the processing times of frames that go through the pipeline.

While for this tests observing the experiment does not change the nature of the results as it does in quantum physics, it does influence them. Therefore, a note is made of the fact that when the nodes of the pipeline are not being monitored, they perform faster. In fact, monitoring just the filtering node instead of all three reduced the average processing time by as much as 100 milliseconds in some instances during testing.

	Detection		re-ID		Filter	
# of targets	μ_t	σ_t	μ_t	σ_t	μ_t	σ_t
1	205.5961	16.8854	292.9155	28.3244	304.0714	28.1093
2	226.3512	21.5463	418.1723	56.8567	430.6499	58.9476
	219.9926	20.3517	370.1779	54.2802	381.8178	55.1203
3	214.7096	25.5417	419.1290	66.8152	432.5906	66.0438
	231.8569	27.5716	450.1596	68.9869	461.7346	69.7035
	233.9296	31.0199	491.4654	55.3439	502.0477	55.6687
4	230.4993	33.2033	529.4627	92.2913	542.3467	93.4401
	225.1693	28.3422	567.9033	85.1343	581.8777	85.7157
5	237.4158	34.8240	559.0810	96.5153	570.2910	98.5397
	239.0184	31.7652	673.8273	98.3405	684.2232	98.7176
6	235.7465	35.2654	687.7012	149.6090	700.5336	148.5247
	224.2694	30.1956	739.9150	111.5804	752.2054	111.2723

Table 4.1: Time metric values on each level of the pipeline, in milliseconds

For solving the detection problem, only the first node of the pipeline is necessary. However all the tests are done with the entire pipeline running. Table 4.1 presents the specific values of the time metrics extracted from the detection node for tests with various number of targets in the frame.

As expected, the processing time required tends to increase with the number of targets considered. However, the increase is not dramatic. From the lowest average –obtained for one target– to the highest one –obtained for five targets– there is only a 20 milliseconds, or a 16.2563% increase.

The same effect can be seen when adding into the equation the re-ID node, but it is much more evident and aggressive in this case. The increases in both average processing time and deviation are more abrupt with the number of targets, suggesting

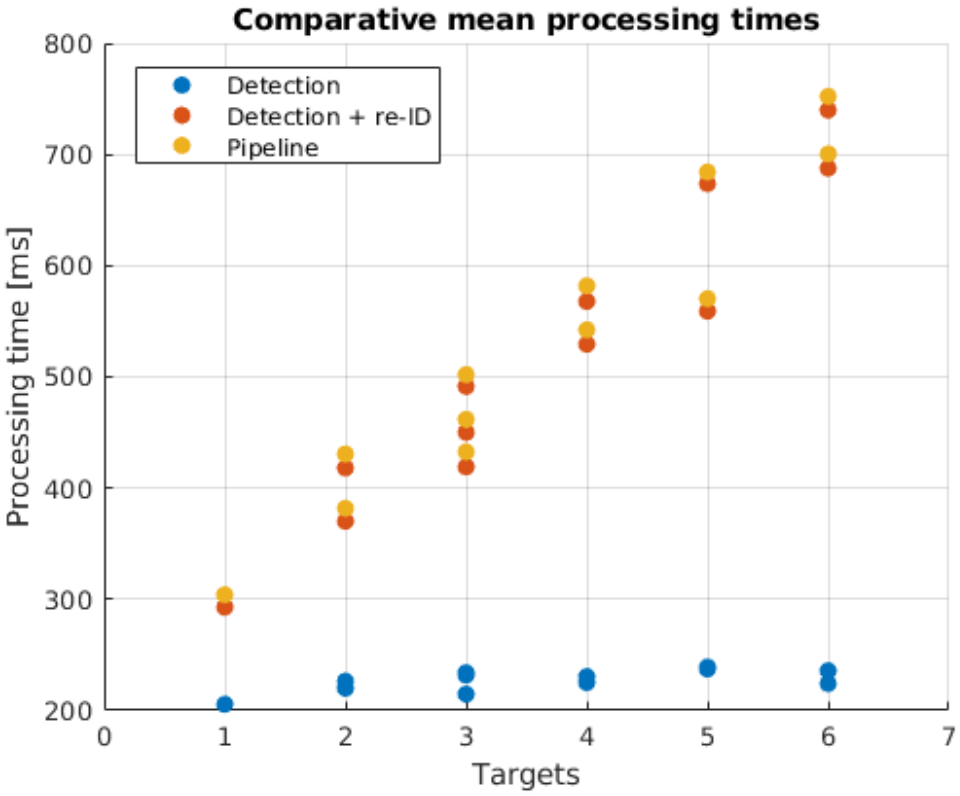


Figure 4.1: Average processing time on each level of the pipeline

that OSNet is the more computationally taxing of the two CNNs, or at least when used jointly with the Hungarian algorithm.

