Multi-Camera Multi-Person Tracking Using Reinforcement Learning

Axel Kärrholm, Linus Rickman

Master's thesis
2022:E32

Lund University

Faculty of Engineering
Centre for Mathematical Sciences
Mathematics

CENTRUM SCIENTIARUM MATHEMATICARUM

Abstract

The problem of multi-object-tracking in a network of cameras is an interesting and non-trivial problem. Given videos from a number of cameras the goal of *Multi-Camera Multi-Object Tracking* (MCMOT) is to find the full visible trajectory of each pedestrian from the videos as the pedestrians move across cameras. Compared to monocular tracking the main difficulties lie in associating pedestrian detections to tracks in locations where the camera views overlap. We develop two MCMOT methods; a traditional threshold-based *Simple Online and Realtime Tracking* (SORT) algorithm and a reinforcement learning track management method. The methods are evaluated on Blender 3D simulated data sets and on real-world recordings from calibrated cameras. The reinforcement learning model outperforms the more traditional method in *Multi-Object Tracking Accuracy* (MOTA) up to 4% on the simulated and real data sets. Finally, our experiments show that the reinforcement learning method can transfer knowledge from simulated to real data during inference.

Keywords: Multi-Camera Multi-Object Tracking, Reinforcement Learning, Computer Vision, Machine Learning, Object Detection

Acknowledgements

We would like to thank Anders Robertsson for supervising us during the thesis project. We would also like to direct a special thanks to our supervisor Maria Priisalu for the tips on training the reinforcement learning agents and for her insightful comments on our report.

Division of Work

We state that both authors have contributed equally to this thesis.

List of Abbreviations

CNN, Convolutional Neural Network

DLT, Direct Linear Transformation

HOG, Histogram of Oriented Gradients

IOU, Intersection over Union

MARL, Multi-Agent Reinforcement Learning

MCMOT , Multi-Camera Multi-Object Tracking

MOTA, Multi-Object Tracking Accuracy

MOTP, Multi-Object Tracking Precision

MTMCT, Multi-Target Multi-Camera Tracking

NMS, Non-Maximum Suppression

PPO, Proximal Policy Optimisation [37]

R-CNN, Region-Based Convolutional Neural Network [14]

RL, Reinforcement Learning

SIFT, Scale-Invariant Feature Transform

SORT, Simple Online and Realtime Tracking [2]

SVD, Singular Value Decomposition

TRPO, Trust-Region Policy Optimisation [36]

YOLO, You Only Look Once (image detection network) [32]

Contents

1 Introduction

The *multi-camera multi-object tracking* (MCMOT) problem arises in the context of for example surveillance systems, autonomous driving, urban planning and sports. In surveillance systems, one might want to automatically follow pedestrians' movements across a network of cameras without the need of having a human manually monitoring the cameras. In sports, one could want to analyse players' movements across the field during a game. When it comes to urban planning, the flow of pedestrians in public areas can be surveyed in order to better plan the layout of walkways, street lights etc. In the autonomous car industry there is a need to accurately track and forecast pedestrians in order to avoid accidents. All of these tasks can be formulated as multi-camera multi-object tracking problems.

The MCMOT problem is closely related to the mono-camera (monocular) tracking problem and many of the methods used in the multi-camera setting are based on mono-camera tracking systems [21]. The need for multiple cameras arises when one wants to cover and track a larger area than possible with a single camera. Moreover, by using multiple cameras one can more accurately determine the object's position in 3D space by triangulating the detections from different cameras. There is no unique definition of the MCMOT problem but it is often formulated as an estimation problem, where the positions of a number of objects (pedestrians) are to be found. The goal is to estimate the set of trajectories (tracks) $\{\mathbf{x}_t^1 \dots \mathbf{x}_t^i \dots \mathbf{x}_t^{M_t}\}_t$ for all M_t objects in the scene for each time frame t. A track or trajectory $\{\mathbf{x}_t^i\}_{t=t_i}^{N_i}$ is a sequence of states from the time frame t_i it enters the scene to the time N_i that it leaves the scene with a unique id i. The state $\mathbf{x}_t^i$ could differ widely depending on the approach taken. A few common examples are to track the centre position of each object, or some area that the object occupies. The estimation of the trajectories is done using information from multiple cameras $c_k \in \mathcal{C}$, that each give rise to a video sequence $\{\mathbf{f}_1^k \dots \mathbf{f}_t^k \dots \mathbf{f}_N^k\}$ where $\mathbf{f}_t^k$ is the frame (image) at time t from the video of camera c_k. Furthermore, N is the number of frames in the video sequence from each camera c_k (note that we assume that all cameras produce the same number of frames and that they are synchronised in time).

The objective is to find trajectories that as closely as possible estimate the ground truth positions while maintaining consistent correspondence to the correct identity in the real world. In reality there will be a discrepancy between the track id numbers, positions and the number of tracks for the estimated objects and the real objects in the scene and in order to get a working system, we need to solve numerous difficult sub-tasks. These sub-tasks include object detection in video, the association of detections to tracks, transformation from coordinates in image space to coordinates in the 3D world, the fusion of associated detections and the estimation of the objects' states. Some of the difficulties for the object detector to overcome are different lighting conditions, sizes, orientations and occlusions of objects. When detections are associated to tracks, false detections, missed detections and objects situated closely together can cause problems. In the multi-camera setting we also have to associate detections in different cameras to a common object in the 3D scene, adding further complexity compared to the single camera system.

In this thesis we focus on real-time algorithms that can run approximately at a frame rate of > 25 FPS. Specifically, our focus lies on pedestrian tracking in a network of stationary, partially overlapping cameras. The tracking algorithms we analyse do not depend on visual data and are therefore not limited to pedestrians. Like many of the proposed algorithms to solve the MCMOT problem, our method will belong to the tracking-by-detection paradigm [2, 26, 4]. Methods within this paradigm first detect objects in video data and then associate the detections to existing trajectories of objects. These associated detections are then used to update said trajectories. A good way to visualise this paradigm is to consider the following figure:

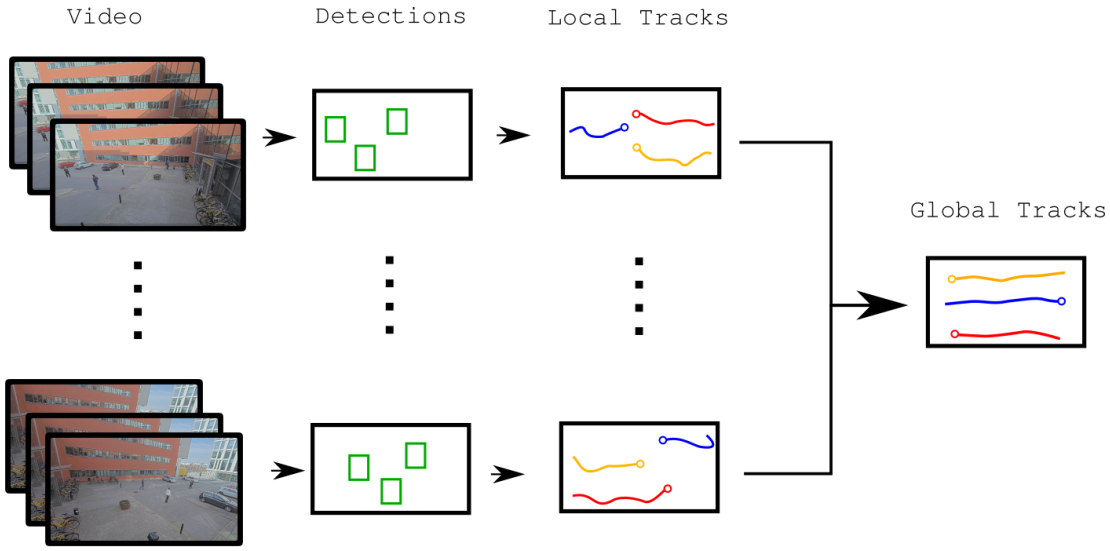


Figure 1: Overview of a tracking-by-detection pipeline. Detections are made in frames from video. The detections are used to create local tracks. The local tracks are projected into a global coordinate system where they build the global tracks.

In Figure 1 we see how we first perform detections in video streams, to then associate these detections to tracks and then finally fuse these local tracks in image space into global tracks in the 3D world. We first implement a baseline model, namely the *Simple Online and Realtime Tracking* (SORT) algorithm [2], proposed by Bewley et al. for intra-camera tracking. This method was chosen for its real-time performance and competitive accuracy and precision. With this method, the detections at each time frame are associated to the active tracks by using the Hungarian algorithm [25]. The SORT algorithm uses a Kalman filter to update the estimates of the tracks. We also extended the SORT algorithm to a multi-camera setting.

Furthermore, we implement a reinforcement learning algorithm for the tracking inspired by the article [34] by Rosello and Kochenderfer. In this article each track is modeled as a reinforcement learning agent [34], thus treating the MCMOT problem as a *Multi-Agent Reinforcement Learning* (MARL) problem. This reinforcement learning method is intended to allow for a more dynamic decision process in the tracking as opposed to the SORT algorithm where hard thresholds are used. These thresholds are used for example to determine when to use a detection, discard it or

terminate a track. The RL agents instead learn from training data when to initiate tracks, terminate tracks or use the detections to update the tracks. While Rosello and Kochenderfer implemented a MARL method for the mono-camera setting, we extend this method and implement a *multi-camera* reinforcement learning model for the MCMOT problem. The details for all algorithms can be found in the theory and methodology sections 3 and 4. Lastly, to evaluate the performance of the algorithms in this thesis we apply the widely used metrics *Multi-Object-Tracking Accuarcy* (MOTA) and the *Multi-Object-Tracking Precision* (MOTP) [1].

1.1 Problem Formulation

This thesis aims to explore methods for solving the multi-object multi-camera tracking problem. We have briefly touched upon the definition of this problem in the introduction above, but will now give the formal definition that we use. In the MCMOT problem we have a network with K cameras $\mathcal{C}$ that are all synchronised in time with the same frame rate. These cameras produce video footage that cover the scene that we are interested in. Each camera c_k in the camera network $\mathcal{C}$ produces a video sequence $\{\mathbf{f}_1^k \dots \mathbf{f}_t^k \dots \mathbf{f}_N^k\}$ of length N. The goal is to track, i.e, estimate the states $\{\mathbf{o}_t^j\}_{t=t_j}^{N_j}$ of all M objects that are present in the scene at any time. The object j in the scene produces a sequence of states from the first frame t_j to the last frame N_j for which it is visible in any of the cameras. The sequence of states $\{\mathbf{o}_t^j\}_{t=t_j}^{N_j}$ together with a unique id number is what we will refer to as a track or a trajectory of an object. The length of the trajectory is the number of frames for which the object is present in the scene, in any camera.

The goal of any MCMOT algorithm is to produce tracks $\{\mathbf{x}_t^i\}_{t=t_i}^{N_i}$ that correspond to the true tracks of the objects in the scene. The algorithm will at the end of the tracking output M_e number of tracks. If the tracking is done perfectly, we will have found a one-to-one correspondence between the estimated tracks $\{\mathbf{x}_t^i\}_{t=t_i}^{N_i}$ and the true object tracks $\{\mathbf{o}_t^j\}_{t=t_j}^{N_j}$ in the scene. This means that $M = M_e$ and that for all object tracks j, there exists a corresponding estimated object track i of equal length and same starting frame with each $\mathbf{x}_t^i = \mathbf{o}_t^j$ for all frames t. In general this will not be the case and we will have differences both in the number, length and states of the tracks meaning we can not find this one-to-one correspondence.

Any algorithm that tries to solve the MCMOT problem and outputs a number of tracks needs to have a way of deciding when to start a new track, when to update a track and when to terminate a track. Starting a new track means that the MCMOT algorithm decides that an object has entered the scene. Updating a track means that the length of the track is increased by one and that a new state estimate for the next frame is found. Terminating a track $\{\mathbf{x}_t^i\}_{t=t_i}^{N_i}$ means that its length is fixed and can no longer be updated.

There are many ways one could go about improving a pipeline that solves the MCMOT problem. We propose a novel method to perform multi-camera multi-object tracking using reinforcement learning. With this method, some tracking decisions

are made by a neural network trained on simulated data using reinforcement learning. These decisions include starting, updating and terminating tracks. It also includes decisions as to which detections it should trust and when to use them. This RL-method will be compared to the more traditional SORT algorithm. The aim of this thesis is to answer the question: how well suited is the reinforcement learning approach for track management in a multi-camera setting, compared to the more traditional SORT method, evaluated using the multi-object-tracking accuracy and precision metrics?

2 Background

Algorithms proposed to solve the MCMOT problem generally consist of several parts and can take widely different approaches to the problem. In this section we will introduce some of the most common approaches to multi-camera tracking and how they have evolved with time. This includes introducing how the detections are obtained, how the data association is made and how the trajectories are updated. Some of the first detection algorithms in tracking systems were based on background subtraction. With this method, a reference background model is used [43]. The difference between the background model and the current frame is used to determine where in the image pedestrians are located. This type of detection is sensitive to any changes in the scene from the reference image and if the background changes, without the background model being updated, the detections will be very noisy. Changes in pixel intensities are also not always the result of a person being present. Another method for object detection that have been commonly used is the *Histogram of Oriented Gradients* (HOG) method [9].

The background model and HOG approach to pedestrian detections have been widely used but are no longer considered state-of-the-art. Today, most image detection algorithms are deep learning-based. Two of the most commonly used methods are YOLO [32] and R-CNN [14]. The field is however being researched extensively and many new approaches become available every year [28, 47, 16]. In the tracking-by-detection paradigm one usually tries to associate the detections to already existing tracks, or create tracks from leftover detections. There exist many different approaches to the data association problem. A common method is to use the Hungarian algorithm to associate new detections to existing tracks with a predefined distance function [2]. Others try to learn this distance function with a neural network [44] and then apply the Hungarian algorithm. Clustering methods are also a common choice such as DBSCAN [45].

Apart from different detection and data association algorithms, an important distinction between MCMOT algorithms is that of online and offline methods. An offline method is a method that can not be deployed in real-time, either because it is too slow or because it uses information from future detections [46, 40, 5]. In an offline setting, large parts or even the whole video sequence is known beforehand and the tracks can be created using global information over all time frames. In this setting it is common to formulate the multi-object tracking problem as a network-flow problem [49]. With this approach, a graph is created where the detections in each frame are represented as nodes in a graph. The nodes from adjacent frames in this graph are connected by weighted edges, where the weights are the probability for the detections to originate from the same object. The tracking problem can then be solved by optimising the network flow with a min-cost flow algorithm. In certain applications, such as urban planning or sport analysis, the offline approach might be acceptable. However, in this thesis we focus on online, real-time algorithms that are more suitable, for example, in autonomous driving and camera surveillance systems. If a tracking algorithm is fast enough to be run in real-time ($\geq$ 25 FPS), we refer to it as an *real-time method*, otherwise we refer to it as an *offline method*.

Tracking algorithms can also be split into two further categories namely *visual* and *non-visual* methods. Visual methods make use of the appearance of a person in a detection, while non-visual methods only take the detections' spatial properties into account. An example of a visual tracking system is the so called deep-SORT algorithm that uses of the appearance of a pedestrian to associate detections to tracks [44]. A related problem is the so called *Re-Identification* (Re-Id) problem. Here, one wants to associate the id:s of persons reappearing in the scene after having been lost in the tracking. This is especially relevant in non-overlapping camera settings. This is usually done by first extracting features from detected pedestrians using convolutional neural networks and then comparing them using for example another network [16].

In tracking algorithms it is common to use some motion model to better track the concerned objects. A common approach is the Kalman filter which has been used extensively [2]. Other filtering approaches are the particle filter [17] and the joint probabilistic data association filter [33]. Lastly, due to the cumbersome process of annotating large amounts of video data, some work has been done on simulated data sets for multi-camera multi-object tracking. The JTA data set [10], for example, is extracted from the video game Grand Theft Auto for pose estimation and multi-camera tracking. Other publically available data sets used for benchmarking include the WILDTRACK data set [7], the PETS 2009 data set [11] and the EPFL data sets [13].

3 Theory

In this section, the mathematical models and theory used in this thesis are explained. We start by going over the camera model as it is commonly used in the field of computer vision and its calibration. Then, the neural network used for human detection is described. After the detector model has been explained, we go over the Kalman filter used for tracking followed by the data association algorithm. Next, the basic concepts of reinforcement learning and the *Proximal Policy Optimisation* algorithm are described. Finally, we will describe the evaluation metrics used to evaluate the tracking results.

3.1 Cameras

When working with cameras and projective geometry homogeneous coordinates are often used. This coordinate system simplifies a lot of the calculations and is therefore introduced before moving on to the camera model.

3.1.1 Homogeneous Coordinates

A point $\mathbf{p} = (x, y)^T \in \mathbb{R}^2$ in the Cartesian coordinate system will in homogeneous coordinates be represented by the set of vectors

$$\tilde{\mathbf{p}} = \lambda(x, y, 1)^T \in \mathbb{R}^3, \tag{1}$$

where $\lambda \in \mathbb{R} \setminus \{0\}$ [18]. We use tilde to indicate that a vector is written using homogeneous coordinates. Thus, the homogeneous coordinates $(5, 2, 1)^T$ and $(10, 4, 2)^T$ both represent the same point, $(5, 2)^T$ in Cartesian coordinates. Adding an extra element to represent points will allow these points to be translated and rotated with a single matrix multiplication, something that would not be possible in Cartesian coordinates. A translation and rotation of a point $\tilde{\mathbf{p}}$ in homogeneous coordinates can be expressed with a single matrix multiplication by concatenating a 3×3 rotation matrix $\mathbf{R}$ with a 3×1 translation vector $\mathbf{t} \in \mathbb{R}^3$ and multiplying them together like

$$\begin{pmatrix} \mathbf{R} & \mathbf{t} \end{pmatrix} \tilde{\mathbf{p}} = \begin{pmatrix} \mathbf{R} & \mathbf{t} \end{pmatrix} \begin{pmatrix} \mathbf{p} \\ 1 \end{pmatrix} = \mathbf{R}\mathbf{p} + \mathbf{t}. \tag{2}$$

In general, if we have a vector $\mathbf{p} \in \mathbb{R}^n$ in Cartesian coordinates, its homogeneous representation is $\tilde{\mathbf{p}} = \lambda(\mathbf{p}, 1)^T$ where $\lambda \in \mathbb{R} \setminus \{0\}$ as before. With this short introduction of homogeneous coordinates we move on to introduce the pinhole camera model.

3.1.2 The Pinhole Camera Model

Images are 2D-representations of the 3D-world. The pinhole camera model can be used to model the projections of points in the camera's local 3D coordinate system $\mathbf{x}_c = (x, y, z) \in \mathbb{R}^3$ onto points $\mathbf{x}_{im} = (x, y) \in \mathbb{R}^2$ in an image plane. The projected points we see in the image are located in this image plane. The easiest way to visualise the pinhole camera model is to consider the following image.

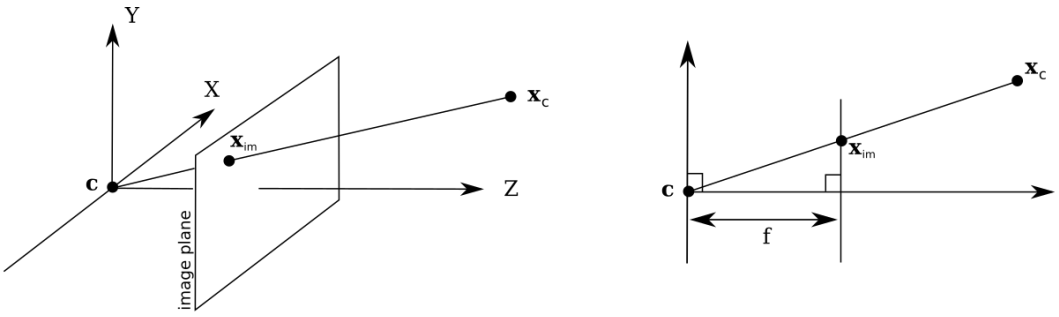


Figure 2: The pinhole camera model

From Figure 2 above we see how a point $\mathbf{x}_c$ in the camera's coordinate system projects onto the image plane at a point $\mathbf{x}_{im}$ along the line connecting $\mathbf{x}_c$ with the origin. The projection $\mathbf{x}_{im}$ is obtained by dividing the scene point $\mathbf{x_c}$ with the z coordinate and multiplying with f

$$(x, y, z) \rightarrow \left(\frac{fx}{z}, \frac{fy}{z}, f \right),$$

where f is called the *focal length* of the camera and defines the distance between the camera centre $\mathbf{c} \in \mathbb{R}^3$ and the image plane. Since we project the point along a line towards the origin and the z-coordinate becomes f, this gives us the point $\mathbf{x}_{im}$ in the image plane in the camera's coordinate system.