While on the detection node the time performance values are more scattered across tests done with the same number of targets, when adding the re-ID one, the time values tend to stay in the same range across tests. That is because in some of the tests, people are crossing in front of one another, and some targets are, at time, not visible. This means that the detection node has an easier job, as it does not generate as many boxes during such times, while the re-ID node still has to evaluate all possibilities.

(a) Detection **(b)** re-ID

Figure 4.2: Incomplete detection & re-ID due to targets not being visible

In Figure 4.2 the system tracks six targets, but two of them are not properly visi-

ble. Therefore, YOLO does not detect them, but the Hungarian algorithm in the re-ID node still has to check the available boxes against the galleries pertaining to the hidden targets.

Going forward, compounding these times with those of the filtering node does not add much to the total. In fact, the addition to the average processing time revolves around 12 milliseconds, regardless of the number of targets in the test. That confirms that that the filter is far less computationally taxing than the other two nodes. This is the main reason why it is included. While the node is not required to solve the problem of re-ID, it does help in specific situations and –as it is now proved– it does so without affecting the speed of the pipeline significantly.

A comparative overview of the average processing time on each level of the pipeline is provided in Figure 4.1 to reinforce the observations made up until this point.

Indeed, taking in account all of the discussed results, it becomes apparent that the re-ID node is responsible for most of the delays in processing. Each new target adds a latency of at least 50 milliseconds, which can quickly degrade the overall performance of the system.

Deciding on whether the system is fast enough depends on the specifics of the application for it to be integrated in. However, the conclusion remains, that considering the complexity of the problem and the limitations of the hardware used, the system performs admirably, and good enough for the purposes of the application developed by the RoPeRT research group.

4.2.2 Quality of results

While the focus of the thesis is on discussing the time performance of the system, it is clear that the quality of the results is also important. After all, the proposed system could run better than real-time, but return useless results. This subsection aims to prove that that is not the case. For assessing the quality of the results, the metrics proposed in subsection 4.1.1 are used, but it is still required to decide on the structure of the tests on which these metrics are to be applied.

There are essentially two types of tests included in this analysis:

1. **Lane tests**, in which each target stays or walks in their own lane, without crossing in front or behind another target, as seen in Figure 4.3a.

2. **Wild tests**, in which the targets are free to move around the scene as they please, as seen in Figure 4.3b.

The two types are chosen to extract different conclusions. *Lane tests* should be easier and should reflect how well the re-ID problem is solved, while *wild tests* aim to check the adaptability of the system.

(a) Lane test **(b)** Wild test

Figure 4.3: Frames extracted from the two types of tests

Detection

For the detection node, the only metric used is the detection rate, automatically computed, which represents a measure of the percentage of expected targets that are properly sensed over time. Figure 4.4 shows how the detection node performs in each of the studied test cases.

In analysing the figure, one outlier value becomes evident. The one-target test is considered *wild* because of the unusual poses that the target takes. Other than that, it can equally be thought of as a *lane* test, where the target has the entire frame as their individual lane. Then it does not appear as an outlier anymore. Indeed in the *wild* tests it is at times impossible for the detection node to find all of the targets, as exemplified before in Figure 4.2, therefore the detection rates on these tests are lower. Despite that, they do not fall below 85%, and this is thanks to the permissive tuning of the detection node. With more restrictive parameters, the node continues to perform well on a reduced number of targets, but for tests with five or six targets, the rate falls to almost below 10%.