When dealing with the pinhole camera model we need to distinguish between two different coordinate systems. The first system is the camera's local coordinate system. This coordinate system has the camera at the origin with its principal axis along the z-axis as seen in Figure 2. The second coordinate system that we consider is the global world coordinates which is required as the camera can be rotated and translated anywhere in the world. Hence, to get the projection $\mathbf{x}_{im}$, a point $\mathbf{x}_w \in \mathbb{R}^3$ in the global world coordinates first needs to be transformed into the cameras local coordinate system. This can be done by applying a 3×3 rotation matrix $\mathbf{R}$ and a translation vector $\mathbf{t} \in \mathbb{R}^3$ to the point $\mathbf{x}_w$ in the world coordinate system and we get:

$$\mathbf{x}_c = \mathbf{R}\mathbf{x}_w + \mathbf{t}, \tag{3}$$

where $\mathbf{x}_c$ is the point in the camera's coordinate system [18]. Moreover, when we are dealing with images we usually consider the image plane in terms of pixel

coordinates. By projecting a scene point $\mathbf{x}_w$ to a point $\mathbf{x}_{im}$ in the image plane we do not get this immediately. The pixel coordinates can be found by mapping the image plane to the "pixel plane". This mapping can be done linearly in the pinhole camera model with the so called intrinsic parameters of the camera that we store in a 3×3 matrix, usually denoted by $\mathbf{K}$. The whole projection from a 3D scene point $\mathbf{x}_w \in \mathbb{R}^3$ in world coordinates to an image point $\mathbf{x}_p \in \mathbb{R}^2$ in pixel coordinates can now be expressed as a single matrix multiplication:

$$\mathbf{P} \begin{pmatrix} \mathbf{x}_w \\ 1 \end{pmatrix} = \mathbf{K} \begin{pmatrix} \mathbf{R} & \mathbf{t} \end{pmatrix} \tilde{\mathbf{x}}_w = \lambda \begin{pmatrix} \mathbf{x}_p \\ 1 \end{pmatrix} = \lambda \tilde{\mathbf{x}}_p, \tag{4}$$

where $\tilde{\mathbf{x}}_w$ and $\tilde{\mathbf{x}}_p$ are the world and pixel-coordinate expressed in homogeneous coordinates. The matrix $\mathbf{P}$ in Equation (4) is often referred to as the *projection matrix* and consists of the *intrinsic matrix* $\mathbf{K}$ and the *extrinsic matrix* $[\mathbf{R}\ \mathbf{t}]$. Here $\mathbf{R}$ is a unitary rotation matrix and $\mathbf{t}$ the translation vector from Equation (3). We define the intrinsic parameters $\mathbf{K}$ of the camera as

$$\mathbf{K} = \begin{pmatrix} rf & \alpha f & u_0 \\ 0 & f & v_0 \\ 0 & 0 & 1 \end{pmatrix}. \tag{5}$$

In Equation 5, f is the camera's focal length, r is the aspect ratio, α is the skew coefficent between the x- and y-axis and (u_0, v_0) is the camera's principal point. In this formulation, we use homogeneous coordinates, and the λ is introduced because the scale is arbitrary. So, in order to get the pixel coordinate $\mathbf{x}_p$ after multiplying the world coordinate $\tilde{\mathbf{x}}_w$ with the projection matrix $\mathbf{P}$, we need to normalise the output vector's third coordinate to 1 in order to project the coordinate onto the image plane.

The pinhole model is not a good model for all cameras and is an approximation of how a camera work. Some phenomena that this model does not take into consideration is lens distortions and blurring. The former commonly correlates strongly with the distance to the centre of the image and is in those cases called radial distortion. However, without having to resort to more difficult models, this model is enough for our purposes and for the cameras used in this thesis. This concludes the introduction to the pinhole camera model and we now move on to how we can find the sought after projection matrix $\mathbf{P}$.

3.1.3 Camera Calibration

To be able to transfer detections from a camera to world coordinates, the camera's projection matrix $\mathbf{P}$ needs to be estimated for each camera. There exists a handful of methods to find the projection matrix. Furthermore, a camera is called calibrated if its intrinsic parameters are known. We use *Direct Linear Transformation* (DLT) to calibrate the camera as in [29]. With this direct method, a homogeneous linear system of equations is constructed. This system might not be solvable directly due to imprecise point matches so we solve it in a least square sense. In order to do this

we need a set of at least six point matches $(\tilde{\mathbf{x}}_p^i, \tilde{\mathbf{x}}_w^i)$ between the homogeneous world coordinates $\tilde{\mathbf{x}}_w^i$ and the homogeneous pixel coordinates $\tilde{\mathbf{x}}_p^i$. Given that we have m point matches $(\tilde{\mathbf{x}}_p^i, \tilde{\mathbf{x}}_w^i)$ with a corresponding λ_i we can construct a system of linear equations from Equation 4:

$$\mathbf{Av} = \begin{pmatrix} \mathbf{B}_1 & -\tilde{\mathbf{x}}_p^1 & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{B}_2 & \mathbf{0} & -\tilde{\mathbf{x}}_p^2 & \dots & \mathbf{0} \\ \vdots & \mathbf{0} & \mathbf{0} & \ddots & \mathbf{0} \\ \mathbf{B}_m & \mathbf{0} & \mathbf{0} & \dots & -\tilde{\mathbf{x}}_p^m \end{pmatrix} \begin{pmatrix} \mathbf{p}_1^T \\ \mathbf{p}_2^T \\ \mathbf{p}_3^T \\ \lambda_1 \\ \vdots \\ \lambda_n \end{pmatrix} = \mathbf{0}, \tag{6}$$

where $\mathbf{p}_i$ are the rows of the projection matrix, $\tilde{\mathbf{x}}_p^i$ are the homogeneous pixel coordinates of the image corresponding to the homogeneous real world coordinate $\tilde{\mathbf{x}}_w^i$ and $\mathbf{B}_i$ are 3×12 block matrices defined as

$$\mathbf{B}_i = \begin{pmatrix} (\tilde{\mathbf{x}}_w^i)^T & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & (\tilde{\mathbf{x}}_w^i)^T & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & (\tilde{\mathbf{x}}_w^i)^T \end{pmatrix}.$$

The solution to the corresponding least square problem $\min ||\mathbf{Av}||^2$ is the eigenvector of $\mathbf{A}^T\mathbf{A}$ with the smallest eigenvalue. The derivation of this can be found by using Lagrange multipliers [29]. The smallest eigenvalue of $\mathbf{A}^T\mathbf{A}$ can be computed with a Singular Value Decomposition $\mathbf{A} = \mathbf{USV}^T$ and the solution is the column of $\mathbf{V}$ corresponding to the smallest singular value.

After the camera matrix $\mathbf{P}$ has been estimated, the intrinsic and extrinsic matrices can be determined by RQ-decomposition. We note that $\mathbf{P} = [\mathbf{KR} \mid \mathbf{Kt}]$ and use the RQ-decomposition of $\mathbf{KR}$ into an orthogonal and a upper triangular matrix to find the intrinsic and extrinsic matrices. The upper triangular part corresponds to the intrinsic matrix, $\mathbf{K}$, and the orthogonal rotation matrix corresponds to the extrinsic rotation matrix, $\mathbf{R}$.

3.1.4 Inverse Projection

We now go on to describe how we project points in the image space onto a ground plane in the global coordinate system. This is used for example for data association in the multi-camera tracker. When a camera's projection matrix $\mathbf{P} = \mathbf{K}[\mathbf{R}\ \mathbf{t}]$ is known, it is possible to transform pixel coordinates in the image space back to a line in world coordinates.

By knowing the camera centre $\mathbf{c}$, we can more easily express the inverse projections. Since the camera centre $\mathbf{c}$ in the camera coordinates is located at the the origin, we can find the coordinate in world coordinates from Equation (3) by setting $\mathbf{x}_c = \mathbf{0}$. This gives us the equation

$$\mathbf{0} = \mathbf{Rc} + \mathbf{t},$$

where $\mathbf{c}$ is the camera origin in global world coordinates. Since $\mathbf{R}$ is a unitary rotation matrix we get that

$$\mathbf{c} = -\mathbf{R}^{\mathbf{T}}\mathbf{t}. \tag{7}$$

In order to go from a pixel coordinate $\mathbf{x}_p \in \mathbb{R}^2$ to a set of world coordinates $\mathbf{x}_w \in \mathbb{R}^3$ we need to solve the inverse projection problem. From Equation (4), we solve for the world coordinate $\mathbf{x}_w$

$$\mathbf{K}(\mathbf{R}\mathbf{x}_w + \mathbf{t}) = \lambda\tilde{\mathbf{x}}_p \tag{8}$$

$$\mathbf{R}\mathbf{x}_w + \mathbf{t} = \lambda\mathbf{K}^{-1}\tilde{\mathbf{x}}_p \tag{9}$$

$$\mathbf{x}_w = \lambda\mathbf{R}^T\mathbf{K}^{-1}\tilde{\mathbf{x}}_p - \mathbf{R}^T\mathbf{t}, \tag{10}$$

where $\mathbf{x}_w$ is the Cartesian world coordinate and $\tilde{\mathbf{x}}_p$ is the homogeneous pixel coordinate and $\lambda \in \mathbb{R}$. Here, $\mathbf{x}_w$ in Equation (10) is a line on parametric vector form representing all world coordinates that projects to the image point $\mathbf{x}_p$. This line l can be formulated using the camera centre, $\mathbf{c}$ as expressed in Equation (7), and a ray direction, $\mathbf{r}$,

$$l(\lambda) = \mathbf{c} + \lambda\mathbf{r}, \tag{11}$$

where $\mathbf{r} = \mathbf{R}^T\mathbf{K}^{-1}\tilde{\mathbf{x}}_p$ is the ray direction in world coordinates. This line can then be projected onto a ground plane by finding the intersection between the ground plane and the line defined by equation (10). Assuming that the ground plane is $z = 0$, the intersection with the ground plane is

$$\lambda = -\frac{c_z}{r_z}, \tag{12}$$

where r_z is the z-coordinate of $\mathbf{r}$, and c_z is the z-coordinate of the camera center, $\mathbf{c}$. The projection onto the same plane in Cartesian coordinates becomes

$$\mathbf{x}_w = l(-\frac{c_z}{r_z}) = \mathbf{c} - \frac{c_z}{r_z}\mathbf{r}. \tag{13}$$

This method can be used to transform a detection in image coordinates to world coordinates for data association, and to update the tracks' states if only a single camera detects the person. If on other hand multiple cameras detect the same person, we can use triangulation to perform data fusion, described in the next section.

3.1.5 Sensor Fusion by Triangulation

To fuse multiple image points $\mathbf{x}_p^k$ in different cameras into one common point $\mathbf{x}_w$ in the 3D world coordinate system, we use triangulation. The problem can be formulated with Equation (4) where $\mathbf{x}_w$ is an unknown world point and the camera

projection matrices $\mathbf{P}^k$ are different. Given that we have m cameras we can construct the system of equations

$$\mathbf{Ap} = \begin{pmatrix} \mathbf{p}_1^1 & -x_p^1 & 0 & \cdots & 0 \\ \mathbf{p}_2^1 & -y_p^1 & 0 & \cdots & 0 \\ \mathbf{p}_3^1 & -1 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & & \vdots \\ \mathbf{p}_1^m & 0 & 0 & \cdots & -x_p^m \\ \mathbf{p}_2^m & 0 & 0 & \cdots & -y_p^m \\ \mathbf{p}_3^m & 0 & 0 & \cdots & -1 \end{pmatrix} \begin{pmatrix} x_w \\ y_w \\ z_w \\ 1 \\ \lambda_1 \\ \vdots \\ \lambda_n \end{pmatrix} = \mathbf{0},$$

where $\mathbf{p}_i^k$ is the i:th row of the projection matrix from camera k, x_p^k, y_p^k are the pixel coordinates of the image point $\mathbf{x}_p^k$ in camera k and x_w, y_w, z_w are the corresponding world coordinate we seek to find. This system might not be solvable due to the accuracy of the calibrated cameras $\mathbf{P}^k$ or errors made in finding the point correspondences in image space. Thus we instead solve it in least square sense and try to find $\min_v ||Av||^2$. The solution can be found by using an SVD of $\mathbf{A}$ like in [29].

3.2 YOLO Detections

Given an image the detection method should identify the bounding boxes of all objects in the image and provide a class for each bounding box. Each bounding box should also get probabilities of belonging to each class that the detection method (detector) can detect. The assigned class label is the class with the highest class probability. An illustration of bounding box detection is shown in Figure 3.

Figure 3: Illustration of bounding box detections. The detector outputs a number of bounding boxes with associated classes and probabilities.

The *You Only Look Once* (YOLO) detection model is a unified model for bounding box detection in images, where a single convolutional neural network (CNN) generates detections of the objects in the image. The architecture of the model enables fast classifications and makes the model suitable for real-time applications and was first proposed by Joseph Redmon et al. [32]. Other methods like R-CNN [14] first generate a number of proposal bounding boxes and then use a classifier on these proposals to detect objects in the scene. Although R-CNN methods have been successful, they are quite slow and thus not as suitable for real-time applications with a high frame rate [32]. The YOLO network instead views the detection as a regression problem and simultaneously proposes bounding boxes and the corresponding class probabilities. After a brief introduction to neural networks we will present the YOLO architecture in detail.

3.2.1 Feed-Forward Neural Networks

A neural network f_θ is a non-linear function approximator that is also referred to as a multi-layer peceptron. The basic building blocks of the neural network are the so called perceptrons. The original perceptron is a function that works as a binary classifier [19]. From an input $\mathbf{x} \in \mathbb{R}^n$ the perceptron outputs either a 0 or a 1 corresponding to the two possible classes the input $\mathbf{x}$ could have belonged to. The function can be written as

$$f(\mathbf{x}) = \begin{cases} 1 & \mathbf{w}^T\mathbf{x} + b > 0 \\ 0 & \text{otherwise,} \end{cases} \tag{14}$$

where $\mathbf{w} \in \mathbb{R}^n$ is a weight vector and $b \in \mathbb{R}$ is called the bias term [19]. The input $\mathbf{x} \in \mathbb{R}^n$ is referred to as a feature vector (some numeric representation of the object we are interested in). Neural networks are structured in ordered layers of perceptrons. The feed-forward nature of the network comes from the fact that each layer takes the output from the previous layers as input and feeds forward the output to the next layer. One does not usually make the output of the perceptron binary in a neural network. Instead it is common to apply a differentiable *activation function* σ to $\mathbf{w}^T\mathbf{x} + b$ that bounds the output of the perceptron. This gives a more smooth decision boundary and the added differentiability of the perceptron is very useful when one wishes to learn the parameters of the network [19].

As mentioned the network is ordered in layers. This allows us to to view the network as a composition of several functions, where each function is represented by a layer. We have three kinds of layers, namely the input layer, the hidden layers and the output layer. The input layer simply takes the input and forwards it to the next layer. The hidden layer comprises of a set of perceptrons that are often called neurons. These neurons are connected to the neurons in both the previous layer and the subsequent layer with so called edges. Each edge has a corresponding weight $w \in \mathbb{R}$ associated to it. The perceptron applies a non-linearity σ to the weighted sum of its input vector. This input vector to the perceptron is the output from the

previous layer. More formally the perceptron j in layer k outputs

$$o_j^k = \sigma(b_j^k + \sum_i w_i^k x_i^k),$$

where b_j^k is the neuron's bias, w_i^k the weight from the incoming edge from perceptron i in the previous layer and the input x_i^k is the output o_i^{k-1} from perceptron i in the previous layer $k-1$. The network is a composition of its layer f_k with

$$f_k(\mathbf{x}_{k-1}) = \begin{bmatrix} o_1^k \\ \vdots \\ o_j^k \\ \vdots \\ o_{n_k}^k \end{bmatrix}, \tag{15}$$

where n_k is the number of neurons and o_j^k is the output from the j:th neuron in layer k. The output of the final layer thus becomes

$$\mathbf{y} = f_n \cdots \circ f_k \cdots \circ f_1(\mathbf{x}_0),$$

where $\mathbf{x}_0$ is the input to the input layer, $\mathbf{y}$ the output and each f_k represents the output from each layer. The final layer is what is called the output layer. Figure 4 below illustrates this feed-forward structure.

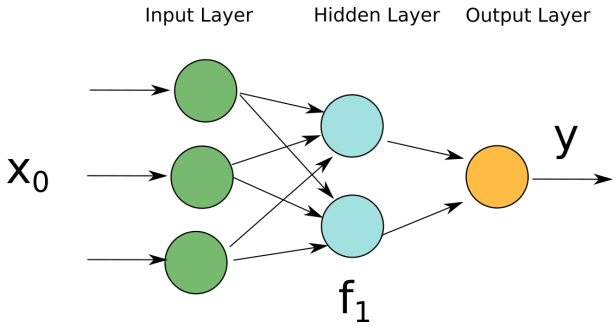


Figure 4: A small feed forward network.

The weights and biases of the neural network are optimised to minimise a loss function $\mathcal{L}$. We call the process of optimising the parameters training. A typical loss function in classification tasks is the sum of the squared errors

$$\mathcal{L} = \sum_{i=1}^{n} (\mathbf{y}_i - \hat{\mathbf{y}}_i(\theta))^2,$$

where $\mathbf{y}_i$ is the ground truth of object i, n is the number of elements in our data set and $\hat{\mathbf{y}}_i$ is the output and θ the parameters (weights and biases) of the network. The ground truth labels are the true classifications of the elements in our data set.

Given a data set $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^N$ that consists of input vectors $\mathbf{x}_i$ and ground truth labels $\mathbf{y}_i$ of said input vectors, a loss function $\mathcal{L}$ measures the error of the model (neural network) on the data set.

The training, or optimisation of the network parameters θ is often done using *Stochastic Gradient Descent* (SGD). One usually splits the data set into a training set and test set. The training set is used to train the model and the test set is used to evaluate the model. SGD is an iterative gradient descent method that estimates the gradient of the loss function on a small batch of the data. The data batch is a subset of the training data chosen at random. The minimisation in the vanilla SGD takes the form

$$\theta^{i+1} = \theta^i - \eta \nabla \mathcal{L}(\theta^i), \tag{16}$$

where η is called the learning rate. In gradient descent methods we take a step in the negative direction of the gradient. The step size is what we referred to as the learning rate. The method works by randomly selecting batches and taking a small step in the negative direction of the gradient over the loss function defined on this batch. There are many variations of SGD and a common choice is the ADAM method [23].

3.2.2 Convolutional Neural Networks

In a fully connected layer each node in each layer is connected to all the nodes in the surrounding layers. Image data is usually very large and the number of weights in a fully connected network grows extremely fast as its depth increases. A way to mitigate this in image data is to use convolutions in each layer instead of fully connected layers. Convolutional filters extract local features in an image and feeds it forward to the next layer as input. This means that the perceptron only receives input from a local subset of the previous perceptrons. The convolution thus replaces the weighted average $\mathbf{w}^T\mathbf{x}$ part in Equation 14. A convolution of a gray-scale image is defined as follows:

$$y(i,j) = \sum_u \sum_v x(i-u, j-v) w(u,v),$$

where w is the filter kernel and x is the image. Intuitively, the convolution can be viewed as sliding a window over the image and multiplying its weights with the pixels' weights extracting local features resulting in a smaller input size to the next layer. We now give a few examples of the most popular activation functions.

3.2.3 Activation Functions and Non-Linearities

One of the most common activation functions is the sigmoid function defined as

$$\sigma(x) = \frac{1}{1+e^{-x}}. \tag{17}$$

This function is intended to mimic neuron activation in the human brain. As signals are sent through synapses, the neuron fires only once it has received a strong enough input signal [19]. According to Equation (17), it outputs a very small number for input values of $x < 0$ and the function quickly approaches 1 for $x > 0$.

Another commonly used activation is the *Rectified Linear Unit* (ReLU) activation function. It is less computationally heavy than the sigmoid function and reduces the vanishing gradient problem. The vanishing gradient problem occurs when we update the parameters of the network with the chain rule. In the chain rule we multiply the partial derivatives of each layer together and if these partial derivatives are small the resulting gradient to update the network will also be small making the network difficult to train [19]. As can be seen from Equation (17) the gradient of the sigmoid quickly become small for x far away from the 0. The ReLU function is defined as

$$\text{ReLU}(x) = \max(0, x).$$

Another variant is the leaky ReLu which allows for small gradients when $x < 0$ that further helps reduce the vanishing gradient problem. As the gradient of the normal ReLU function is zero when $x < 0$, the neurons that get input with $x < 0$ die and will not be able to be updated. The leaky version rectifies this and is defined as

$$f(x) = \begin{cases} x & x > 0 \\ 0.01x & \text{otherwise.} \end{cases}$$

The final non-linearity that we present is the max-pooling layer. This is a way to downsample and the remove the dependence on translations of the inputs. It is similar to a filter and works by first deciding on a size of the pooling-kernel. The filter then slides over the image and outputs the maximum value of the features outputted by the previous layer [24]. This has the effect of reducing the dimension of the output from a layer. This can be visualised in Figure 5 below were the input in a layer is reduced by max-pooling.

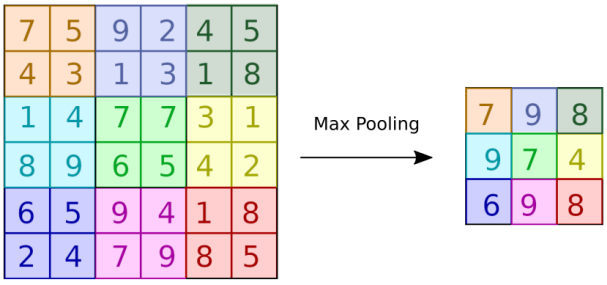


Figure 5: Example of max-pooling with a 2×2 kernel and stride 2.

From Figure 5 we see how the input dimension of the 6×6 input is reduced to a 3×3 image. This filtering can be done with different strides allowing for overlap between input it slides over. In Figure 5 above we have a stride of 2, meaning that we move the pooling kernel with a step of 2 in both the x and y direction of the image.

3.2.4 The YOLO Architecture

The YOLO detection model is a unified detection model and consists of a single convolutional neural network, unlike R-CNN models that process the image in multiple passes. The YOLO architecture takes a square 3-channeled RGB image of a fixed size as input. In our case, the size of the input image is $608 \times 608 \times 3$. This image is passed through 7 convolutional layers followed by two fully connected layers leading up to an output layer. All detections and classifications are generated directly from this output.

The image is divided into $S \times S$ equally sized grid cells. For each grid cell, the network predicts B bounding boxes b with an associated confidence value, c. So a bounding box is in this setting defined by the five parameters

$$b = (x, y, w, h, c),$$

where x and y are the pixel coordinate of the bounding box's center coordinates, w and h are the width and height of the bounding box, and c is the confidence of each bounding box. The confidence c represents the *Intersection Over Union* (IOU) distance between the predicted bounding box and the ground truth bounding boxes used in training. The IOU is a measure of how similar the bounding boxes are and is defined as

$$IOU = \frac{A(b_1 \cap b_2)}{A(b_1 \cup b_2)},$$

where $A(\cdot)$ is the area function and b_1, b_2 are the compared bounding boxes. The measure is illustrated in Figure 6.