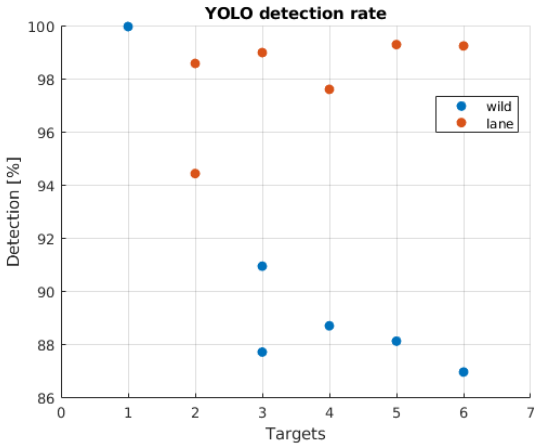
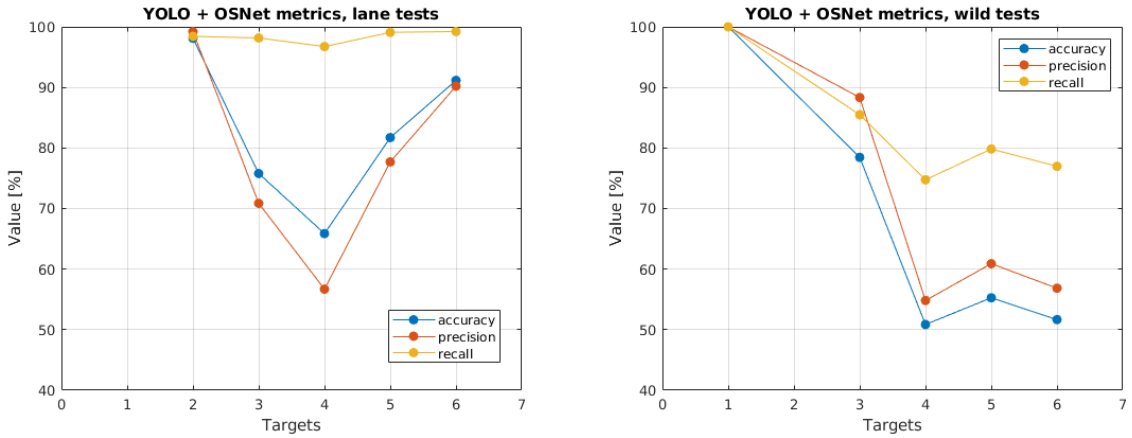


Figure 4.4: Detection rates for both *lane* and *wild* tests

re-ID

On this level, the *accuracy*, *precision* and *recall* metrics become relevant. Figure 4.5 presents their evolution over the used test cases.

Of the three metrics in discussion, the *recall* yields the highest values. That is mostly thanks to the high detection rates of the previous node. However, having more boxes to identify than targets encourages false positives to appear, as reflected by the



(a) *Lane* tests **(b)** *Wild* tests

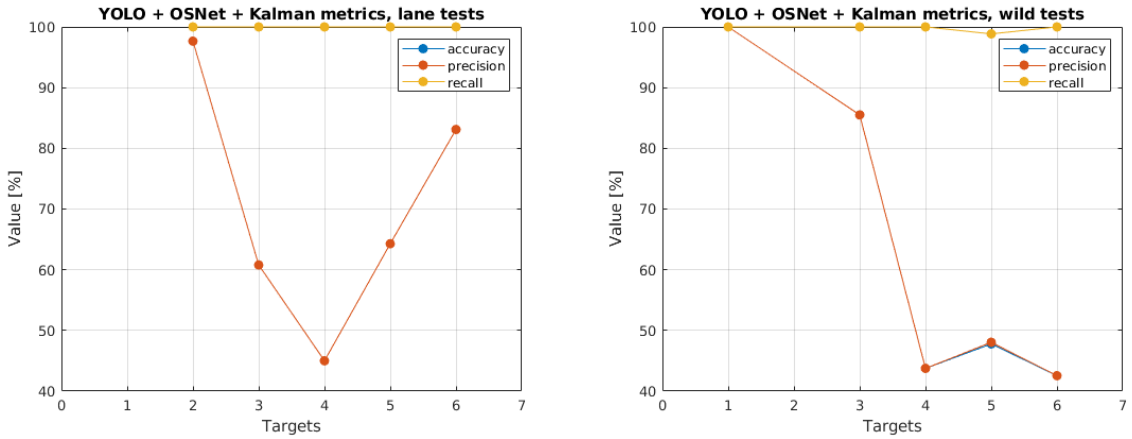
Figure 4.5: Performance indices of the detection + re-ID ensemble on the two types of tests

dip in *precision* scores. Perhaps a different set of parameters would improve the values on tests with two to four targets, but as the number of targets increases, the system becomes more unstable. The chosen parameters offer a good compromise between the quality of results on smaller numbers of targets and bigger ones alike.

Of course the pipeline so far performs better in *lane* tests than in *wild* ones, but what is important here is that in either scenario, *accuracy* and *precision* do not fall below 50%, which is indeed highly remarkable.

Filter

Going further to the end of the pipeline, the results on the filtering node are aggregated and presented through Figure 4.6.



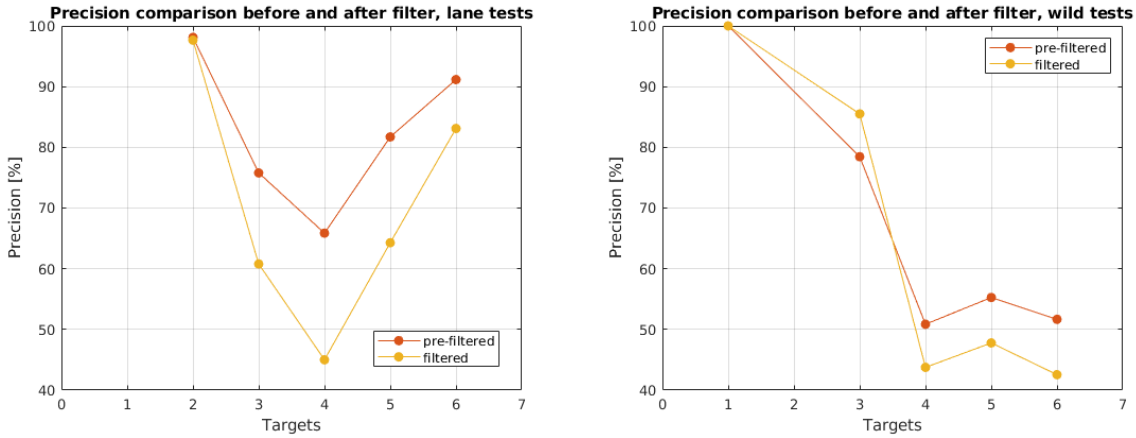
(a) *Lane* tests **(b)** *Wild* tests

Figure 4.6: Performance indices of the pipeline on the two types of tests

The most immediate observation to be made here is that the recall score when using the filtering node is almost, if not really, 100%. This means that the mechanism is indeed effective at eliminating *false negatives*.

At the same time, the *accuracy* and the *precision* seem to have identical or similar values to one another across tests. Taking into account their formulae, Equation 4.3 and Equation 4.4, the conclusion follows that there are no more *true negatives* either. That is because the unidentified boxes forwarded by the re-ID node effectively get discarded and do not influence any of the filters. However, this is only true because, in this testing environment, the system knows the number of targets it has to track. In situations where this information is not provided, the results might differ.

Again, the pipeline performs better on *lane* tests than on *wild* ones. But while the results remain relevant in both scenarios, the scores tend to drop lower than before, and Figure 4.7 expresses that more clearly.