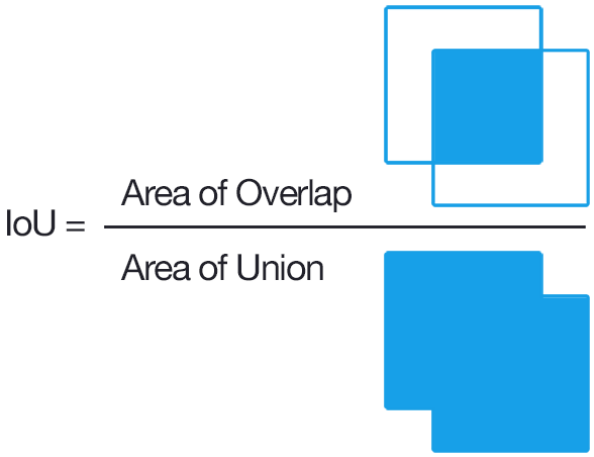


Figure 6: Illustration of the intersection over union.

Apart from B bounding boxes, each grid cell also produces C class probabilities. These probabilities represent the conditional probability that an object is of a specific class, given that there is an object in the grid cell. By filtering out low confidence bounding boxes and combining the remaining ones with the grids' class probabilities, the resulting detections are obtained. The process is illustrated in Figure 7.

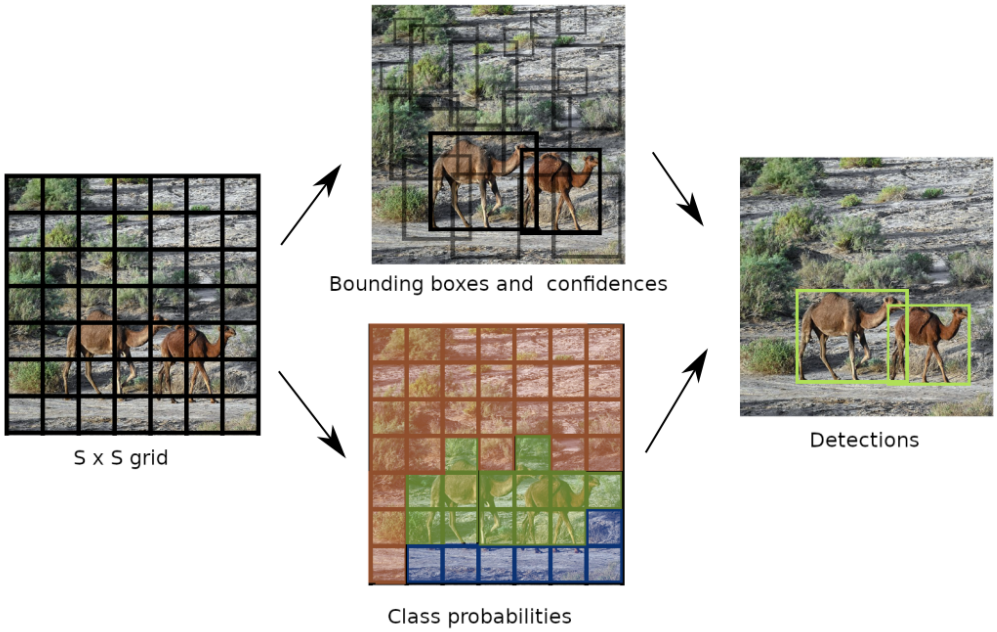


Figure 7: Illustration of the YOLO pipeline from image to detections.

In our case, the grid parameter S is set to 7 and the number of bounding boxes per grid cell B is set to 2. The maximum number of objects that theoretically could be detected in a frame is therefore $7 \times 7 \times 2 = 98$. Since only two bounding boxes are generated for each grid, this prevents the YOLO architecture from detecting many small objects located close together in the scene. The YOLO model is open source and is currently available in its 4th incarnation, YOLO-v4 [3]. For the YOLO-v4 network, the number of classes C is 80. Even though we only use the class *person* in this thesis, the same pipeline could be used to track other classes of objects.

3.2.5 Non-Maximum Suppression

Due to the structure of YOLO, sometimes multiple bounding boxes could be generated from a single object. To get a more accurate detection system, we need a method to filter out duplicated detections. A commonly used greedy algorithm to do this is the *Non-maximum supression* (NMS) method [35].

The NMS algorithm is the following Algorithm 1:

Algorithm 1 Non-Maximum Suppression

1. Place all detected bounding boxes in a proposal set $\mathcal{P}$ and let $\mathcal{D}$ be an initially empty set for the filtered bounding boxes and define a threshold N for the maximum IOU.
2. Remove the bounding box b_i with highest confidence c_i from $\mathcal{P}$ and add it to the set of filtered bounding boxes $\mathcal{D}$.
3. Calculate the IOU between the selected bounding box b_i and every bounding box in the proposal set $\mathcal{P}$ and remove bounding boxes with IOU:s higher than the pre-defined threshold N from $\mathcal{P}$.
4. Repeat the process from 2 until no bounding boxes remain in the set $\mathcal{P}$.

3.3 Kalman Filtering

A common approach to update the estimated states of the tracked objects is to use Kalman filtering. In this thesis we use this filter both for tracks in pixel coordinates and for tracks in the global coordinate system. The Kalman filter is the best linear unbiased estimator and thus the optimal choice under linear and Gaussian assumptions [22]. In the context of tracking we assume that no control signal is present, so this will be omitted in the following presentation. The dynamic model can be described by the equations

$$\mathbf{x}_k = \mathbf{A}_k \mathbf{x}_{k-1} + \mathbf{q}_k \tag{18}$$

$$\mathbf{y}_k = \mathbf{H}_k \mathbf{x}_k + \mathbf{r}_k. \tag{19}$$

where $\mathbf{x}_k \in \mathbb{R}^m$ is the hidden state that we wish to track, $\mathbf{y}_k \in \mathbb{R}^n$ the noisy measurements, $\mathbf{A}_k$ the $m \times m$ state transition matrix, $\mathbf{H}_k$ the $n \times m$ observation matrix and $\mathbf{q}_k \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}_k)$ and $\mathbf{r}_k \sim \mathcal{N}(\mathbf{0}, \mathbf{R}_k)$ are m and n dimensional multivariate Gaussian distributions, respectively, with corresponding covariance matrices $\mathbf{Q}_k$ and $\mathbf{R}_k$. The Kalman filter proceeds by alternating between making predictions and updating the beliefs of the states by using the received measurements. In the prediction step, the previous state $\mathbf{x}_{k-1}$ is filtered through the transition model and updates the state covariance matrix accordingly, as shown by Equation 20,

$$\begin{aligned} \hat{\mathbf{x}}_{k|k-1} &= \mathbf{A}_k \mathbf{x}_{k-1} \\ \mathbf{P}_{k|k-1} &= \mathbf{A}_k \mathbf{P}_{k-1|k-1} \mathbf{A}_k^T + \mathbf{Q}_k, \end{aligned} \tag{20}$$

where $\mathbf{P}_{k-1|k-1}$ is the previous covariance matrix for our hidden states $\mathbf{x}_{k-1}$. Once the prediction step is done, we update our belief of the current state with the new measurement. The first part of the update step is to calculate the innovation residual $\tilde{\epsilon}_k$ (difference between prediction and measurement) and its covariance $\mathbf{S_k}$. This is given by

$$\begin{aligned} \tilde{\epsilon}_k &= \mathbf{y}_k - \mathbf{H}_k \hat{\mathbf{x}}_{k|k-1} \\ \mathbf{S}_k &= \mathbf{H}_k \mathbf{P}_{k|k-1} \mathbf{H}_k^T + \mathbf{R}_k, \end{aligned} \tag{21}$$

where $\tilde{\epsilon}_k$ is the innovation and the $\mathbf{S}_k$ is the innovation covariance. The covariance update in Equation 21 for $\mathbf{S}_k$ follows from Equation 19. Once the innovation and its covariance have been calculated the Kalman gain $\mathbf{K}_k$ is used to update the estimate of the state and the state covariance by

$$\begin{aligned} \mathbf{K}_k &= \mathbf{P}_{k|k-1}\mathbf{H}_k^T\mathbf{S}_k^{-1} \\ \hat{\mathbf{x}}_{k|k} &= \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k\tilde{\epsilon}_k \\ \mathbf{P}_{k|k} &= (\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)\mathbf{P}_{k|k-1}. \end{aligned} \tag{22}$$

With these equations we can now receive measurements from our cameras and update our beliefs of the positions of the pedestrians. This concludes the presentation of the Kalman filter. It will be used extensively in this thesis to update the beliefs of the positions of the pedestrians, both locally in the pixel coordinates of each camera and globally in world coordinates.

3.4 Data Association and the Hungarian Algorithm

When we now have methods to detect humans (Section 3.2) in images and to transform these detections to a common coordinate system (Section 3.1.4), we need a method to associate new measurements (i.e, detections) to existing tracks in real-time. Thus, a fast algorithm for solving the so called *Assignment problem* is needed. A commonly used algorithm to perform association in tracking problems is the Hungarian algorithm. This algorithm is based on the work of the two Hungarian mathematicians Dénes Kőnig and Jenő Egerváry and was first described by Harold Kuhn in a 1955 paper [25].

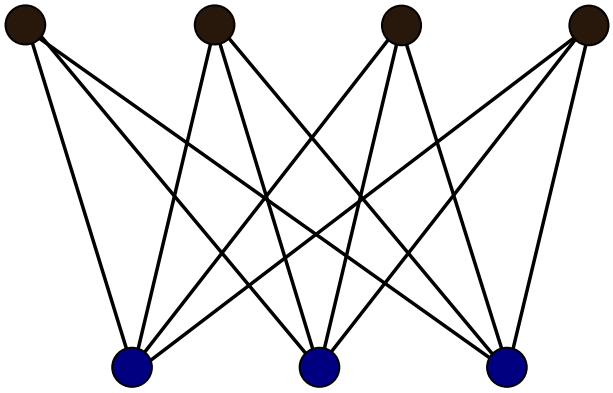


Figure 8: A complete bipartite graph. The top vertices and bottom vertices represent two independent and disjoint sets where every vertex in one set is connected to every vertex in the other set.

Given a weighted complete bipartite graph, as shown in Figure 8, the Hungarian algorithm will find the association between the elements of the two sets $\mathcal{A}$ and $\mathcal{B}$

that minimises the sum of the weights of the associated edges. This assignment problem can formally be expressed as

$$\min_{f} \sum_{a \in \mathcal{A}} \mathbf{C}(a, f(a)), \tag{23}$$

where $a \in \mathcal{A}$ and f is a bijection from $\mathcal{A}$ to $\mathcal{B}$. If an element $a \in \mathcal{A}$ is mapped to $b = f(a) \in \mathcal{B}$ we say that a is assigned to b. With the Hungarian algorithm, this minimisation can be implemented in polynomial time with worst case time complexity of $\mathcal{O}(n^3)$, where n is the number of vertices in the largest set [25]. In our case, one set in the bipartite graph consist of the tracks' predicted state, $\hat{\mathbf{x}}_i$. The other set consist of the received detections $\mathbf{d}_i$. The weights on these edges will be determined by the distance between the track's prediction and the new measurement. In the SORT algorithm, the negative intersection over union distance is used. This distance is defined as

$$IOU_d = -\frac{A(b_1 \cap b_2)}{A(b_1 \cup b_2)}, \tag{24}$$

where $A(\cdot)$ is the area function and b_1, b_2 are the concerned bounding boxes. Note that since the Hungarian algorithm is formulated as a minimisation problem, the IOU is negated so that a smaller IOU_d becomes preferable.

In the global coordinates, we use the Euclidean distance although other distance measures, like the Mahalanbobis distance are sometimes used [8]. To solve the assignment problem, we use the matrix formulation of the algorithm and let the square, $n \times n$, cost matrix $\mathbf{C}$ represent an adjacency matrix for a bipartite graph. If the number of detections differ from the number of predictions in a frame, we use zero-padding dummy values to create the square matrix, $\mathbf{C}$. If we permute the columns and the rows of the cost matrix, $\mathbf{C}$, so that the resulting assignment costs appear on the diagonal of the matrix, the problem will be equivalent to finding the permutation matrices $\mathbf{L}$ and $\mathbf{R}$ that minimise the trace of the permutated matrix,

$$\min_{\mathbf{L},\mathbf{R}} Tr(\mathbf{LCR}). \tag{25}$$

The algorithm finds a minimal matching for an $n \times n$ matrix and is described in Algorithm 2 below.

Algorithm 2 Hungarian algorithm on the $n \times n$ matrix $\mathbf{C}$

1. Subtract the smallest element of each row from every element in that row
2. Subtract the smallest element of each column from every element in that column
3. Cover all zeros in the matrix $\mathbf{C}$ by drawing horizontal and vertical lines so that a minimum number of lines are used. If n lines were needed, encircle n zeros so that the encircled zeros are the only encircled zeros on each row and column. These are the resulting assignments. If less than n lines were needed, continue to the next step.
4. Find the smallest entry not covered by any line and remove it from each row that is not crossed out and add it to each column that is crossed out. Then go to Step 3.

The algorithm is illustrated by the following example, using a 3×3 cost matrix.

$$\begin{pmatrix} 7 & 4 & 5 \\ 3 & 8 & 9 \\ 11 & 15 & 12 \end{pmatrix} \xrightarrow{1} \begin{pmatrix} 3 & 0 & 1 \\ 0 & 5 & 6 \\ 0 & 4 & 1 \end{pmatrix} \xrightarrow{2} \begin{pmatrix} 3 & 0 & 0 \\ 0 & 5 & 5 \\ 0 & 4 & 0 \end{pmatrix} \xrightarrow{3} \begin{pmatrix} \cancel{3} & \cancel{0} & \cancel{0} \\ 0 & 5 & 5 \\ 0 & 4 & 0 \end{pmatrix} \rightarrow \begin{pmatrix} 3 & \textcircled{0} & 0 \\ \textcircled{0} & 5 & 5 \\ 0 & 4 & \textcircled{0} \end{pmatrix}$$

The encircled elements at positions i, j in the rightmost matrix corresponds to the resulting assignment of element i in the first set with element j in the second set. The minimal sum of association costs is in this example $4 + 3 + 12 = 19$. This concludes the overview of the data association algorithm.

3.5 Reinforcement Learning

In this section we will give a short introduction to *Reinforcement Learning* (RL). Reinforcement learning is a subset of machine learning that deals with agents interacting with an environment sequentially. The goal of the agent is to maximise a utility function that describes the task that the agent should perform. The utility functions can describe different tasks, for example winning a game of chess.

As the agent interacts with the environment it gets feedback based on the actions it takes in different states. This feedback comes in the form of a scalar reward for each action taken and the problem is formulated as a maximisation problem over the accumulated rewards that an agent can get. This interaction between the environment and the agent is illustrated by Figure 9.

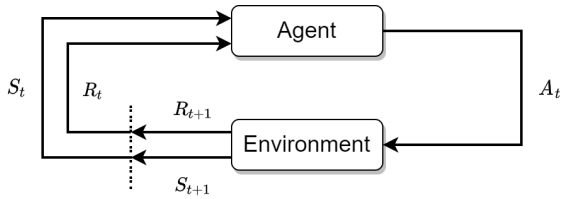


Figure 9: Agent-environment interaction

Reinforcement learning solves an unknown Markov Decision Process. The agent starts in some state $S_0 \in \mathcal{S}$ that is defined by the environment where $\mathcal{S}$ is the set of all possible states. It then takes some action $a \in \mathcal{A}$ with $\mathcal{A}$ being the set of all possible actions. Following this action it receives a scalar reward $R_1 \in \mathcal{R} \subset \mathbb{R}$ and a new state S_1. This procedure continues as time goes on and produces a trajectory [38]

$$S_0, A_1, R_1, S_1, A_2, R_2 \ldots$$

The outcome of an action only depends on the current state and not on previous states. This is the Markov property

$$P(S_{t+1} = s' | S_t = s_t, A_t = a_t, A_{t-1} = a_{t-1}, \ldots S_0 = s_0) =$$
$$P(S_{t+1} = s' | S_t = s_t, A_t = a_t).$$

This means that the probability that we end up in state s' given that we have visited all states $s_0, ...s_t$ and taken actions $a_1, ...a_t$ only depends on the previous action and state pair (s_t, a_t) [30]. With this formulation we have introduced a new concept, namely the transition probability P. This transition probability tells us that our actions might not be deterministic. We may thus have a distribution of next states S_{t+1} that we end up in after having taken action a_t in state s_t. The goal of the agent is to maximise the rewards that it receives. In fact, we are interested in optimising the expected value of all future rewards. So our target objective function to maximise is

$$J = \mathbb{E}\big[\sum_{t=0}^{\infty} \gamma^t r_t\big]. \tag{26}$$

Notice that we have added a so called discount factor $\gamma \in (0, 1)$ that is used to prioritise immediate rewards over future rewards. We are more interested in the current reward than future rewards that we may potentially get. In the so called infinite time horizon the discount factor is required. If an agent can continue to take actions forever this could potentially result in an infinite sum of rewards (see Equation 26) and make the problem unfeasible. For the finite time horizon however, we still might want a discount factor to determine how much we value rewards in the future compared to now. With a discount factor of $0 < \gamma < 1$ the future rewards will decay exponentially.

The agents take actions according to a policy. The policy that we want to learn takes the form

$$\pi : \mathcal{A} \times \mathcal{S} \to [0, 1], \tag{27}$$

where $\pi(a, s)$ is the probability of taking an action a when in the state s.

In this thesis we use the Proximal Policy Optimisation (PPO) [37] method that has recently been found to outperform other methods in multi-agent reinforcement problems [48]. With this method we immediately optimise the parameterised policy. The policy is parameterised by a neural network. This makes use of something called the advantage function and for this we need the definition of both the value function and the Q-value function. The value function is defined as follows

$$V^{\pi}(s) = E_{\pi}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ... \mid S_t = s_t] = E_{\pi}[J \mid S_t = s],$$

and is the expected sum of the discounted future rewards, given that we are in state s_t and use policy π to take actions. This can be viewed as an estimate of how good it is for the agent to be in state s_t. The Q-value function is defined as

$$Q_{\pi}(s_t, a_t) = E_{\pi}[J \mid A_t = a_t, S_t = s_t]. \tag{28}$$

where J is the sum of the discounted rewards as defined in Equation (26). The Q-value gives us the expected total discounted future reward given that we are in state s_t and taking action a_t under policy π. The value function and Q-value function are often approximated using neural networks.

For completeness we also present the equations to infer a policy from the Q-values and the value function. The Bellman equations [30] gives us a policy through the value function by

$$\pi^*(s) = \underset{\text{a}}{argmax} \sum_{s'} P(s,a,s')[R(s,a,s') + \gamma V^*(s')]. \tag{29}$$

This means that we choose the action that maximises the reward R_t in the current step t and the discounted sum of future rewards. Again $P(s,a,s')$ is the transition probability. In a similar fashion we can infer the policy from the Q-values by

$$\pi^*(s) = \underset{\text{a}}{argmax} \quad Q^*(s,a). \tag{30}$$

However, in this thesis we use the proximal policy optimisation method. We thus immediately try to find the policy by approximating it with a neural network. To be able to understand this method we first need to introduce the concept of advantage. The advantage is defined as follows

$$A(s,a) = Q(s,a) - V(s). \tag{31}$$

The advantage can be interpreted as how much we gain from taking action a in state s compared to the "average" action. This follows from that the Q-value gives us the expected total discounted reward from taking action a in state s and that the value $V(s)$ gives us a measure on how good this state s is. With these concepts we now briefly explain the PPO algorithm.

3.5.1 Proximal Policy Optimization

The *Proximal Policy Optimization* (PPO) by Schulman, et al. [37] is a recent reinforcement learning algorithm that aims at finding the policy function directly. The algorithm is based on *Trust Region Policy Optimization* (TRPO), which is another policy gradient method with guaranteed monotonic improvement [36].

The *Trust Region Policy Optimization* defines a trust region that guarantees that the policy distribution does not change too quickly. This is done by subjecting the objective function to a Kullback Leibler-divergence (KL-divergence) constraint. This constraint ensures that the old policy that we sampled from does not change too much when we do a gradient descent update.

The PPO is similar to TRPO but optimises a lower bound of the objective function and is simpler to implement. Instead of employing a trust region with a KL-divergence constraint it bounds the gradients to ensure that the policy does not change too rapidly.

It is common to use neural networks for policy functions and to estimate the advantage values. Usually one has two neural networks to achieve this, one policy network and one value network. PPO utilises the following loss function

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[\min \left(r_t(\theta)\hat{A}_t, \text{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t \right) \right], \tag{32}$$

where θ is the parameters of the policy. The term r_t is defined as

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}.$$

which is the ratio between the probability distributions of the policy being updated π_θ and the old policy $\pi_{\theta_{old}}$. This term arrives from the fact that PPO (and TRPO) performs importance sampling and allows us to reuse samples gathered with the previous policy in an on-policy RL-method. This term is then multiplied with an estimate of the advantage function $\hat{A}_t$. We then clip this loss function, to only allow the ratio of the policy we are updating to the old policy, to change with ϵ. This allows only small changes in policy and replaces the KL-divergence constraint in the TRPO algorithm.