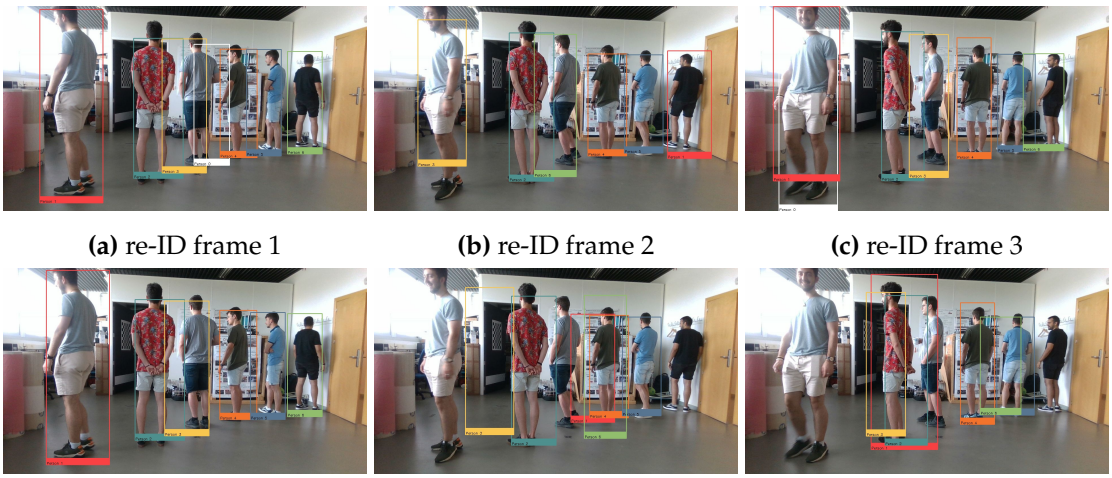
(a) *Lane* tests

(b) *Wild* tests

Figure 4.7: Comparison of precision with and without using the filtering node

Indeed, in most cases the precision drops when adding the filtering node. That is because the re-ID node can produce *false positives* in one frame and correct them in the next, but the effect of the false measurements influence the Kalman filters over time. So, effectively, the filtering node has an integrative effect on *false positives*, as seen in Figure 4.8. In the second frame, the re-ID node assigns the boxes wrongly, but then goes back to normal in the third frame. Meanwhile, on the filter node, the information provided by the re-ID node alters the results so much that even if the third set of data is good, the results do not go back to normal.

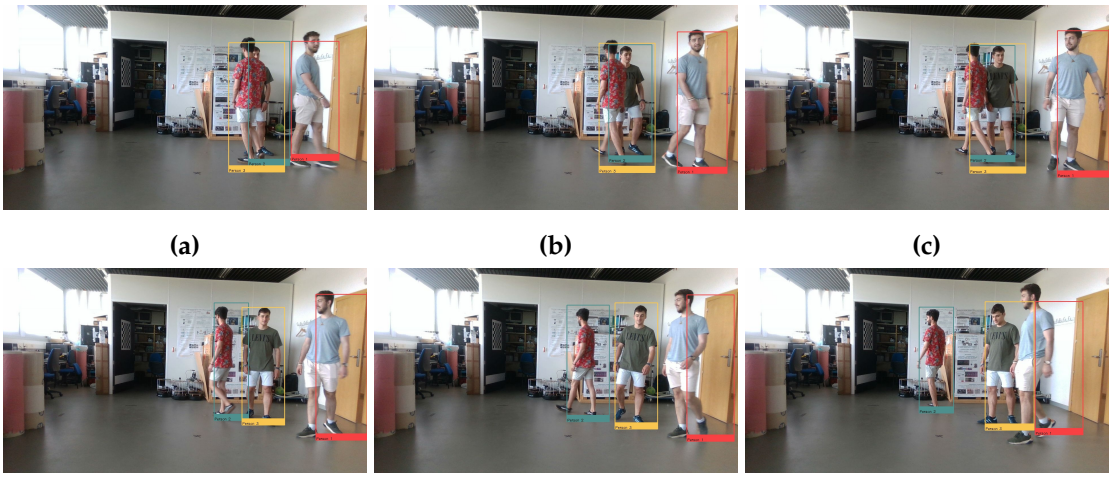
Additionally, sometimes the filter node creates *false positives* by themselves. When a target changes their direction of movement abruptly, the filter needs a couple of measurements to recover, as seen in Figure 4.9. And if a target changes their direction while being hidden behind other targets, then it becomes impossible for the filter to notice it. Despite all of this, the system performs quite well even when these kinds of situations appear, as seen in the previously analysed graphs.



(a) re-ID frame 1 **(b)** re-ID frame 2 **(c)** re-ID frame 3

(d) filter frame 1 **(e)** filter frame 2 **(f)** filter frame 3

Figure 4.8: Comparison of precision with and without using the filtering node



(a) **(b)** **(c)**

(d) **(e)** **(f)**

Figure 4.9: Results on filter node when target changes the direction of movement

Overall

Finally, the behaviour of the pipeline and its components can be summed up. In short, each node has the following effect:

- The **detection node** introduces *false negatives* if it is set too restrictive, and increases the risk of obtaining *false positives* when it is more permissive.
- The **re-ID node** introduces *false positives*.
- The **filtering node** eliminates *false negatives*, but amplifies the *false positives* that propagate from the re-ID node.

With the parameters tuned as described in the previous section, a satisfactory compromise is obtained, and the pipeline provides useful results.

Chapter 5

Conclusions

The objective of this master's thesis was to detail on the design, the implementation and, most importantly, the performance of a lightweight software pipeline for person re-identification tailored for use on systems with limited resources. So, unlike other works done in the field, this one focused on the lightweight attribute of the architecture, aiming to obtain the best possible processing speeds. And with processing times on the experimental prototype being kept consistently under one second for varying numbers of targets, the system provides a viable solution for use in real-time applications.