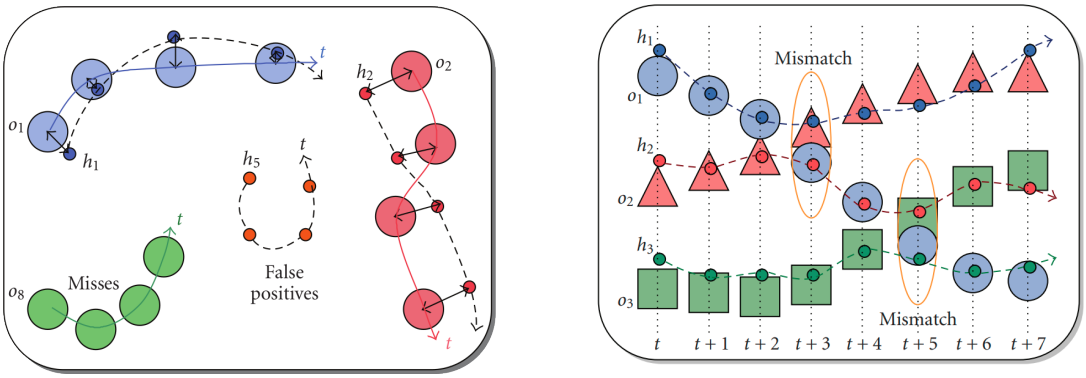
The estimate of the advantage is found by using a value network that is optimised jointly with the policy. The advantage can then be estimated as

$$\hat{A}_t = -V(s_t) + r_t + \gamma r_{t+1} + ... + \gamma^{T-t+1} r_{T-t} V(s_T).$$

This advantage tells us how good the actions we are taking are [37]. In multi-agent reinforcement learning (MARL) with policy sharing, multiple agents are present and take actions in the environment using the same policy. The PPO algorithm then works by letting agents take step in copies of the environment for a fixed number of time steps. The advantages are calculated and the policy is updated with mini-batches of the collected experiences. These agents share the same policy but act in copies of the same environment to decorrelate the updates of the policy. This works as the experiences encountered by a single agent are highly correlated. So by letting agents act in copies of the environment the collected experiences and thus gradients will be more diverse. This procedure is then repeated [37] and the agents collect experiences in a rollout and the policy is updated via the PPO loss function.

3.6 Performance Measures

To train the model, evaluate the tracking results and compare the results between algorithms we need an evaluation measure. Two of the most widely used performance metrics in the field of Multi-Object tracking are the *Multiple Object Tracking Accuracy* (MOTA) and the *Multiple Object Tracking Precision* (MOTP), proposed by Bernardin and Stiefelhagen in the 2008 article [1]. Some common errors that can occur in tracking can be visualised by Figure 10.



(a) Illustrations of false positives and false negatives

(b) Illustrations of identity switches.

Figure 10: Examples of tracking errors. Small circles are tracking results and large shapes are ground truth objects.

The MOTA score is a combined metric that takes into account three different types of tracking errors: false positives, misses, and identity switches. We call the number of these errors in each time frame t for fp_t (false positives), m_t (misses) and mme_t (identity mismatches). This sum is then normalised with the total number of ground truth objects in all frames ($\sum_t g_t$) and this value is subtracted from 1 in order to get the $MOTA$ metric.

$$MOTA = 1 - \frac{\sum_t (fp_t + m_t + mme_t)}{\sum_t g_t} \tag{33}$$

A perfect tracking would have no errors and a $MOTA$ of 1. Since there is no limit on the amount of errors that can occur, the metric is not bounded from below and $MOTA \in (-\infty, 1]$. In order to calculate the number of errors and types of errors, we need to obtain a correspondence between the estimated tracks that our tracking algorithm produces and the ground truth tracks for each time step t. This is done using the Hungarian algorithm. In order for an estimated track to be associated to a ground truth track g at frame t, we require that the distance between their states at this frame is less than some threshold.

In the intra-camera tracking we require the IOU similarity between the ground truth object's bounding box and the estimated track's bounding box at frame t to be at least 0.5 for the assignment to be considered a match. For the global estimated tracks, a threshold of 0.5 meters is chosen. A ground truth track that do not get an associated track to it at frame t, is considered a miss at this frame. Similarly an estimated track that do not get an associated ground truth track to it at frame t, is considered as a false positive at this frame. Lastly, an estimated track that gets a different ground truth track associated to it in frame t then in frame $t-1$, is considered an identity mismatch. However, if two estimated tracks swap ground truth tracks in a frame t this is only counted as one identity mismatch as can be

seen in Figure 10 b). Note that during tracking, we allow a smaller IOU overlap (0.3 minimum) in the camera for a detection to be associated to a track. Furthermore, in the global tracking we allow a max distance for the association to be 6 meters both for the baseline and the RL-method.

Another common metric is the $MOTP$ which describes how close the estimated track positions are to its associated ground truth positions. For each time step t the sum of the distances d between tracking estimates and the associated ground truth positions are divided by the total number of matches m.

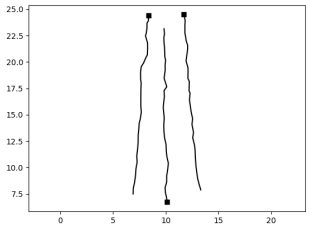
$$MOTP = \frac{\sum_t d_t}{\sum_t m_t} \tag{34}$$

Note that $MOTP \in [0, \infty)$. The Python package `motmetrics` [20] was used to calculate the metrics.

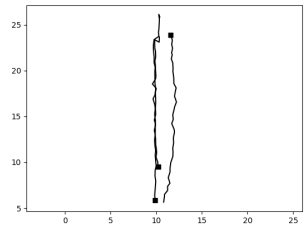
4 Methodology

4.1 Data Sets and Calibrations

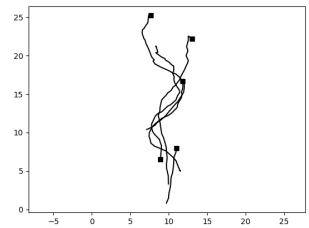
To train and evaluate our models, we generate six different data sets. Firstly, a real-world test scene with two cameras is set up and calibrated. Recordings from these cameras are manually annotated. The real-world scene is used to model one of the simulated scenes. We implement a simulator environment using the open source 3D-engine Blender. Using this simulator, we generate two scenes with different camera configurations. From these scenes we generate a number of data sets with varying number of pedestrians with different walking patterns.



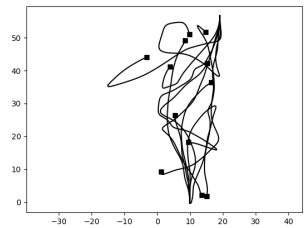
(a) `realworld1`.



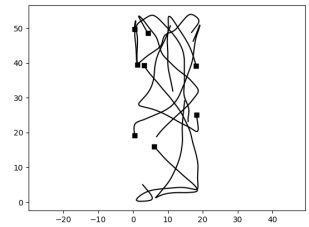
(b) `realworld2`.



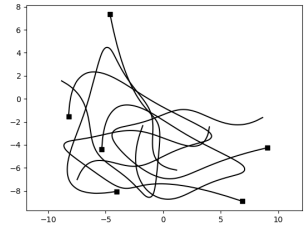
(c) `realworld3`.



(d) First 1000 frames of the simulated data set `trainingdata`.



(e) `testdata`.



(f) `simdata`.

Figure 11: Top-down view of the ground truth pedestrian positions. Squares represent the last position for each track in the interval. The data sets (a)-(c) were manually annotated and the ground truth tracks for (d)-(f) were extracted from the simulator. Scale in meters.

Figure 11 shows a top-down view of the ground truth positions of the pedestrians for all data sets used in this thesis.

4.1.1 Simulated Data

For the training of the reinforcement learning agents, a large data set of annotated video sequences is desirable. The process of annotating long video sequences with many persons can, unfortunately, be very cumbersome and time-consuming [7]. Thus, an automated and quick method for the generation of long, annotated video sequences of pedestrians is preferable. We chose Blender for this automation process due to its open-source nature and simple API for Python scripting.

We want the model trained on simulated data to be able to perform well on real-world data. The simulated data set should therefore closely resemble the real-world. Our pipeline does not use any visual features after the detection so we mainly want the *detections* on the simulated data to behave similarly to the detection from real-world data. For this purpose, a realistic 3D model of a person is created. This is done using photogrammetry. Approximately 30 images of a person are taken at different angles and put into a the *AliceVision Meshroom* photogrammetry pipeline [15]. In this pipeline a point cloud is generated from the images, this point cloud is then meshified and lastly texturised. The resulting 3D mesh is shown in Figure 12. This model is imported into Blender. Lastly, a walk cycle animation is created for the model.

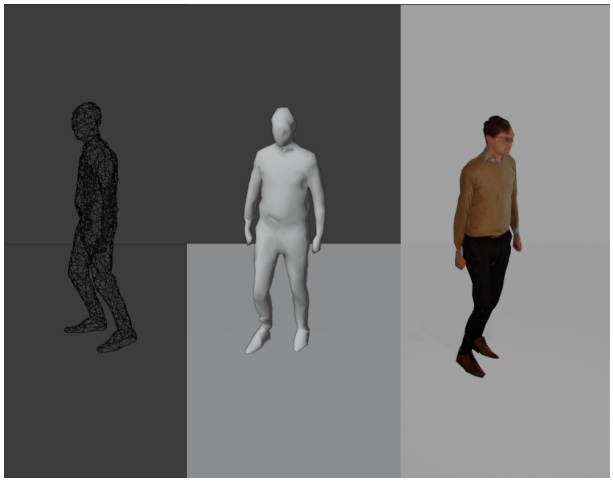


Figure 12: The photogrammetrised model used in the simulator in different render modes. From left to right the model is rendered with a wireframe, with simple diffuse shading, and with textured shading.

Two model scenes are created in the Blender environment with cameras, lighting, textured buildings, walls and ground. In these scenes, a number of person meshes are added. A script for generating random walking paths in the simulator environment is written to simulate pedestrians. From a set of triangles, we define a domain

$\mathcal{D} \subset \mathbb{R}^2$ in the simulator world as the union of these triangles. The domain $\mathcal{D}$ is used to determine the x, y positions in the scene where a person could be positioned. By using triangles to model the domain, we can achieve more complex domains than what would be possible with, for example, rectangles. Within this domain, we guarantee that a person is not positioned inside a wall, too far away from the camera's field of view etc.

Each person's path is then defined by a discrete set of ordered points $\mathbf{x}_t = (x, y)_t \in \mathcal{D}$ inside this domain. These points are generated by first spawning an initial point, $\mathbf{x}_0 \in \mathcal{D}$, together with an initial direction, $\mathbf{d}_0 \in \mathbb{R}^2$ of unit length. Then, for N steps, a hypothetical step is taken in the direction $\mathbf{d} \in \mathbb{R}^2$. We make sure that the hypothetical step remains inside one of the triangles by transforming the point $\mathbf{x}_t + \mathbf{d}$ into each triangles' barycentric coordinates. If the point $\mathbf{x}_t + \mathbf{d}$ remains inside the predefined domain $\mathcal{D}$, this point is added to the list of ordered points and used as a starting point for the next step.

If a hypothetical point is placed outside of the defined scene, a new direction is randomly picked and if the new point remains inside the domain, that point is added instead. Furthermore, after each step, the direction vector, $\mathbf{d}$, is rotated by a small random angle in radians, $\theta \sim \mathcal{N}(0, \sigma^2)$ with $\sigma = 0.1$, drawn from a Gaussian distribution. The length of $\mathbf{d}$ is also resampled at each step so that $|\mathbf{d}| \sim \mathcal{N}(1, \sigma^2)$, with $\sigma = 0.1$. From this set of points, a Bézier curve is interpolated. Each person in the scene is then animated to follow their respective curve. The resulting paths we use in the data sets in this thesis are visualised in Figure 11. We use Blender's built-in real-time renderer Eevee, to render video files from all cameras in the scene and export them in `mp4` format. A single frame from the data set `simdata` is shown in Figure 13

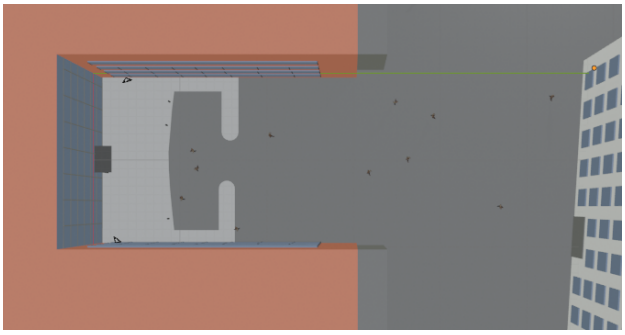
We also write another script to export csv-files containing ground truth information and scene description. We need this data to train our model and to evaluate it using the evaluation metrics. For each frame t in the scene, the 3D positions of each person that is visible in at least one camera is exported. A person is determined to be visible in a camera if a ray cast from the person's centre can reach the camera centre without intersecting with another object. Hence, if a person is occluded in all cameras, the person's position is not exported to the ground truth position file. We also need to obtain the ground truth bounding boxes in each camera's image space. This is done by projecting each vertex from the photogrammetrised mesh into the pixel space and determining the minimum and maximum x- and y-coordinate of those projections to get bounding boxes on the form $[x_{min}, x_{max}, y_{min}, y_{max}]$.

Figure 13: One frame from Camera 2 in the simulated dataset `simdata`.

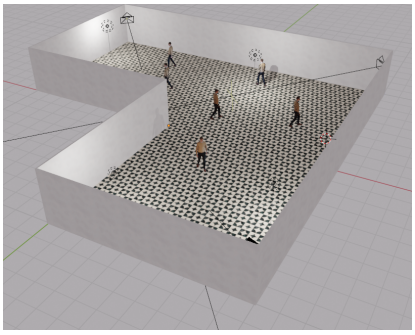
In summary, we use Blender to create a 3D scene with walls, ground, cameras, lighting, textures, etc. In this scene, our simulation script generates random walking pedestrians and exports the following:

- `mp4` video files from all camera perspectives
- A `csv` file containing the cameras' intrinsic and extrinsic parameters.
- A `csv` file containing ground truth bounding boxes of visible persons in the images space for all cameras.
- A `csv` file containing ground truth world coordinate positions for persons visible in any camera.
- A `csv` file containing the scene description in the form of floor triangles.

We generate three data sets in the simulator environment. The two datasets `training-data` and `testdata` are both modeled after the real-world test scene described below. The `trainingdata` set is 5000 frames long, has a frame rate of 24 fps and consists of three cameras and 12 persons. It would be possible to generate for example 1000 different videos from the scene but due to time and storage limitation we settled with one long 5000 frame training data set. The `testdata` set is 1000 frames long, has a frame rate of 24 fps, and consists of three cameras and eight persons. The data set `simdata` has three cameras, six persons and is set up with a different scene and camera configuration compared to the first two data sets. This is used to evaluate how dependent the global RL-agents are on the scene they were trained on. In Figure 14 follows a top-down view of the `trainingdata`/`testdata` scene (a) and a view of the simdata scene (b).



(a) Top-down view of simulated scene used for `trainingdata` and `testdata`.



(b) View of simulated scene used for `simdata`.

Figure 14: Blender scenes used in thesis.

4.1.2 Real-World Data

We also evaluate our models on video recordings from real cameras. We record these videos from the building we model the `trainingdata`/`testdata` scenes from. In this scene two cameras are used, placed on opposite sides of a passageway. A top-down view of the test scene is shown in Figure 15. We record three test sets using this set up. Note that the third camera in Figure 15 is only present in the Blender simulations due to hardware limitations during the gathering of the real world data set. The lack of the third camera does not affect the SORT algorithm, since this method is independent on the number of cameras. For our global RL-agent, this means that the input from the third detector will always be zero on these scenes. Camera one and camera two are placed at the same positions in the recorded scenes as in the simulated video.

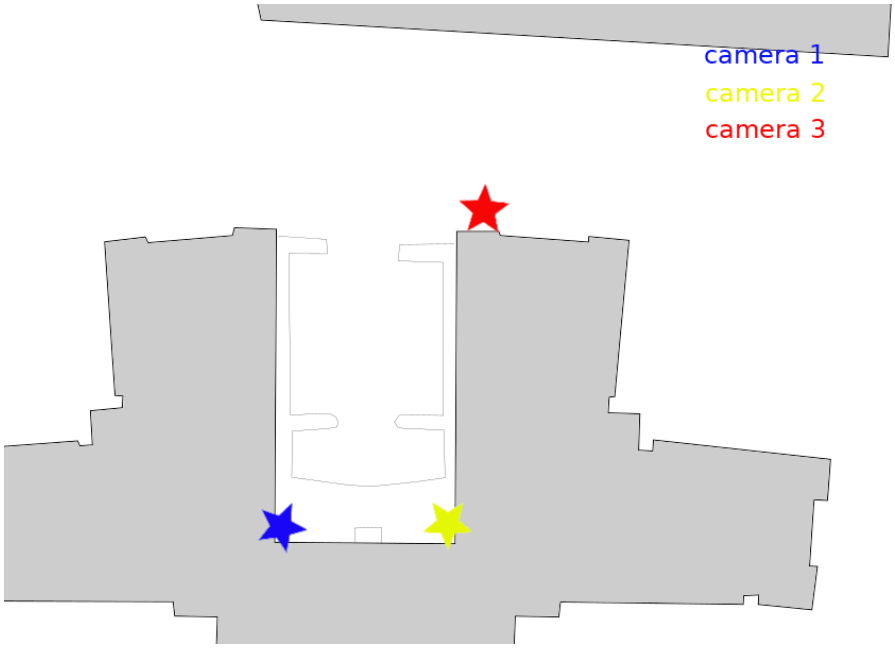
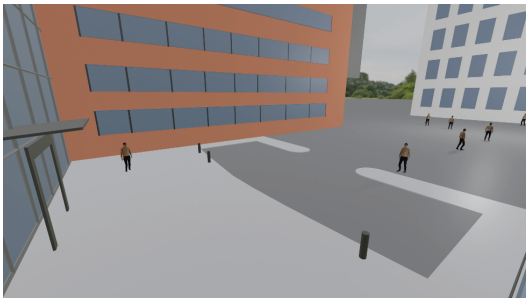


Figure 15: Test scene used both for the data sets `trainingdata`, `testdata` and the real-world recordings in data sets `realworld1-3`. For the simulated data, all cameras are used, and for the recorded data Camera 1 and Camera 2 are used.

We introduce a global coordinate system with the origin placed in the bottom left corner of the building in Figure 15 close to camera one. The x- and y-axis are aligned along the walls of the building, which we assume to be at a 90° angle. In this coordinate system, we assume that the ground lies in a perfect plane with coordinates $(x, y, 0)$. Using these assumptions, we measure a number of interesting real world coordinates, $\mathbf{x}_w^i \in \mathbb{R}^3$ visible in the cameras with a laser rangefinder. The corresponding image space pixel coordinates, $\mathbf{x}_p^i \in \mathbb{R}^2$, are extracted from the video to obtain a number of point correspondences $(\mathbf{x}_w^i, \mathbf{x}_p^i)$. In Camera 1, we extract 12 point correspondences, and in Camera 2, we extract 13 point correspondences. The calibration process could be automated using, for example, *Scale-invariant feature transform* (SIFT) [27] for feature extraction and the *Random sample consensus* (RANSAC) [12] algorithms for calibration. This however falls outside the scope of this thesis.

Since the recordings do not start at exactly the same point in time in all cameras, the video sequences need to be synchronised. We do this manually by removing a number of initial frames for the recording with the earliest starting time. Then we remove the frames at the end of the longest video to make the video sequences have the same length. Since both cameras have the same frame rate, the frames of the video sequences correspond to the same time.

(a) Frame from `realworld2`. In this scene two persons are walking on a line towards the camera and another person is walking away from the camera.



(b) Frame from `trainingdata`.

Figure 16: Single frame from Camera 2 in the data seta `realworld2` and `testdata`.

After having synchronised and calibrated the real world recordings we manually annotate them to obtain ground truth pedestrian positions in world coordinates. For every 10th frame, the position between the feet of every person visible in the image space are manually selected. Those positions are then projected onto the ground plane using the calibrated camera matrices. If a person is visible in more than one camera, the projected points are fused by triangulation of those points. Finally, the ground truth is linearly interpolated to obtain ground truth positions for all frames. Figures with the resulting annotations are shown in Figure 11.

4.2 Introduction to Pipelines

We will now introduce the two pipelines used in this thesis. First, we introduce our baseline model, based on the *Simple Online and Realtime Tracking* (SORT) algorithm [2]. In order to describe the baseline we go over the tracking algorithm used in each camera and then the extension used to track pedestrians in the 3D world. Secondly, we introduce the Reinforcement Learning (RL) method. We describe the RL method proposed by Rosello and Kochenderfer in [34] and then go over our proposed multi-camera extension of this method. Before we introduce these pipelines we will first give a few important definitions that we will use throughout the presentation of the methods.

Firstly we reiterate that we define a track as a sequence of states with a unique id that belong to a single object. The state can vary depending on what is being tracked and what kind of track we are dealing with. In this thesis we will make a distinction between what we will refer to as *local tracks* $\mathcal{L}_k$ and what we will refer to as *global tracks* $\mathcal{G}$. A local track $l_i^k \in \mathcal{L}_k$ with track id i is a track that belongs to a camera c_k in the camera network $\mathcal{C}$. The states of this track will be in pixel coordinates. The tracked states of the local tracks are the centre positions (x_p, y_p), areas s, aspect ratios r and corresponding velocities $(\dot{x}_p, \dot{y}_p, \dot{s})$ of the bounding boxes of each object. A global track g is also a sequence of states of an object, but it does not belong to any camera. This means that the local tracks that have formed each

global track can originate from any combination of cameras. The states of the global tracks are 3D world coordinates. Specifically, we track the centre position of each pedestrian (x_w, y_w) and its velocity $(\dot{x}_w, \dot{y}_w)$ in the ground plane. For our data sets we define the ground plane as $z = 0$ in the world coordinate system.

We also make the distinction between active and killed tracks. Active tracks are tracks that are still being updated and can receive new detections. A killed track is a track that can no longer be updated and can not get new detections. The active and killed tracks can be both local and global tracks. Furthermore, we will use local tracker and local method interchangeably when we talk about the multi-object tracking algorithm that is run in each camera. In a similar manner we will use global tracker or global method when discussing the tracking that is done in the 3D world. With these definitions we will move on to describe the baseline in more detail and then the RL method.

4.3 Multi-Camera SORT

The first approach we use to solve the real-time multi-camera multi-object tracking problem is the Simple Online and Realtime Tracking (SORT) algorithm. This is applied both for tracking in each camera's pixel coordinates and for tracking in world coordinates. When tracking is performed in world coordinates, we use a version of the SORT algorithm that is adapted for a multi-camera setting. We start by describing the tracking taking place in each camera and then describe how we do the tracking in global coordinates.

4.3.1 Local SORT Tracker

We use the YOLO-v4 [3] network to detect the bounding boxes of the pedestrians and track them using the SORT algorithm. In each time frame t the detector network outputs a number of bounding boxes $\mathbf{d}$ that we will refer to as *detections*. These bounding boxes are all compared to the active local tracks' predictions in the current frame. The predictions are formed by applying the motion model to the most recently tracked state. This is done with a Kalman filter and is described further down. The IOU is calculated between all predictions and detections and a cost matrix $\mathbf{C}$ is formed from these. The Hungarian algorithm, as described in Section 3.4, takes the computed cost matrix as input and solves the assignment problem. In each frame, the SORT algorithm proceeds as in Figure 17 using these associations.

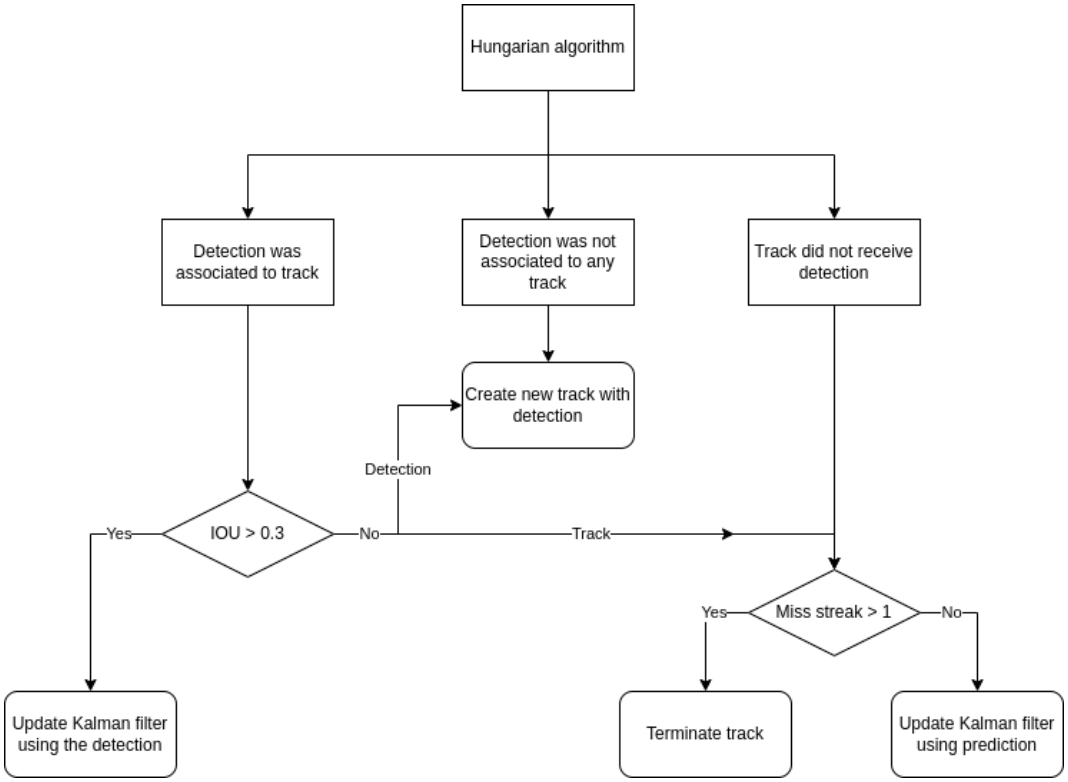


Figure 17: Flowchart describing the decisions made in the SORT algorithm after detections have been associated to tracks using the Hungarian algorithm.

The associations that are made are compared to the minimum association distance $d_{\max}$ (0.3 IOU overlap). If a local track's prediction and its associated detection overlap more than this minimum association distance, the detection is used for a Kalman filter update. Otherwise a new track is initialised with the detection and with a new track id. Tracks that do not receive a detection for more than one frame are terminated.

We will now describe the Kalman filter that is used in the SORT algorithm in the camera's image space [2]. This Kalman filter is used to correct noisy detections and as a motion model to be able to perform predictions. In the image space the SORT algorithm makes use of a constant velocity pedestrian motion model which has also been shown to outperform many of the state-of-the-art pedestrian trajectory forecasting models [39]. The states that are tracked in the Kalman filter are the centre position, area and height-to-width ratio (aspect ratio) of the bounding boxes together with their respective velocities (except for the aspect ratio as it is assumed to be constant). The transition of the state space for each locally detected target in

the image is thus given by

$$\mathbf{x}_k = \begin{pmatrix} x_k \\ y_k \\ s_k \\ r_k \\ \dot{x}_k \\ \dot{y}_k \\ \dot{s}_k \end{pmatrix} = \begin{pmatrix} x_{k-1} + \Delta T\ \dot{x}_{k-1} \\ y_{k-1} + \Delta T\ \dot{y}_{k-1} \\ s_{k-1} + \Delta T\ \dot{s}_{k-1} \\ r_{k-1} \\ \dot{x}_{k-1} \\ \dot{y}_{k-1} \\ \dot{s}_{k-1} \end{pmatrix} + \mathbf{q}_k = \begin{pmatrix} 1 & 0 & 0 & 0 & \Delta T & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & \Delta T & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & \Delta T \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \mathbf{x}_{k-1} + \mathbf{q}_k, \tag{35}$$

where ΔT is the time step length set to 1 as we are tracking from frame to frame, x_k and y_k are the coordinates of the centre of the bounding box, s_k is the area and r_k the aspect ratio of the bounding box. The remaining states are their respective velocities and $\mathbf{q}_k$ is the Gaussian noise as described in the theory Section 3.3. The states modelled with the constant velocity model (all except the aspect ratio) should have 0 acceleration. Therefore any acceleration in these states is modelled as noise. In a similar fashion r_k is assumed to be constant and so its velocity should be 0. Thus any velocity in the r_k state is modelled as noise and the final model noise vector $\mathbf{q}_k$ becomes

$$\mathbf{q}_k = \begin{pmatrix} \frac{\Delta T^2}{2} a_{xy} \\ \frac{\Delta T^2}{2} a_{xy} \\ \frac{\Delta T^2}{2} a_s \\ \Delta T v_r \\ \Delta T a_{xy} \\ \Delta T a_{xy} \\ \Delta T a_s \end{pmatrix}, \tag{36}$$

where it is assumed that all the accelerations a_{xy}, a_s and the velocity v_r are Gaussian with zero mean. We also assume that each state is independent of the others, with the exception of a state and its velocity state pair, i.e, $(x_k, \dot{x}_k)$, $(y_k, \dot{y}_k)$ and $(s_k, \dot{s}_k)$. The ΔT terms in Equation (36) arise from the interpretation of the noise as the accelerations and velocity, respectively.

Furthermore, the x and y accelerations are assumed to have the same variance a_{xy}. Since we do the tracking on a frame-to-frame basis, the time interval $\Delta T = 1$. With this in mind we get that

$$\mathbf{Q}_k = \begin{pmatrix} \frac{a_{xy}^2}{4} & 0 & 0 & 0 & \frac{a_{xy}^2}{2} & 0 & 0 \\ 0 & \frac{a_{xy}^2}{4} & 0 & 0 & 0 & \frac{a_{xy}^2}{2} & 0 \\ 0 & 0 & \frac{a_s^2}{4} & 0 & 0 & 0 & \frac{a_s^2}{2} \\ 0 & 0 & 0 & v_r^2 & 0 & 0 & 0 \\ \frac{a_{xy}^2}{2} & 0 & 0 & 0 & a_{xy}^2 & 0 & 0 \\ 0 & \frac{a_{xy}^2}{2} & 0 & 0 & 0 & a_{xy}^2 & 0 \\ 0 & 0 & \frac{a_s^2}{2} & 0 & 0 & 0 & a_s^2 \end{pmatrix}, \tag{37}$$

where a_s and v_r is the variance of the area and the aspect ratio modelled as acceleration and velocity respectively. The measurement model $\mathbf{y}_k$ has the centre of the bounding box, its area and its aspect ratio as measurement states. So

$$\mathbf{y}_k = \begin{pmatrix} x_k \\ y_k \\ s_k \\ r_k \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix} \mathbf{x}_k + \mathbf{r}_k, \tag{38}$$

with $\mathbf{r}_k$ being the 4x1 vector with the Gaussian measurement noise where each variable being assumed to be independent of the others. Thus the covariance matrix of $\mathbf{r}_k$ is a 4x4 diagonal matrix. Clearly the measurement states should not be independent but the study of the true correlations in the measurement noise is beyond the scope of this thesis and is left as future work. In the original SORT paper the authors assume independent measurement noise and hidden state noise ($\mathbf{Q}_k$ and $\mathbf{R}_k$ to be diagonal matrices). We only make this assumption for $\mathbf{R}_k$. The full details of the Kalman parameters can be found in the appendix, Section A.5. Next, we move on to describing the global baseline tracker.

4.3.2 Global Baseline Tracker

In each time step (frame) we take the current positions of the active local tracks and map them into the global coordinate system to update the global tracks. In order to do the tracking globally we use the an extended version of the SORT algorithm in the world coordinates (that is, using the Hungarian algorithm for data association and the Kalman filter for state estimation). We again use a constant velocity pedestrian motion model but this time in a global plane as the local tracks have been projected onto the ground plane in the 3D world. The mapping from image space to the 3D world can be done either by inverse camera projections onto the ground plane as described in Section 3.1.4 or by triangulation described in Section 3.1.5 if a global track has several local tracks assigned to it.

In the multi-camera SORT (the extended SORT algorithm) we start each frame by performing data association between the local tracks originating from each camera and the active global tracks. Since two local tracks from the same camera generally originate from two different persons, we associate the local tracks with the set of predictions from the global tracks, one camera at a time. Hence, no association is performed between local tracks originating from the same camera. In order to associate the local tracks with the existing global tracks a distance between a global track and a local track is defined. The distance we use, $\boldsymbol{C}_k(\hat{\mathbf{x}}_i,\ \mathbf{p}_j)$, is the Euclidean distance between the prediction $\hat{\mathbf{x}}_i \in \mathbb{R}^2$ from the global track and the projected state $\mathbf{p}_j \in \mathbb{R}^2$ from the local track l_i^k with track id i in camera c_k. This distance is defined as:

$$\mathbf{C}_k(\hat{\mathbf{x}}_i, \mathbf{p}_j) = ||\hat{\mathbf{x}}_i - \mathbf{p}_j||_2, \tag{39}$$

where $\mathbf{p}_j$ is the projection of the midpoint of the bottom side of the bounding box

state $(\frac{x_{min}+x_{max}}{2}, y_{max})$ [note that the origin of the pixel coordinates is in the left top corner of the image and the y-axis is directed downwards, hence y_{max} is used for the bottom side of the bounding box]. The distances from all local tracks in each camera c_k are stored in a cost matrix $\boldsymbol{C}_k$. We solve the association problem for each camera c_k to the global tracks with the Hungarian algorithm. This means that each global track can get either zero, one or multiple local tracks associated to it from the different cameras.

After the associations are done we need to fuse the associated projections from each local track. If a global track gets multiple local tracks associated to it from different cameras, we use triangulation as mentioned before to fuse the local track states and map them into the 3D world. The triangulation is done with the most recent state of each local track. Otherwise, if the global track only got one local track associated to it, we use the projected state from the local track.

We will refer to the fused and projected states from the local tracks as a global detection d_g. The global tracks that received a global detection d_g during the association use these to perform a Kalman update. The global tracks that did not get a global detection, but only have missed one consecutive global detection d_g, performs the Kalman update with its prediction instead. The track thus get one attempt to use the prediction and potentially get another global detection d_g in the next frame. If however, the track missed more than one global detection in a row we terminate the track. The psuedo-code for the global tracker is formulated below in Algorithm 3. In this pseudo-code we assume that each camera has performed a step in the corresponding time frame.

Algorithm 3 SORT Tracking in world coordinates

```
for each f ∈ Frames do
    for each c_k ∈ C do
        Associate the set of local tracks L_k from camera c_k to the set of global
        tracks G with the Hungarian algorithm.
    end for
    for each g ∈ G do
        if g got local tracks associated to it then
            Fuse the associated tracks to a global detection.
            Perform the Kalman update of global track g with the global detection.
        else if g missed one global detection then
            Peform the Kalman update with prediction of global track g.
        else
            Terminate global track g.
        end if
    end for
    for each d_g ∈ D_g do
        Initialise a new global track with d_g and new id.
    end for
end for
```

In the psuedo code in Algorithm 3 $\mathcal{G}$ is the set of active global tracks, $\mathcal{L}_k$ is the set of local tracks from camera c_k, g is a global track, $\mathcal{D}_g$ the set of global detections not used by any existing global track and l_i^k is a local track in $\mathcal{L}_k$. The Kalman filter parameters can be found in the appendix, Section A.5. In the next section we will describe a method used by Rosello and Kochenderfer [34] to more dynamically decide when to terminate tracks, when to use detections versus predictions among other things in image space. We will also describe how our method, inspired by the article [34], do this in the 3D world coordinates in a multi-camera setting.

4.4 Multi-Agent Reinforcement Learning for Multi-Object Tracking

In the following section, we describe the Reinforcement Learning (RL) based method proposed to solve the MCMOT tracking problem. In this method each track has an associated RL agent that manages the track. New detections are associated to local tracks with the Hungarian algorithm and the positions of the tracks are updated using a Kalman filter in the same way as in the SORT algorithm. However, with this method, no predefined thresholds are used to determine when a track should be started, terminated, made hidden, or when a track should update its state using the Kalman predictions. Instead all of these decisions are made by the reinforcement learning agent associated to the track. Based on the RL approach in [34], we extend the method to a multi-camera setting where global context is taken into account and where the data fusion is also managed by the RL agents. Before we present the global RL agent we will first introduce the local RL agent in each camera.

4.4.1 Local RL Model

In each camera c_k, we associate a reinforcement learning agent to each local track l_i^k. The tracks are managed by individual agents with the same observation space and action space that we will describe further down for a single agent. A track is always in one of three different states: visible, hidden, or inactive. A *hidden* track is a track that is not actively tracked but that keeps updating its state using the Kalman predictions. This state is intended for situations with short to medium long occlusions. Tracks are in the *inactive* state before being initialised and after having been terminated. All tracks that are not hidden nor inactive are called *visible* tracks. The visible tracks are the active tracks in the scene that are used for evaluation of the tracking algorithm. This means that to calculate the evaluation metrics like MOTA and MOTP, as described in Section 3.6, we only compare the ground truth tracks at frame t with the tracks that are visible at frame t. Hidden tracks can also be made visible while keeping the same track-id, if it should receive new detections.

Like the baseline, for each frame the detections are associated to the visible and hidden tracks with the Hungarian algorithm. New inactive agent/track pairs are created dynamically so that all detections can be associated to a track. After the detections have been associated to the tracks, a local observation vector, $\mathbf{o}_l$, is con-

structed for each agent. This vector describes the local track and its associated detection. For the local RL agent, $\mathbf{o}_l \in \mathbb{R}^{18}$. The observation vector $\mathbf{o}_l$ can be divided into the states of the track $\mathbf{x}_k$ and the associated detection $\mathbf{d}$. An agent observes its track's state and the associated detection's features. The track's states consist of the track's predicted Kalman state $\hat{\mathbf{x}}_t \in \mathbb{R}^7$, the one-hot encoded track's current activation state (visible/hidden/inactive), the streak of detections and the streak of misses. The detection features are the associated detection $\mathbf{d}$, the association cost and the confidence c of the YOLO detection $\mathbf{d}$. The Kalman prediction and the detection are described in Section 4.3.1. If no detection is associated to the track, the detection, $\mathbf{d}$, is set to all zeros. Following [34] the streak of detections and misses are normalised using a sigmoid function to avoid numerical instability. Each track's observation vector $\mathbf{o}_l$ is then used as an input to the neural network and the agents take actions by sampling from a probability distribution over the actions $a \sim \pi_\Theta(a|o_l)$ where π_Θ is the policy of the agent modelled by a neural network.

The possible actions an agent can make are shown in Table 1. By using action a_0, the agent terminates and resets a track and puts it in an inactive state. This state is intended to be used when a person leaves the scene. Action a_1 restarts the track by resetting the Kalman states and initiating the Kalman filter with its current detection. Action a_2 takes the detection associated to the track by the Hungarian algorithm and updates its Kalman filter using this bounding box. Action a_3 ignores possible detections and uses the Kalman filter's prediction to update the track's state, intended for situations with short occlusions or missed detections. Action a_4 is similar to action a_3 but also puts the track into a hidden state. This action is intended for situations with medium long occlusions.

Table 1: Actions defined for the local tracking agent.

Action	**Description**	**State after action**
a_0	Terminate a track	Inactive
a_1	Restart a track using detection	Visible
a_2	Uses a detection to update track	Visible
a_3	Uses Kalman prediction to update track	Visible
a_4	Uses Kalman prediction to update track and puts track in hidden state	Hidden

We now move on to describe the model for the global agent before we explain the reward functions, network architectures and the training setup used.

4.4.2 Global RL Model

The global RL model we propose in this thesis is similar to the local RL model. All global tracks get an RL agent associated to it. For each frame, the active and hidden local track's current states are associated to the global tracks' predictions in the same way as in the SORT algorithm using the Hungarian algorithm. In this way a list of local tracks are associated to each global track. Then an observation vector, $\mathbf{o}_g \in \mathbb{R}^{27}$, is constructed for the global track. In a similar fashion to the local RL

agent we can divide the observation vector for the global RL agent into two parts. The observation vector will consist of the states of the RL agent's associated global track and features from the possible local tracks from the three closest cameras. The global track's states again consist of the track's Kalman prediction $\hat{\mathbf{x}} \in \mathbb{R}^4$ consisting of the predicted position in the ground plane x, y and the predicted velocities $\dot{x}, \dot{y}$, the one-hot encoded activation states of the track, the streak of detections and the streak of misses.

The features from the possible local tracks l_k from each camera c_k are the associated local track's state projected onto the ground plane $\mathbf{x_p} = (x_d, y_d)$, the association cost between $\mathbf{x}_p$ and the global track, the distance between $\mathbf{x}_p$ and the camera c_k it originated from, a one-hot encoded element of whether the global agent has got an associated local track l_i^k and the YOLO confidence of the detection track l_i^k received to estimate its state. If the local track does not get a detection from a camera at this frame the confidence is simply put to zero. Furthermore, if no local track l_i^k from camera c_k is associated to a global agent, all the detection features are put to zero. The streak of detections and streak of missed detections are normalised by a sigmoid as in the single camera case. Inspired by the mono-camera approach, the same actions are included in the Global RL agent, but instead of a single action for using the detection, all combinations of the 3 possible associated local tracks l_i^k are used. In Table 2 below we summarise the possible actions.

Table 2: Actions defined for the global tracking agent. Action 2-8 are all possible combinations of detections.

Action	**Description**	**State after action**
a_0	Terminate a track	Inactive
a_1	Restart a track with new id	Visible
a_2	Uses local track from camera 1 to update Kalman filter.	Visible
a_3	Uses local track from camera 2 to update Kalman filter.	Visible
a_4	Uses local track from camera 3 to update Kalman filter.	Visible
a_5	Uses local track from cameras 1,2 to update Kalman filter.	Visible
a_6	Uses local track from cameras 1,3 to update Kalman filter.	Visible
a_7	Uses local track from cameras 2,3 to update Kalman filter.	Visible
a_8	Uses local track from cameras 1,2,3 to update Kalman filter.	Visible
a_9	Uses Kalman prediction to update track	Visible
a_{10}	Uses Kalman prediction to update track and puts track in hidden state	Hidden

In total there are 11 possible actions for the agent to take. When action a_0 is taken, the track is terminated. Action a_1 restarts a track with its associated detections (fuses them) and gives the track a new id if the agent is inactive, otherwise it keeps its id. Actions $a_2 - a_8$ represent all $2^3 - 1 = 7$ combinations of associated local tracks from the three closest cameras where at least one local track is used to update the global track. Action a_9 uses the Kalman filter prediction and keeps the track visible. Finally, action a_{10} puts the track in a hidden state and updates the Kalman filter using the filter's prediction $\hat{\mathbf{x}}$.

In this global setting, the agents should be able to take every action at all times without the program crashing. If an agent tries to fuse tracks from cameras where no track has been associated to the global track, we simply ignore that action and let the track stay in its current position. Just like in the single-camera case only visible tracks contribute to the global MOTA and MOTP measures. Now that the action and observation spaces of the agents, both local and global, have been explained we continue to describe the details of the training in Section 4.5 below.

4.5 Training Setup for RL

To train the reinforcement learning agents, we use the library stable-baselines3 [31] that contains an implementation of the PPO algorithm. It also allows the creation of custom policy and value-networks using PyTorch. Furthermore, to create the environment the agents use for training, we use the multi-agent libraries PettingZoo [41] and SuperSuit [42]. The PettingZoo environment allows one to easily create custom environments with a similar API as the popular gym library [6] from OpenAI. It also allows multiple agents in the environment to take actions in lock-step and receive rewards simultaneously. For the training of both the local and global agents we use fully connected neural networks with ReLu activations. All agents share the same policy network.

The training is done on multiple copies of the environment in parallel and using a GPU to parallelise the training. However, we found that since the networks are small we do not benefit much from training on the GPU. Instead what benefits us the most (in terms of training speed) is the number of cores on the CPU. The wrapper-library SuperSuit is used to allow the vectorisation of our parallel PettingZoo environments. SuperSuit also enables the sharing of the policy and value network across all copies of the environment and across all agents.