The proposed system is designed on a modern version of ROS and is using a combination of classic methods and tools alongside modern, state-of-the-art ones to solve its tasks. Two specialised CNNs, YOLO and OSNet, enable the pipeline to detect and describe targets, while the classical Hungarian algorithm and Kalman filter help identify the targets and refine the detection results. For the experimental prototype, an Intel NUC computer, lacking a GPU and featuring a modest CPU provided the computationally restricted environment needed to properly test the viability of the design.

The work done towards this project reveals that solving a complex problem such as the one at hand can be done using a lightweight architecture without sacrificing the quality of the results. On the custom data set which the prototype was tested against, the proposed pipeline showed promising results, with the quality indices ranging between $50\% - 100\%$ for unfiltered results, and $40\% - 100\%$ for filtered ones.

Another relevant information the project has offered is about the complexities of the tasks that the problem of re-identification imposes. Object detection alone can be optimised to introduce around as little as 200 milliseconds of latency on the prototype hardware. Meanwhile, assigning identities to detected objects seems to add latency between 50 and 100 milliseconds for each object requiring identification.

Limitations and Future work

The proposed architecture is not presented as perfect. It is instead an important first step towards implementing state-of-the-art concepts in environments that traditionally do not have the luxury of being able to handle them. Naturally, there is room for improvement.

The quickest betterment to be made could be to fine-tune the parameters. Or, more specifically, to employ auto-tuning techniques, making the system easier to use and more versatile. Then, the version of YOLO that is used here is the smallest one available and still performs admirably. However, considering the small influence that it has on the speed of the system at this point, it would be worth to try replacing it with slightly larger, but newer versions of the network. The re-identification node does its job well too, but it could potentially do an even better job by considering the previous positioning of people in the frame besides the feature descriptors given by OSNet. Finally, the filter node has limited capabilities for now. It helps in some scenarios, but it introduces errors in others. More advanced techniques might address this.

All of these are punctual aspects which could be improved upon in the future. But at the same time, there is the macroscopic problem of integration. As far as this thesis goes, the experimental setup is standalone and not working as part of a more interactive system. The next step would be to integrate the proposed pipeline in the one already developed within the RoPeRT research group. This would result in a distributed multi-robot system truly capable of multi-target tracking and re-identification.

References

[1] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)*, pages 1–6, 2017.

[2] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[3] Greg Welch, Gary Bishop, et al. An introduction to the kalman filter. 1995.

[4] Jonathan G Richens, Ciarán M Lee, and Saurabh Johri. Improving the accuracy of medical diagnosis with causal machine learning. *Nature communications*, 11(1):1–9, 2020.

[5] Amelia Fiske, Peter Henningsen, Alena Buyx, et al. Your robot therapist will see you now: ethical implications of embodied artificial intelligence in psychiatry, psychology, and psychotherapy. *Journal of medical Internet research*, 21(5):e13216, 2019.

[6] Jay Lee, Hossein Davari, Jaskaran Singh, and Vibhor Pandhare. Industrial artificial intelligence for industry 4.0-based manufacturing systems. *Manufacturing letters*, 18:20–23, 2018.

[7] Aaron Daniel Cohen, Adam Roberts, Alejandra Molina, Alena Butryna, Alicia Jin, Apoorv Kulshreshtha, Ben Hutchinson, Ben Zevenbergen, Blaise Hilary Aguera-Arcas, Chung ching Chang, Claire Cui, Cosmo Du, Daniel De Freitas Adiwardana, Dehao Chen, Dmitry (Dima) Lepikhin, Ed H. Chi, Erin Hoffman-John, Heng-Tze Cheng, Hongrae Lee, Igor Krivokon, James Qin, Jamie Hall, Joe Fenton, Johnny Soraker, Kathy Meier-Hellstern, Kristen Olson, Lora Mois Aroyo, Maarten Paul Bosma, Marc Joseph Pickett, Marcelo Amorim Menegali, Marian Croak, Mark Díaz, Matthew Lamm, Maxim Krikun, Meredith Ringel Morris, Noam Shazeer, Quoc V. Le, Rachel Bernstein, Ravi Rajakumar, Ray Kurzweil, Romal Thoppilan, Steven Zheng, Taylor Bos, Toju Duke, Tulsee Doshi, Vinodkumar Prabhakaran, Will Rusch, YaGuang Li, Yanping Huang, Yanqi Zhou, Yuanzhong

Xu, and Zhifeng Chen. Lamda: Language models for dialog applications. In *arXiv*. Google, 2022.