4.5.1 Local RL-Agent Training

For the local RL agent we train three different neural networks, one for each camera in the simulated Blender scene. The network size is the same as in [34], namely a fully connected network with three layers [128, 64, 32], where the elements in the vector are the number of perceptrons in each layer. We use this architecture for both the value network as well as the policy network. The training is done by selecting a random contiguous 200-300 frames long subset of the 5000 frames long training data set. This interval is resampled for each new episode. In each step, each agent takes an action based on its observation vector $\mathbf{o}_t$ and then get (potentially) new detections associated to it, giving rise to the observation vector $\mathbf{o}_{t+1}$ of the next timestep $t+1$. After the association is done with the Hungarian algorithm the new observation vector is formed and a reward r_t is calculated for each agent. In [34] all

agents received the same reward, namely

$$r_t = \frac{1}{T} - \frac{m_t + fp_t + mme_t}{\sum_{t'=1}^{T} g_{t'}}. \tag{40}$$

In this expression T is the length of the episode, m_t misses, fp_t the number of false positives and mme_t the number of mismatch errors in frame t. The $\sum_{t'}^{T} g_{t'}$ is the number of ground truth objects in the entire scenario. If we sum up all the rewards over all the time steps, the total rewards becomes the MOTA. However, due to difficulties making the RL agents converge using this reward function, we take another approach. We start by pre-training the policy network. This is done by constructing a composite reward function. The first part of it, r_1^t, gives each agent a 1 in reward if it takes the action that our baseline would take, and 0 if it didn't in frame t. That is

$$r_1^t = \begin{cases} 1, & \text{if action taken was same as baseline's action} \\ 0, & \text{otherwise.} \end{cases}$$

The second part of the reward is made to incentivise good actions from a MOTA perspective. This second part is for each frame t defined as

$$r_2^t = \begin{cases} 1, & \text{if visible and part of a true positve (match)} \\ 0, & \text{if visible and part of a false positive or id-switch} \\ r_3^t, & \text{otherwise,} \end{cases}$$

where r_3^t is the one step MOTA reward defined as

$$r_3^t = 1 - \frac{fp_t + mme_t + m_t}{g_t},$$

where have that fp_t, m_t and mme_t are the same as in Equation (40).The reward r_2^t gives active agents a big reward if they successfully become part of a match. This can be calculated as we have access to the ground truth of the tracking data during training. Furthermore, we give visible agents a reward of 0 if they take part in false positives or id-switches. The r_3^t is intended for the hidden and inactive agents. We can't say which hidden or inactive agents are a part of misses (false negatives) and so we simply give these agents the one step MOTA (r_3^t) as reward. This is done so that the reward falls in the same range in both pre-training and the main training afterwards, thus allowing for the learned value network to be useful after pre-training. Finally the pre-training reward is set as

$$r_{local-pre}^t = \frac{r_1^t + r_2^t}{2}. \tag{41}$$

We pre-train three policies (one for each camera) with the reward (41) before starting the final training. For the final training we instead drop the supervised part of the reward (r_1^t) that motivates the agent to act like the baseline. Instead we choose a

mixture between r_2^t and r_3^t as the final reward. The reason that we still do not train on pure MOTA like the authors of [34] is that we found that giving each agent a unique reward helps the training converge for the local RL agents. The final training reward is given as a mean of the r_3^t and r_2^t described above, i.e,

$$r_{local-final}^t = \frac{r_3^t + r_2^t}{2}. \tag{42}$$

The policy is pre-trained for approximately 20 million time steps and then trained for additionally 65 million steps. The results of the single-camera training can be found in the Section A.3 of the appendix. The details about the hyper-parameters including batch sizes, learning rates etc can also be found in the appendix, Section A.4.

4.5.2 Global Pre-Training and Training

We now move on to the training of the global RL agents. For these agents we use a slightly bigger policy network but let the value network remain the same size as in the local case, described in Section 4.5.1. The policy network that we use is a fully connected neural network consisting of four layers and ReLu activation. The number of nodes in each layer is [192, 128, 64, 32]. We also experienced some troubles getting this agent to converge using the suggested reward function in [34]. We therefore again pre-trained the agent for about 30 million steps to behave similarly to the baseline before training on the one step MOTA reward r_3^t.

The pre-training reward is the same as r_1^t for the global RL agent, except that the actions are taken to make decisions about global tracks in the 3D world ground plane as opposed to in the image space. After the pre-training is done we train with a pure one step MOTA reward r_3^t and do not include the agent specific rewards r_2^t as we did for the local agent. That is, we use only the r_3^t function as the reward function. The pre-training is done on 100 frames long subsets of the training data. The "post" training is done on 500 frames long subsets. The details of the hyper-parameters can again be found in the appendix Section A.4.

5 Results

In this section we present the results of our two methods, the threshold-based SORT method and our proposed reinforcement learning method. We evaluate these on video from a recorded scene, and on video from a Blender simulation modelled on the same scene. We start by presenting the calibration of the cameras for the real-world data. Then, we proceed by presenting the results of the detections on the simulated data sets. We then present the Multi-Object Tracking Accuracy (MOTA) and the Multi-Object Tracking Precision (MOTP) on all data sets and showcase the behaviour of the algorithms with a few examples. Lastly, we also present the training results from the reinforcement learning agents and some properties of their learned policies.

5.1 Camera Calibration

Figure 18 shows the world coordinate points $\mathbf{x}_w^i$ projected by the calibrated projection matrix $\mathbf{P}$ as red circles for camera two. In the same figure, the blue crosses are the corresponding image space points $\mathbf{x}_p^i$ manually selected from the camera's image space.

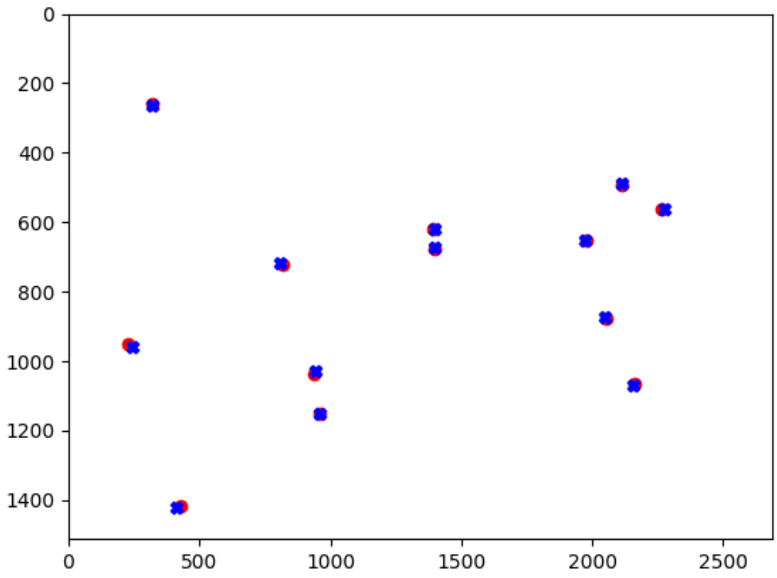


Figure 18: Calibration of the real-world data set in the pixel space of Camera 2. The blue crosses are the selected image points in pixel coordinates. The red circles are their corresponding world coordinates projected with the calibrated camera matrix, $\mathbf{P}$.

5.2 Detection Results

The basis of our pipeline are the detections from the cameras. The YOLO detector performed well on the simulated data as well as on the real-world data. Figure 19 shows a frame with almost perfect detection on the simulated data set `trainingset`. Note the missing detection on the right-hand side of the image. Additionally, Figure 20 shows the detections on a frame from the real-world data set `realworld1`.

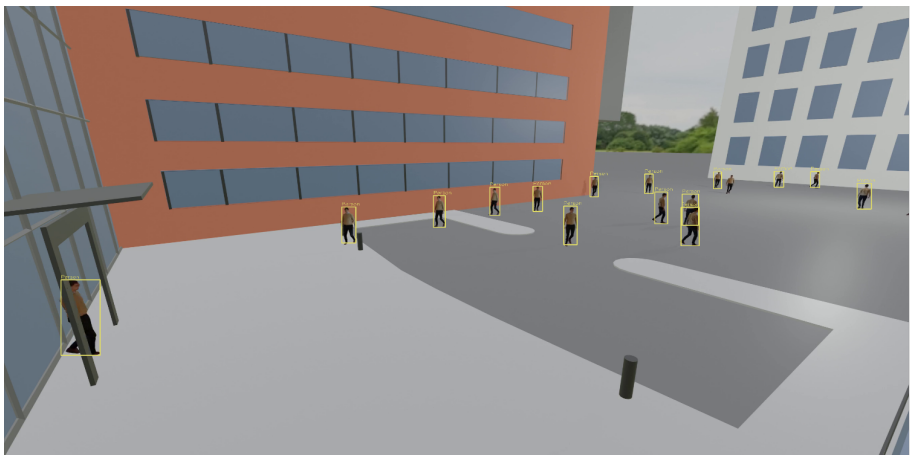


Figure 19: YOLO detection bounding boxes from `trainingset`. Each yellow box represents a detection.

Figure 20: YOLO detection bounding boxes from `realworld1` dataset. Each yellow box represents a detection.

Table 3 shows the result of the YOLO detector on the two simulated data sets. The detections and ground truth were associated using the Hungarian algorithm and a detection was considered a match if IOU > 0.5. Otherwise it would generate a false positive and a false negative. The detector accuracy was similar on all evaluated data sets.

Table 3: Table with results from the YOLO detection network on two data sets with ground truth annotations for the local bounding boxes. FP, is the number of false positives, FN is the number of false negatives, GT is the number of ground truth bounding boxes.

Dataset	FP↓	FN↓	Matches↑	GT	Precision↑	Recall↑
`trainingdata`, camera 1	773	2079	45600	47679	98.3 %	95.6 %
`trainingdata`, camera 2	767	490	23861	24351	96.9 %	98.0 %
`trainingdata`, camera 3	159	666	24825	25491	99.4 %	97.4 %
`testdata`, camera 1	107	210	6589	6799	98.4 %	96.9 %
`testdata`, camera 2	69	50	3025	3075	97.8 %	98.4 %
`testdata`, camera 3	25	95	4772	4867	99.5 %	98.0 %

5.3 Tracking Results

In the following section we show the main tracking evaluation results on all data sets. When evaluating the RL agents we used the deterministic policy given by the action with the highest probability. In training however, the actions were sampled from the stochastic policy. In Table 4 we can see that the RL model reached higher MOTA scores and lower MOTP scores (in meters) on all data sets except for the `simdata` set, which was generated from a different scene with a different camera configuration than the scene the RL agent was trained on.

Table 4: Table with global MOTA, MOTP scores and similar metrics for two simulated Blender scenes. FP is the number of false positives, FN is number of false negatives, IDs is the number of identity switches.

Dataset	Method	MOTA↑	MOTP↓	FP↓	FN↓	IDs↓	Precision↑	Recall↑
trainingdata	Baseline	75.2 %	0.190	7822	**5782**	**205**	86.5 %	**89.6 %**
	RL	**78.4 %**	**0.184**	**5997**	5862	210	**89.3 %**	89.5 %
testdata	Baseline	79.8 %	0.179	876	**720**	15	89.2 %	**91.0 %**
	RL	**81.6 %**	**0.176**	**718**	743	**12**	**91.0 %**	90.7 %
realworld1	Baseline	52.3 %	0.219	226	231	1	76.3 %	75.9 %
	RL	**56.2 %**	**0.217**	**210**	**210**	**0**	**78.1 %**	**78.1 %**
realworld2	Baseline	52.5 %	0.243	**261**	264	2	**76.4 %**	76.2 %
	RL	**52.6 %**	**0.241**	264	**262**	**0**	76.3 %	**76.4 %**
realworld3	Baseline	73.5 %	0.214	252	237	2	86.5%	87.2 %
	RL	**75.6 %**	0.214	**228**	**221**	2	**87.7 %**	**88.1 %**
simdata	Baseline	**87.5 %**	**0.130**	**240**	**201**	**9**	**94.4 %**	**86.5 %**
	RL	-10.7 %	0.290	1290	2662	27	41.9 %	25.9 %

For conciseness, we only present the global tracking results here, i.e, when tracking is performed in 3D world coordinates. The reader can find the results for the tracking in each camera in image space in the appendix, Section A.2.

5.3.1 Visualisation of Results

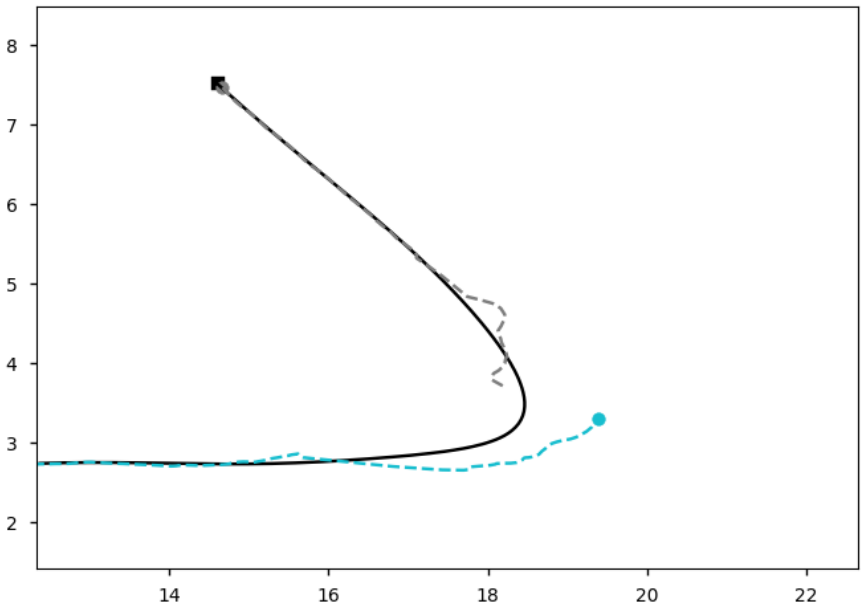
In Table 4, we can see a trend that the RL model outperforms the baseline in MOTA on almost all of the data sets. We now present some differences in tracking results in global coordinates, visualised as tracks from a top-down view.

In Figures 21–23, the ground truth tracks are represented by black solid lines and the tracking results are shown as the coloured dashed lines. Each colour represents a unique track-id. The round circle represents the end of the estimated tracks and the square represents the end of a ground truth track in the interval. Note that the baseline model and the RL model have the same max distance (6 m) for the association of projections to active tracks. Hence, the improvement in the tracking accuracy is only due to better track management (track birth, track termination, when to use predictions, etc.).

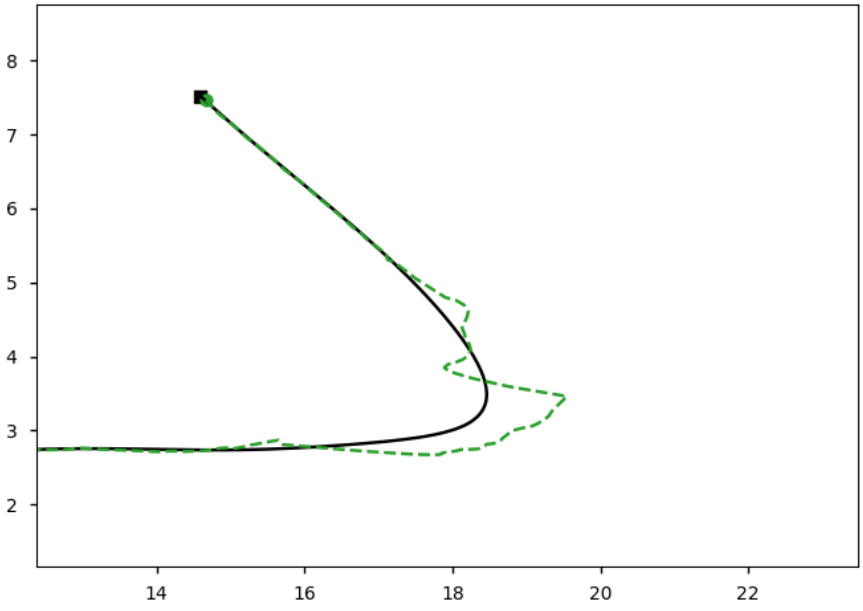
The baseline model in Figure 21 loses track of the pedestrian in the semi-sharp turn in the test set, and then re-initialises the track again when it gets new detections. The RL-agent, however, manages to keep the track alive long enough to keep tracking the person using the same track-id.

In Figure 22 we compare the tracking results on a section of the `realworld2` data set. In this data set three people are present. Ones moves from the bottom up along the line of $x = 12$ and the two others are walking behind each other along $x = 10$. The baseline model loses track of the person multiple times, while the RL-based model manages to keep the track-id:s consistent. Furthermore, the tracks are closer to the ground truth in the RL-case and the baseline produced a more wobbly result which is reflected in the MOTP.

Finally, we show the resulting tracks on the whole `realworld1` data set in Figure 23. Also here, the RL-agents do a better job at not loosing track of a person and the resulting tracks are also closer to the ground truth than for the baseline case.

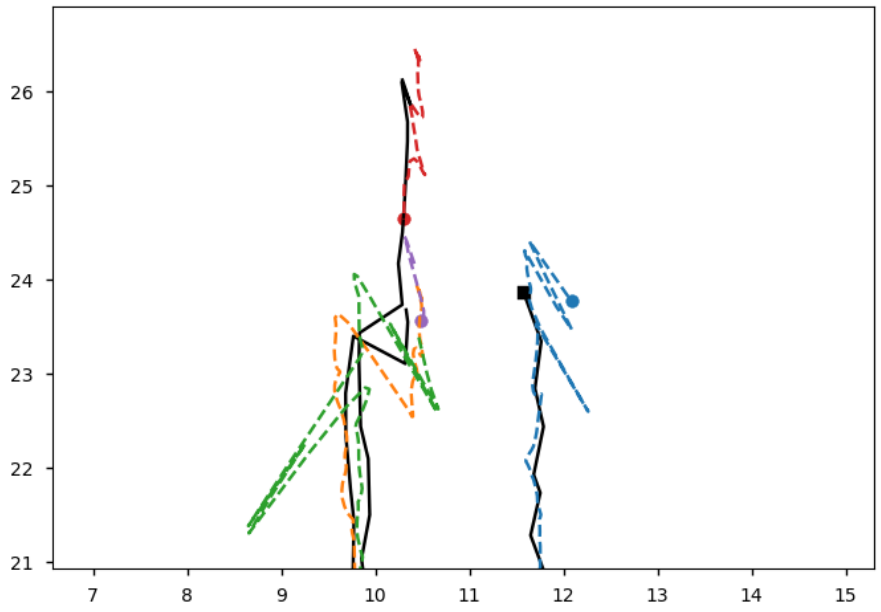


(a) Tracking results from the baseline model.

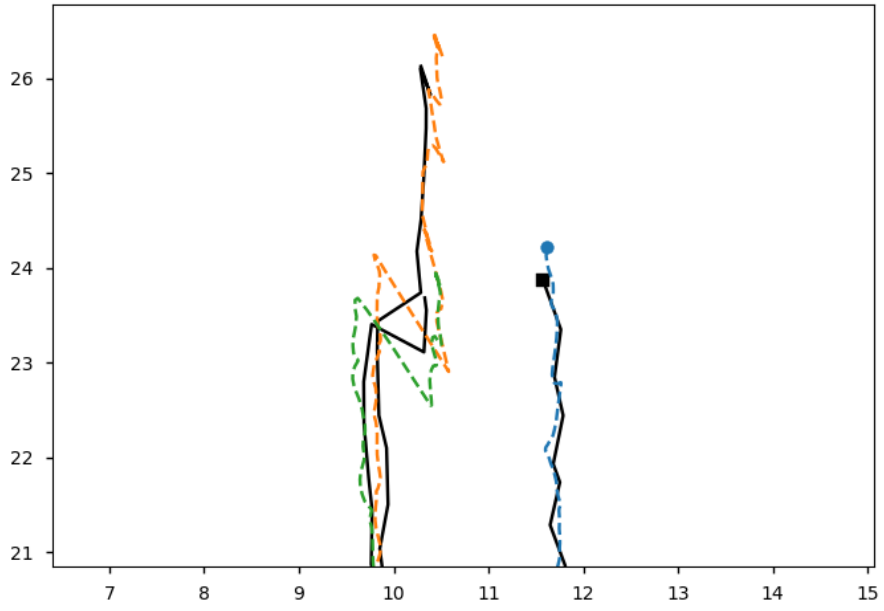


(b) Tracking results from the RL-based model.

Figure 21: Comparison of tracking results between baseline model (a) and RL-based model (b) on a section of the `testdata` data set. The baseline model loses the track in the sharp turn in the bottom right corner, while the RL model follows the ground truth track. Scale in meters.

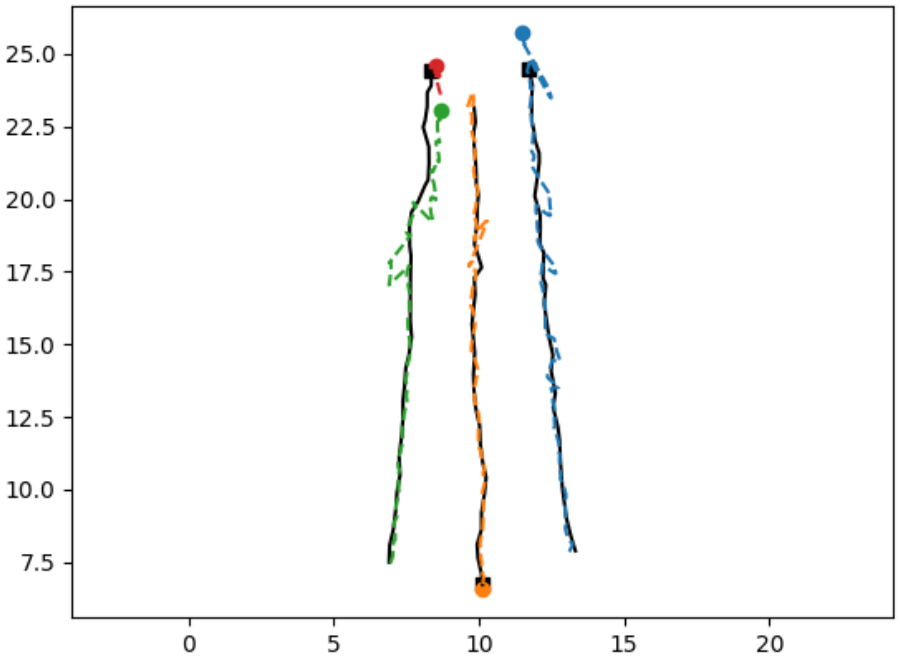


(a) Tracking results from the baseline model.