[8] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.

[9] Ahmed Elgammal, Bingchen Liu, Mohamed Elhoseiny, and Marian Mazzone. Can: Creative adversarial networks, generating" art" by learning about styles and deviating from style norms. *arXiv preprint arXiv:1706.07068*, 2017.

[10] Hung-Min Hsu, Tsung-Wei Huang, Gaoang Wang, Jiarui Cai, Zhichao Lei, and Jenq-Neng Hwang. Multi-camera tracking of vehicles based on deep features re-id and trajectory-based camera link models. In *CVPR workshops*, pages 416–424, 2019.

[11] Hongpeng Wang, Jinchao Zhu, Wan Dai, and Jingtai Liu. A re-id and tracking-by-detection framework for multiple wildlife tracking with artiodactyla characteristics in ecological surveillance. In *2019 IEEE International Conference on Real-time Computing and Robotics (RCAR)*, pages 901–906. IEEE, 2019.

[12] Mengran Gou, Srikrishna Karanam, Wenqian Liu, Octavia Camps, and Richard J Radke. Dukemtmc4reid: A large-scale multi-camera person re-identification dataset. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 10–19, 2017.

[13] Ergys Ristani and Carlo Tomasi. Features for multi-target multi-camera tracking and re-identification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6036–6046, 2018.

[14] Murtaza Taj and Andrea Cavallaro. Distributed and decentralized multicamera tracking. *IEEE Signal Processing Magazine*, 28(3):46–58, 2011.

[15] Thi Thanh Thuy Pham, Thi-Lan Le, Hai Vu, Trung Kien Dao, et al. Fully-automated person re-identification in multi-camera surveillance system with a robust kernel descriptor and effective shadow removal method. *Image and Vision Computing*, 59:44–62, 2017.

[16] Sara Casao, Abel Naya, Ana C. Murillo, and Eduardo Montijano. Distributed multi-target tracking in camera networks. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1903–1909, 2021.

[17] Iñigo Etayo Gil. Asignación distribuida de tareas dinámicas en sistemas multi-robot. Master's thesis, Universidad de Zaragoza, Zaragoza, Aragón, España, 2020.

[18] Eduardo Montijano, Danilo Tardioli, and Alejandro R. Mosteo. Distributed dynamic sensor assignment of multiple mobile targets. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4921–4926, 2019.

[19] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*, 2015.

[20] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[21] Kaiyang Zhou, Yongxin Yang, Andrea Cavallaro, and Tao Xiang. Omni-scale feature learning for person re-identification. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 3701–3711. IEEE, 2019.

[22] Joseph Redmon and Ali Farhadi. Yolo9000: Better, faster, stronger, 2016.

[23] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[24] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[25] Kaiyang Zhou, Yongxin Yang, Andrea Cavallaro, and Tao Xiang. Learning generalisable omni-scale representations for person re-identification, 2019.

[26] H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2):83–97, 1955.

[27] F. Roberts and B. Tesman. *Applied Combinatorics*. CRC Press, 2009.

[28] David F. Crouse. On implementing 2d rectangular assignment algorithms. *IEEE Transactions on Aerospace and Electronic Systems*, 52(4):1679–1696, 2016.

[29] H. W. Kuhn. Variants of the hungarian method for assignment problems. *Naval Research Logistics Quarterly*, 3(4):253–258, 1956.

[30] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, Andrew Y Ng, et al. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5, Kobe, Japan, 2009.

[31] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[32] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[33] Kaiyang Zhou and Tao Xiang. Torchreid: A library for deep learning person re-identification in pytorch. *arXiv preprint arXiv:1910.10093*, 2019.

[34] Xile Gao, Haiyong Luo, Bokun Ning, Fang Zhao, Linfeng Bao, Yilin Gong, Yimin Xiao, and Jinguang Jiang. Rl-akf: An adaptive kalman filter navigation algorithm based on reinforcement learning for ground vehicles. *Remote Sensing*, 12:1704, 05 2020.

[35] P. Fung and M. Grimble. Dynamic ship positioning using a self-tuning kalman filter. *IEEE Transactions on Automatic Control*, 28(3):339–350, 1983.

[36] O.V. Korniyenko, M.S. Sharawi, and D.N. Aloi. Neural network based approach for tuning kalman filter. In *2005 IEEE International Conference on Electro Information Technology*, pages 1–5, 2005.