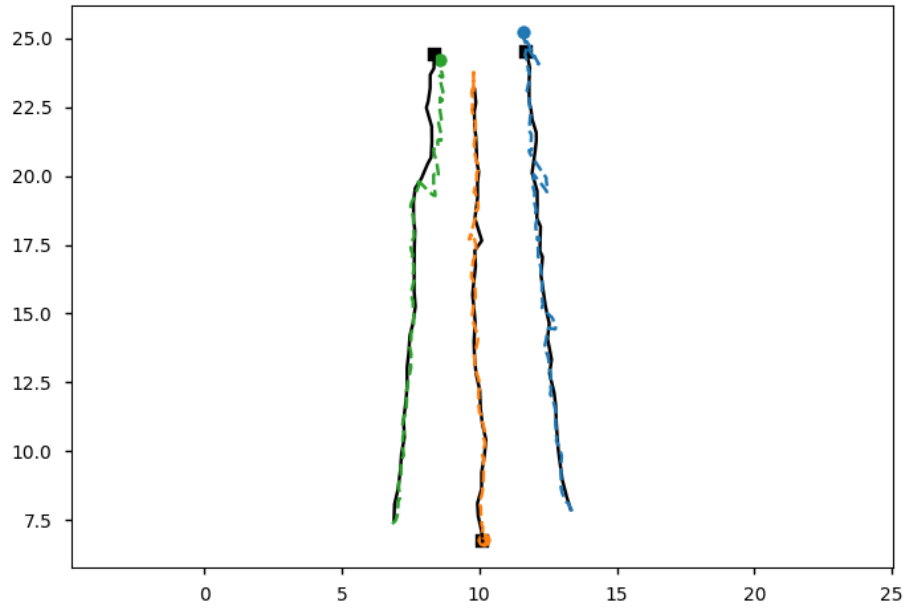


(b) Tracking results from the RL-based model.

Figure 22: Comparison of tracking results between baseline model (a) and RL-based model (b) for a region of the `realworld2` data set. Scale in meters.



(a) Tracking results from the baseline model.



(b) Tracking results from the RL-based model.

Figure 23: Comparison of tracking results between baseline model (a) and RL-based model (b) on the `realworld1` data set. Scale in meters.

5.3.2 Triangulation Results

We now show some further visualisations of the tracking. In Figures 24, 25 below we show how local tracks from different cameras have been associated to the same global track and are fused to form the states of the global track. The yellow, red and blue dots are the states from each camera projected onto the ground plane. We also show the triangulated and Kalman filtered states in purple and the ground truth in black. In the top figure, we see how the associated ground projections from two local tracks, one in Camera 1 and one in Camera 3 are fused using triangulation and then filtered to produce the purple global track. In Figure 25 we see how the pedestrian always visible in camera two, disappears in camera 1 and enters the field of view of Camera 3 and then again becomes visible in Camera 1.

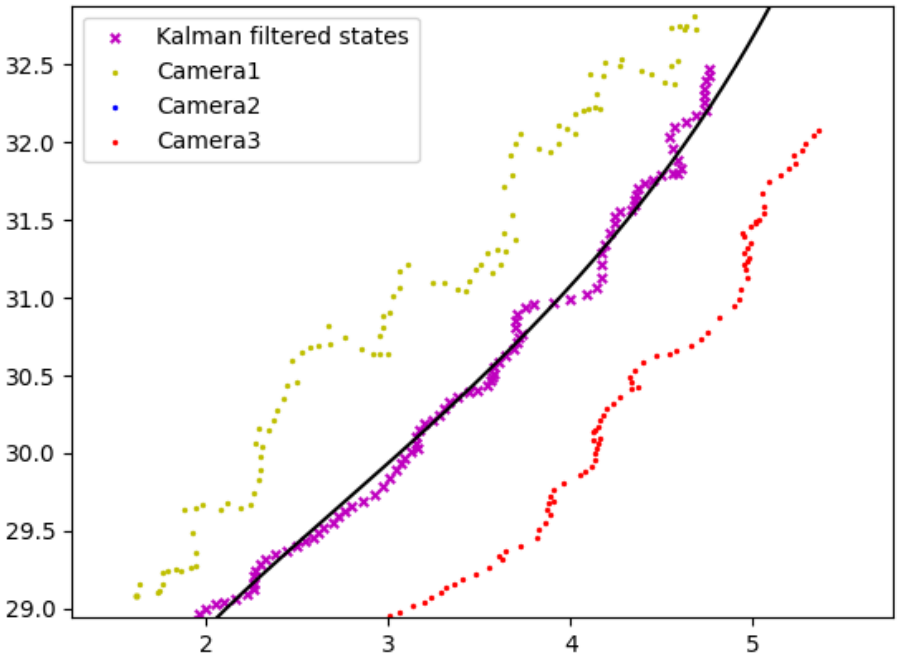


Figure 24: Projected image points as dots and the triangulated and Kalman filtered states as crosses on the `testdata` data set. Solid black line represents the ground truth positions. Scale in meters.

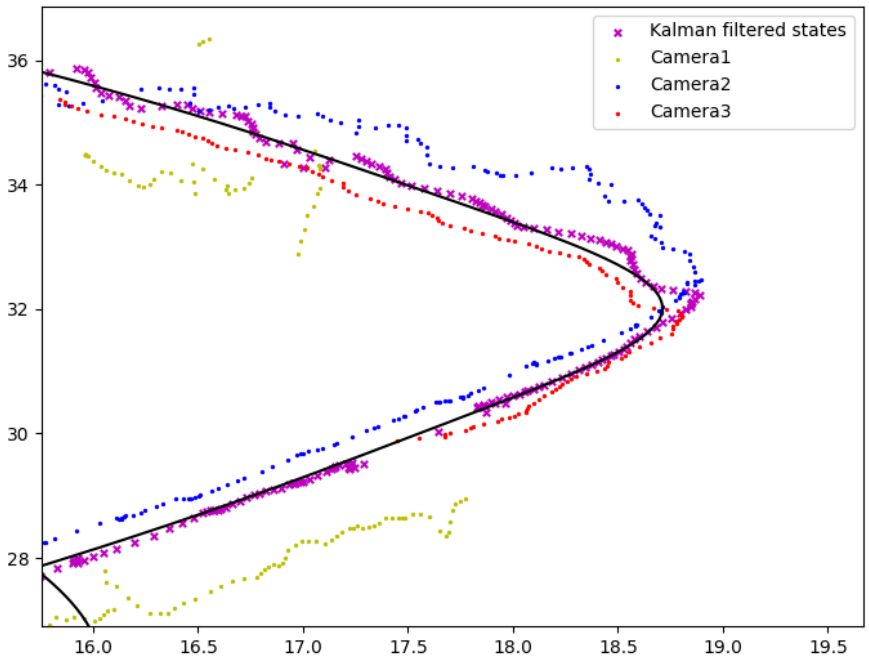


Figure 25: Projected image points as dots and the triangulated and Kalman filtered states as crosses on the `testdata` data set. Solid black line represents the ground truth positions. Scale in meters.

5.4 Reinforcement Training Results

In this section, some results of the training process and the distributions of actions taken by the global reinforcement learning agents are presented. In Figures 26 and 27 below we show a visualisation of the actions taken by the RL-agents on the data sets `trainingdata` and `testdata`. Each row represents the distribution of actions taken by the agents when they received a combination of associated local tracks from the different cameras. The first row represents the situation where no local tracks were associated to the agent ([0,0,0]). The second row represents the situation where the agents receive a local track from the first camera only ([1,0,0]), and so on. Note that a_0 terminates the track, a_1 restarts the Kalman filter, $a_2 - a_8$ use a combination of local tracks to update the Kalman filter, a_9 uses the Kalman prediction to update its state, and a_{10} also uses the prediction but in addition, also puts the track in a hidden state.

In Figures 26 and 27 we can see that when the global RL agents received local tracks, they learned in most cases to use all of them to update the Kalman filter. The RL model however sometimes used a subset of the associated local tracks as opposed to the baseline. Furthermore, the RL model also occasionally ignored the local tracks associated to it and updated the filter using the Kalman prediction,

and occasionally restarted the Kalman filter. When no local tracks were associated to it, illustrated on the first row of Figure 26, the RL agents sometimes used the Kalman prediction and put the track in a hidden state. Sometimes it terminated the track, and sometimes it used an illegal action (triangulating non existing local tracks), making the track stay at the same location.

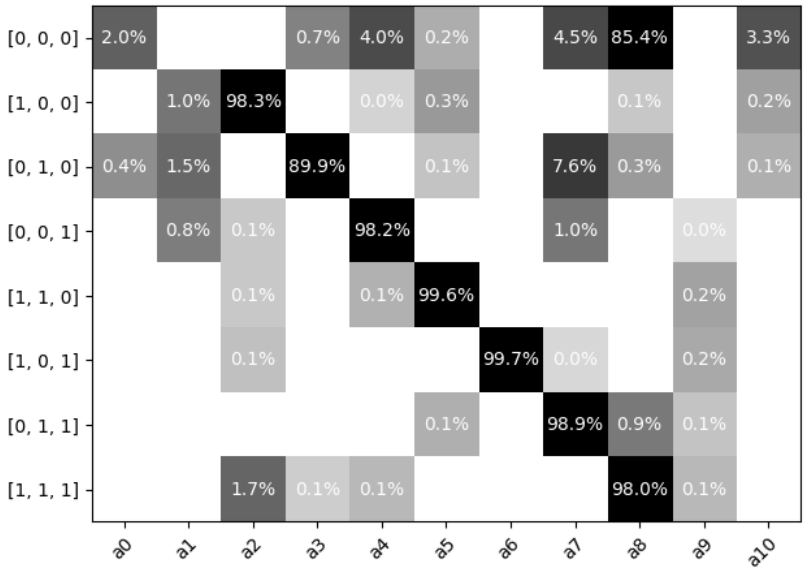


Figure 26: Proportion of times actions $a_0 - a_{10}$ were taken by the RL agents on the `trainingdata` data set. Each row in the figure represents the combinations of local tracks associated to the agent from the cameras, [Camera 1, Camera 2, Camera 3]. A zero (0) means that no local track is associated to the agent from the camera, and a one (1) means that a local track was associated to the agent from the camera. Note that a_0 terminates the track, a_1 restarts the Kalman filter, $a_2 - a_8$ use a combination of local tracks, a_9 uses the Kalman prediction to update its state, and a_{10} also uses the prediction and in addition puts the track in a hidden mode.

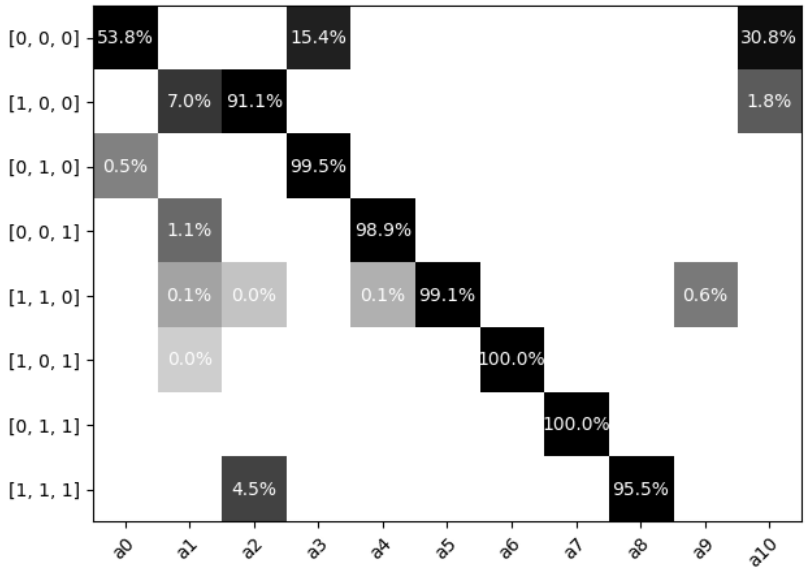


Figure 27: Proportion of times actions $a_0 - a_{10}$ were taken by the RL agents on the `testdata` data set. Each row in the figure represent the combinations of local tracks associated to the agent from the cameras, [Camera 1, Camera 2, Camera 3]. A zero (0) means that no local track was associated to the agent from the camera, and a one (1) means that a local track was associated from the camera.

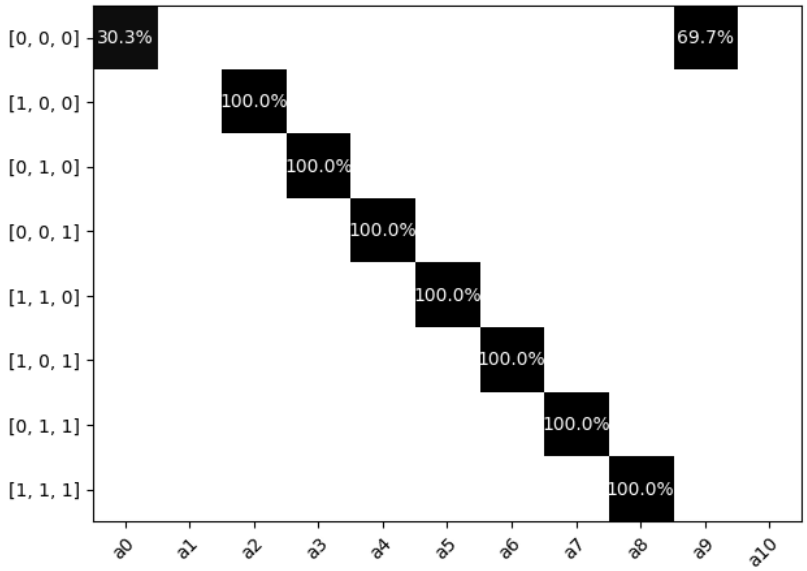


Figure 28: Proportion of times actions $a_0 - a_{10}$ were taken using the SORT baseline on `testdata` data set. Each row in the figure represent the combinations of local tracks associated to the agent from the cameras, [Camera 1, Camera 2, Camera 3]. A zero (0) means that no local track was associated to the agent from the camera, and a one (1) means that a local track was associated from the camera.

The average episode reward for the global RL agent during both pre-training and training are shown in Figures 29 and 30. The local training results have been put in the appendix, Section A.3.

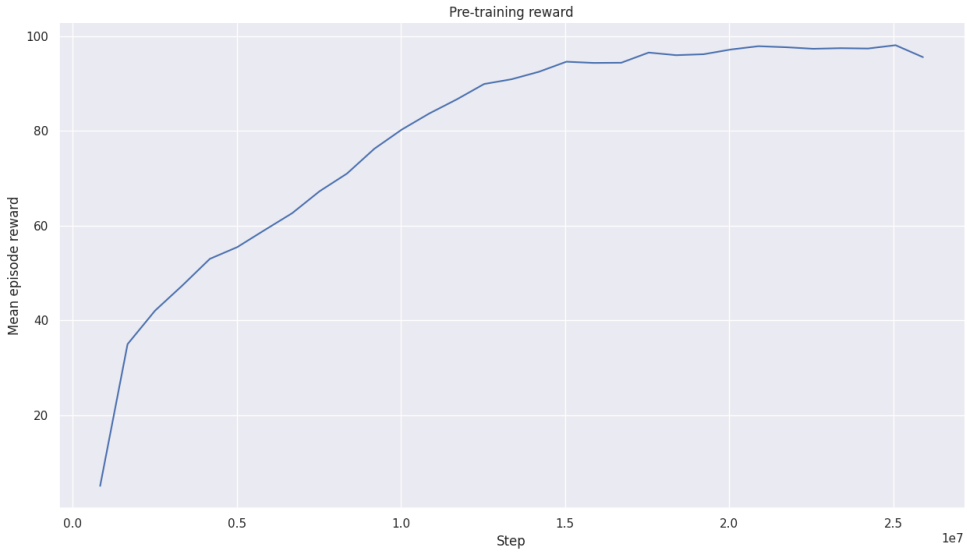


Figure 29: Pre-training reward for the global RL agent

In the pre-training, the RL agent seems to have converged to around 97, meaning that it mimics the baseline thresholds almost perfectly on the training data. Since an episode had length 100, a policy identical to the baseline would get the reward 100. Next we show the results of the training with the final training reward function. According to Figure 30, the RL agents ended up with an average reward of around 410 (the number of steps per episode during training was set to 500 frames), but the variance was still quite high.

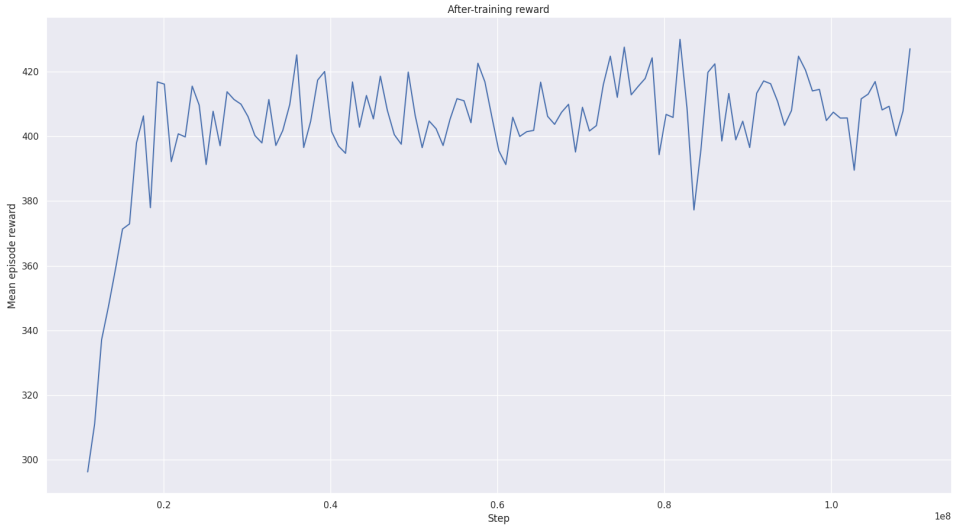


Figure 30: Main training reward

5.5 Time Latency

The YOLO detection had a latency of ≈ 23 ms per frame. Furthermore, the local tracker that runs on a single CPU took ≈ 1.4 ms and the global fusing update took ≈ 4.5 ms per frame on average on the `trainingdata` data set. The code was evaluated using Python and an optimised C implementation would probably be faster. In a setup where the local tracking is run locally in each camera, the resulting process time for each frame would thus be $\approx 23 + 1.4 + 4.5 = 28.9$ ms or 34.6 FPS. It is worth to note that this does not take into account communication latency between the camera and the fusing centre. The estimated FPS shows that the algorithms can be considered real-time. The reinforcement learning-network also had a very short inference time, and thus did not affect the latency of the algorithm compared to the baseline (SORT).

6 Discussion

We will now proceed to discuss the results of this thesis. The first thing to notice is that the reinforcement learning algorithm is on par with the SORT baseline, outperforming the baseline with a few percentages in most scenarios in global setting. Similarly to the results of [34] our experiments confirm that the local RL agents can outperform the local SORT baseline in MOTA score, as can be seen in the appendix Section A.3. However, we only managed to reproduce the results of [34] in Camera 2.

The max distance for the association of detections and tracks (described in Section 4.3.1) was the same for the baseline model and the RL-model, so the improvement in MOTA and MOTP was only due to better choices of what was done with those associations. The baseline model was sensitive to our predefined thresholds. By changing the thresholds in the SORT baseline, we could improve some evaluation measure, while impairing others. The idea with the reinforcement learning approach was to have a more dynamic, non-linear, decision making process for managing the tracks and the detections associated to those tracks. By analysing the actions taken by the RL agent on our data sets, we can conclude that a new, more dynamic decision process was actually learnt.

6.1 Detections and Association

The quality of the detections was a bottleneck in our pipeline. Without accurate detections it is not feasible to achieve a high tracking accuracy. As shown in Table 3 the YOLO detection model had a high accuracy on our data sets. A good data association method is also needed to make the track management work well. The Hungarian algorithm is a good choice but requires that the distance function can accurately compare detections and tracks. For each local camera we used the popular IOU metric.

In the global tracking, the projections of the local tracks from the cameras to the ground plane introduces an error bias. The projections of the local tracks often end up closer to the camera than the ground truth position, which can be detrimental to the associations. The bias can be seen in Figure 24 and is due to multiple factors. Mainly because the central point between the pedestrians' feet (which we project onto the ground plane for association) does not always correspond to the centre bottom position of the bounding box. This problem arises due to the feet not always being aligned with the bottom of the bounding box and that humans are not planes, but have a thickness. The bias thus leads to the projections often being biased towards the direction of the camera. One way to mitigate this is to try to learn the bias or to use a different association cost. There are several algorithms that use visual and motion features for the association like in [26]. We, however, mainly focused on improving the track management.

There is also an important difference between the associations of the local RL agent and the global RL agent. During the association of detections in a camera to a local RL agent, no threshold for the association cost was set (as opposed to the baseline that requires $IOU > 0.3$). This means that in the local RL method it was entirely up to the agent to decide if it wants to use a detection or not. Thus no detections were filtered away in the association.

For the global RL agents it is slightly more difficult to not filter the associations in some way. When associating the local tracks to the global RL agents we perform the Hungarian algorithm over all global tracks, even if the global track is not in the field of view of the camera. Since we do not want to associate local tracks to global tracks that are not not in view of the camera, we implement a threshold of 6 m for association to mitigate most of this problem. The main issue arise when new objects enter the scene and could be erroneously associated to a global track that is not in the field of view of the camera that the object first appeared in. This makes it more difficult to initiate new global tracks as the Hungarian algorithm would try to associate the local track to an existing global track, even if it is outside of the cameras field of view. With a 6 m threshold this would rarely happen and it is up to the agent to learn to not use local tracks associated from cameras in the wrong place.

We could work around this threshold by defining a domain for each of the cameras' (c_k) field of view in world coordinates. Then we could decide to only associate a local track l_k^i to the global tracks $g \in \mathcal{G}$ positioned in the domain of camera c_k. This would ensure that new global tracks could be initiated from local tracks and not be associated to a global track not in field of view. Due to time constraints we did not have time to implement this and the threshold method worked well on our scenes.

6.2 Training and Behaviour of Reinforcement Learning Agents

As mentioned in Section 4.5.1 we had some troubles to get the RL agents to converge in training using only a reward based on MOTA like [34]. We managed to work around this by pre-training the policy network to mimic the policy of the baseline. The reason that the MOTA based reward did not work from start could be that the training data was not diverse enough, that we did not manage to choose good training hyperparameters or that the network architecture was not suitable.

Due to the convergence problems, we pre-trained the RL agents to behave like the baseline first and their polices were quite similar after this stage. After the pre-training (see Figure 30) we managed to get a converging behaviour in training with the MOTA based reward of around 410. This training was done to change the policies of the global RL agents to maximise the MOTA and would thus allow them to move away from the baseline policy. The distributions of actions taken by the RL agents on the test set however showed that the global RL agents still behaved similarly when deciding when to use actions $a_2 - a_8$. These actions correspond to using different combinations of associated local tracks from the cameras to update

the Kalman filter. However, with regards to the other actions the policy changed more. It is interesting to note that the learned policy seems to exploit the fact that "illegal" actions made the track stay in the same position. Those illegal actions were used more often than the Kalman predictions, indicating that the Kalman parameters might not be optimal, or at least that the Kalman predictions were not accurate in situations with missing detections. It would have been interesting to stop the pre-training early and not let the RL-agent learn too much of the baseline's policy. For some of the training cases (for example the local RL agent in camera one see Figure 34 in Section A.3 in the appendix) the training seemed to only fluctuate around some value and not improve much after the pre-training. Perhaps once the local agent learnt the approximate baseline it ended up in some local maxima that was difficult to get out of. If we instead stopped the pre-training earlier and then continued with only a MOTA based reward we might learn a policy further from the baseline. However due to time constraints we did not have time to do this.

We believe that a more diverse training set and a more advanced policy network might increase the performance of the global RL agents. Furthermore, our neural network is a shallow network with ReLU activations. Perhaps using more complex architectures and tricks to help training would help the RL agent to take better actions. This would include things like batch normalisation, skip connections, dropout layers etc. More importantly we believe the agent did not have time to encounter enough situations where actions different from the baseline would benefit the MOTA, and therefore it could not learn a more complex behaviour than it did. These situations could be occlusions as they were somewhat sparse in the training data. The global agent did however outperform the baseline on both the simulated test set and the real-world data. From the results in Table 4 we can see that the agent trained on a synthetic data set got improved tracking results compared to the SORT algorithm on real video recordings. This indicates that it is possible to train our proposed method on simulated data and improve the results on real world data.

6.3 Scene Variations and Simulation of Data

We now proceed and discuss some variations in the scenes and their impact on the performance of the RL agents. First we remind the reader that we have modelled two simulated scenes. The first scene is where we generated the training and test data sets. The second scene is where we generated the `simdata` set (see Figure 14). As expected, the local RL agents performed the best on the camera that they were trained on. The difference in performance between the local RL method and the SORT method was quite small for all the data sets which can be seen in the appendix Section A.3. For Camera 2 in the training and test set, we managed to recreate the results in [34] and gained approximately 3% in MOTA score on the test data set set compared to the SORT algorithm. However, for the other cameras this was not the case and RL agents performed slightly worse on the test set. Due to time constraints we only trained the local RL agents on the cameras from one of

the scenes and we did not train any local RL agents on the cameras of the `simdata` set. Instead, we evaluated the RL agent, trained on Camera 2 in the `trainingdata` set, on the cameras of the `simdata` set. The performance of this local RL agent still managed to perform comparably well on the `simdata` set compared to the SORT algorithm. This was most likely due to the input in image space being normalised with the resolution of the cameras and that the scenes were quite similar from a single camera perspective (i.e parts of the image space that were occluded were in approximately the same locations in both scenes etc).

In a few cameras however, spatial context of the scene differed quite a bit. For example Camera 2 in the training data, when pedestrians enter the scene from the right, they can only do so where the door is located as can be seen in Figure 32b. This differs from Camera 3 where pedestrians can enter along the majority of the right side of the image space (see Figure 32 c). We can thus not expect the local RL agent trained on Camera 2 to perform equally well on Camera 3.

The performance of the global RL policy was highly dependent on the scene. The learned policy did not transfer its knowledge to other scenes with different coordinate systems and camera setups, as shown by the poor results on `simdata` in Table 4. This indicates that the model was overfitted to our particular setup of cameras. However, this bias to a specific camera network and scene is perhaps expected. One could try to train on a much larger model and on more scenes and with different camera locations to reduce the overfitting to a specific scene setup. One could also try to add some type of scene description and camera matrices as input to the policy network to reduce the bias.

Another limitation of the simulated scene we trained on was that the pedestrians were in constant motion. To get more realistic behaviour we would have to spend more time creating the simulated scene but since the constant velocity pedestrian model works quite well as shown in [39] we did not do this. A benefit of simulating data in Blender however is that it allows one to choose exactly what scenarios to train on and it gives you the ground truth of these scenarios too.

6.4 Coordinate Transformation

While the ground in all simulated data was modelled as a perfectly flat plane, this was not true for the real-world data sets where the plane was a coarse approximation. When the ground is not a perfect plane, transformation between image space and the 3D space will not be perfect. In our case, the plane was a rough approximation, but if a more complex ground model exists for a scene, a height map for example, then we could use other techniques such as ray marching to find the intersection between ray and ground plane instead.

As can be seen in Figure 24, the projections of the local tracks had a clear bias. However, as can be seen in the same figure, by using triangulation to fuse the local tracks and filtering the track using the Kalman filter, we managed to get a better

transformation between local coordinates and world coordinates. Furthermore, the further away form the camera a detection was made, the less accurate the projection became. One of the reasons behind using the RL based model was to be able to dynamically use detections with good projections depending on the distance from the cameras, etc. The RL agent did this, at least to some extent, as is shown in Figure 26.

7 Conclusion

7.1 Conclusions

The results show that the simple SORT-baseline works well on our test set. The model is fast, simple and produces tracks close to the ground truth which can be seen by the MOTA and MOTP scores. We also show that the RL method can improve the tracking results compared to the baseline in both image space and in the global coordinates. The global RL method was however highly dependent on the scene that it was trained on.

Finally, we were able to achieve higher tracking accuracy for the RL method compared to the SORT baseline on real-world data by training a model on synthetic data. The network may need to be trained on a more diverse set of scenes in order to produce consistent improvements in tracking results in a more general setting.

7.2 Future work

In the future it would be interesting to incorporate scene descriptions and camera set ups in the RL agent's observation space to remove the scene dependency of the agent.

References

[1] Bernardin, K., Stiefelhagen, R. (2008) *Evaluating Multiple Object Tracking Performance: The CLEAR MOT Metrics*. J Image Video Proc 2008, 246309.

[2] Bewley, B., Ge, Z., Ott L.,Ramos F. and Upcroft B. (2016) *Simple Online and Realtime Tracking*, IEEE International Conference on Image Processing (ICIP) pp. 3464-3468, doi: 10.1109/ICIP.2016.7533003.

[3] Bochkovskiy, A., Wang, C-Y. and Liao, H-Y. M. (2020) *YOLOv4: Optimal Speed and Accuracy of Object Detection*. url: https://arxiv.org/abs/2004.10934v1 (Accessed: 20-03-2022).

[4] Brasó, G., and Leal-Taixé, L. (2020) *Learning a Neural Solver for Multiple Object Tracking*. IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 6246-6256 doi: 10.1109/CVPR42600.2020.00628.

[5] Brendel, W., Amer, M. and Todorovic, S. (2011) *Multiobject tracking as maximum weight independent set*. IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 6246-6256, doi: 10.1109/CVPR42600.2020.00628..

[6] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J. and Zaremba, W. (2016) *OpenAI Gym*. url: https://arxiv.org/abs/1606.01540 (Accessed: 15-04-2022).

[7] Chavdarova, T. et al., (2018) *WILDTRACK: A Multi-camera HD Dataset for Dense Unscripted Pedestrian Detection*, 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, Salt Lake City, pp. 5030-5039, doi: 10.1109/CVPR.2018.00528.

[8] Chiu, H-K., Prioletti, A., Li, J. and Bohg, J. (2020) *Probabilistic 3D Multi-Object Tracking for Autonomous Driving*. url: https://arxiv.org/abs/2001.05673 (Accessed: 17-06-2022).

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

[10] Fabbri, M., Lanzi, F., Calderara, S., Palazzi, A., Vezzani, R. and Cucchiara, R. (2018) *Learning to Detect and Track Visible and Occluded Body Joints in a Virtual World*. url: https://arxiv.org/abs/1803.08319 (Accessed: 17-06-2022).

[11] Ferryman, J. and Shahrokni, A. (2009) *PETS2009: Dataset and challenge,* 2009 Twelfth IEEE International Workshop on Performance Evaluation of Tracking and Surveillance, pp. 1-6, doi: 10.1109/PETS-WINTER.2009.5399556.

[12] Fischler, M. A. and Bolles, R. C. (1981) *Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography*. Commun. ACM 24, 6 (June 1981), 381–395, doi: 10.1145/358669.358692.

[13] Fleuret, F., Berclaz, J., Lengagne, R. and Fua, P. (2008) *Multi-Camera People Tracking with a Probabilistic Occupancy Map* IEEE Transactions on Pattern Analysis and Machine Intelligence. Vol. 30, num. 2, pp. 267-282, doi: 10.1109/TPAMI.2007.1174.

[14] Girshick, R., Donahue, J., Darrell, T. and Malik, J. (2014) *Rich feature hierarchies for accurate object detection and semantic segmentation* IEEE Conference on Computer Vision and Pattern Recognition, pp. 580-587, doi: 10.1109/CVPR.2014.81.

[15] Griwodz, C., Gasparini, S., Calvet, L., Gurdjos, P., Castan, F., et al.. (2021) *AliceVision Meshroom: An open-source 3D reconstruction pipeline*. 12th ACM Multimedia Systems Conference (MMSys 2021), Sep 2021, Istanbul, Turkey. pp.241-247, doi: 10.1145/3458305.3478443.

[16] Guo, Y., Cheung N-M. (2018) *Efficient and Deep Person Re-Identification using Multi-Level Similarity*. IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2018, pp. 2335-2344, doi: 10.1109/CVPR.2018.00248.

[17] Gustafsson, F., Gunnarsson F., Bergman. N., Forssell, U., Jansson, J., Karlsson, R., Nordlund, P-J. (2002) *Particle filters for positioning, navigation, and tracking*. IEEE vol. 50, no. 2, pp. 425-437, doi: 10.1109/78.978396.

[18] Hartley R. and Zisserman A. (2003) *Multiple View Geometry in Computer Vision* Second Edition. Cambridge University Press.

[19] Haykin, S. (2009) *Neural networks and machine learning 3ed*. Pearson.

[20] Heindl, C. (2021) *py-motmetrics*, GitHub repository, url: https://github.com/cheind/py-motmetrics (Accessed: 17-06-2022).

[21] Iguernaissi, R., Merad, D. and Aziz, K. et al. (2019) *People tracking in multi-camera systems: a review*. Multimed Tools Appl 78, pp. 10773–10793, doi: 10.1007/s11042-018-6638-5.

[22] Kalman, R. E. (1960) *A New Approach to Linear Filtering and Prediction Problems*. ASME. J. Basic Eng. March 1960; 82(1): 35–45.

[23] Kingma, D. P. and Ba, J. (2014) *Adam: A Method for Stochastic Optimization*. International Conference on Learning Representations.

[24] Krizhevsky, A., and Sutskever, I. and Hinton, G. E. (2012) *ImageNet Classification with Deep Convolutional Neural Networks*. url: https://papers.nips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf (Accessed: 17-06-2022).

[25] Kuhn, H.W. (1955) *The Hungarian Method for the assignment problem*, Naval Research Logistics Quarterly 2: 83–97.

[26] Liu, W., Octavia C. and Sznaier, M. (2017) *Multi-camera Multi-object Tracking* url: https://arxiv.org/abs/1709.07065 (Accessed: 17-06-2022).

[27] Lowe, D G. (1999) *Object recognition from local scale-invariant features* (PDF). Proceedings of the International Conference on Computer Vision. Vol. 2. pp. 1150–1157. doi:10.1109/ICCV.1999.790410.

[28] Mukilan, P. , Wogderess, S. (2021) *Human object detection: An enhanced black widow optimization algorithm with deep convolution neural network*. url: https://link.springer.com/article/10.1007/s00521-021-06203-3 (Accessed: 17-06-2022).

[29] (2015) *Lecture notes in computer vision* url: https://www.maths.lth.se/media11/FMA270/2015/alllectures.pdf (Accessed: 17-06-2022).

[30] Priisalu Maria, *Lecture notes in machine learning at LTH*

[31] Raffin, A., Hill, A., and Gleave, A., Kanervisto, A., Ernestus, M. and Dormann, N. (2021) *Stable-Baselines3: Reliable Reinforcement Learning Implementations*. Journal of Machine Learning Research. http://jmlr.org/papers/v22/20-1364.html (Accessed: 17-06.2022).

[32] Redmon J., Divvala S., Girshick R. and Farhadi A. (2016) *You Only Look Once: Unified, Real-Time Object Detection*, IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, pp. 779-788, doi: 10.1109/CVPR.2016.91.

[33] Rezatofighi, S. H., Milan, A., Zhang, Z., Shi Q., Dick, A. and Reid, I. (2015) *Joint Probabilistic Data Association Revisited*. IEEE, International Conference on Computer Vision (ICCV), pp. 3047-3055, doi: 10.1109/ICCV.2015.349.

[34] Rosello P. and Kochenderfer M.J. (2018) *Multi-Agent Reinforcement Learning for Multi-Object Tracking*. In Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS '18). International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 1397–1404.

[35] Salscheider, N. O. (2020) *FeatureNMS: Non-Maximum Suppression by Learning Feature Embeddings*. url: https://arxiv.org/abs/2002.07662 (Accessed: 17-06-2022).

[36] Schulman J., Levine S., and Moritz P., Jordan M.I. and Abbeel P. (2015) *Trust Region Policy Optimization*. url: https://arxiv.org/abs/1502.05477 (Accessed: 20-05-2022).

[37] Schulman, J., Wolski, F., Dhariwal P.,Radford, A. and Klimov O. (2017) *Proximal Policy Optimization Algorithms*. url: https://arxiv.org/abs/1707.06347 (Accessed: 05-06-2022).

[38] Sutton R.S and Barton A.G. (2018) *Reinforcement Learning: An Introduction*. Second Edition. MIT Press, Cambridge.

[39] Schöller, C., Aravantinos, V., Lay, F. and Knoll, A. (2019) *What the Constant Velocity Model Can Teach Us About Pedestrian Motion Prediction*. url: https://arxiv.org/abs/1903.07933 (Accessed: 17-06-2022).

[40] Sugimura, D., Kitani, K. M., Okabe, T., Sato, Y., Sugimoto, A. (2009) *Using individuality to track individuals: Clustering individual trajectories in crowds using local appearance and frequency trait*. IEEE 12th International Conference on Computer Vision 2009, pp. 1467-1474, doi: 10.1109/ICCV.2009.5459286..

[41] Terry, J. K. et al. (2020) *PettingZoo: Gym for Multi-Agent Reinforcement Learning*. url: https://arxiv.org/abs/2009.14471 (Accessed: 17-06-2022).

[42] Terry, J. K. and Black, B. and Hari, A. (2020) *SuperSuit: Simple Microwrappers for Reinforcement Learning* Environments. url: https://arxiv.org/abs/2008.08932 (Accessed: 17-06-2022).

[43] Thiago T.S. and Morimoto C.H. (2011). *Multiple camera people detection and tracking using support integration*,Pattern Recognition Letters, Volume 32, Issue 1, Pages 47-55, ISSN 0167-8655.

[44] Wojke, N., Bewley, A. and Paulus, D. (2017) *Simple and Online Realtime Tracking With A Deep Association Metric*. url: https://arxiv.org/pdf/1703.07402.pdf (Accessed: 17-06-2022).

[45] Xu, S., Shin, H. and Tsourdos, A. (2019) *Distributed Multi-Target Tracking with D-DBSCAN Clustering*. Workshop on Research, Education and Development of Unmanned Aerial Systems (RED UAS), 2019, pp. 148-155 doi: 10.1109/REDUAS47371.2019.8999712.

[46] Yang, B., Huang, C., Nevatia, R. (2011) *Learning affinities and dependencies for multi-target tracking using a CRF model*. CVPR pp. 1233-1240, doi: 10.1109/CVPR.2011.5995587 .

[47] You, Q., Jiang, H. (2020) *Real-time 3D Deep Multi-Camera Tracking*. url: https://arxiv.org/abs/2003.11753 (Accessed: 17-06-2022).

[48] Yu, C., Velu, A., Vinitsky, E. Wang, Y., Bayen, A. M. and Wu, Y. (2021) *The Surprising Effectiveness of MAPPO in Cooperative, Multi-Agent Games*. https://arxiv.org/abs/2103.01955 (Accessed: 17-06-2022).

[49] Zhang, L., Li, Y., and Nevatia, R. (2008) *Global data association for multi-object tracking using network flows*. IEEE Conference on Computer Vision and Pattern Recognition, 2008, pp. 1-8, doi:doi: 10.1109/CVPR.2008.4587584.

A Appendix

A.1 Data sets

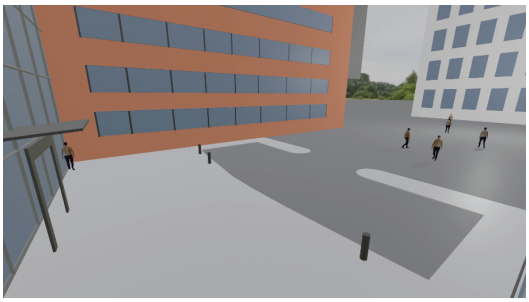
The following table displays information about the data sets used in this thesis.

Name	Type	Number of cameras	Number of persons	Frames	FPS	Duration (s)	Resolution
`simdata`	Simulated	3	6	600	24	25	2688 × 1512
`realworld1`	Recorded	2	3	321	30	10	2688 × 1512
`realworld2`	Recorded	2	3	371	30	12	2688 × 1512
`realworld3`	Recorded	2	5	441	30	17	2688 × 1512
`trainingdata`	Simulated	3	12	5000	24	208	2688 × 1512
`testdata`	Simulated	3	8	1000	24	41	2688 × 1512

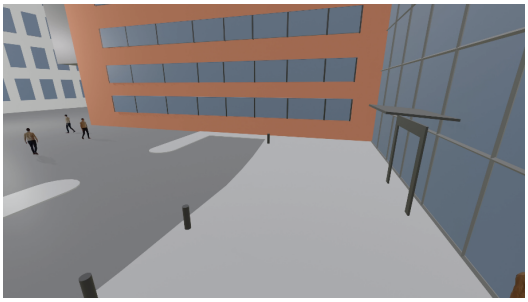
Figures 31 - 33 show the views from the cameras in the different data sets.

(a) Camera 1. (b) Camera 2.

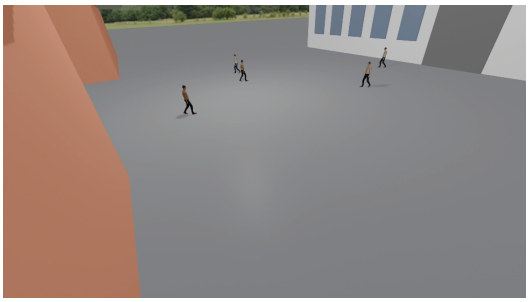
Figure 31: Frame from data set `realworld1`.



(a) Camera 1.



(b) Camera 2.



(c) Camera 3.

Figure 32: Frame from data set `testdata`.

(a) Camera 1.

(b) Camera 2.

(c) Camera 3.

Figure 33: Frame from data set `simdata`.

A.2 Local Tracking Results

The following tables show the results for mono-camera tracking in each camera in all of our data sets using both the baseline and reinforcement learning method.

Dataset	MOTA↑	MOTP↓	FP↓	FN↓	IDs↓	Precision↑	Recall↑
`trainingdata`, Cam 1, Baseline	92.3 %	0.229	1420	**1933**	319	97 %	**95.9 %**
`trainingdata`, Cam 1, RL	**93.9 %**	**0.225**	**470**	2138	**294**	**99.0 %**	95.5 %
`trainingdata`, Cam 2, Baseline	91.4 %	0.253	1320	**635**	142	94.7 %	**97.4 %**
`trainingdata`, Cam 2, RL	**94.7 %**	**0.245**	**373**	804	**117**	**98.4 %**	96.6 %
`trainingdata`, Cam 3, Baseline	**97.0 %**	**0.226**	**232**	**409**	136	**98.4 %**	**99.1%**
`trainingdata`, Cam 3, RL	94.9 %	0.235	516	693	**81**	98 %	97.3%

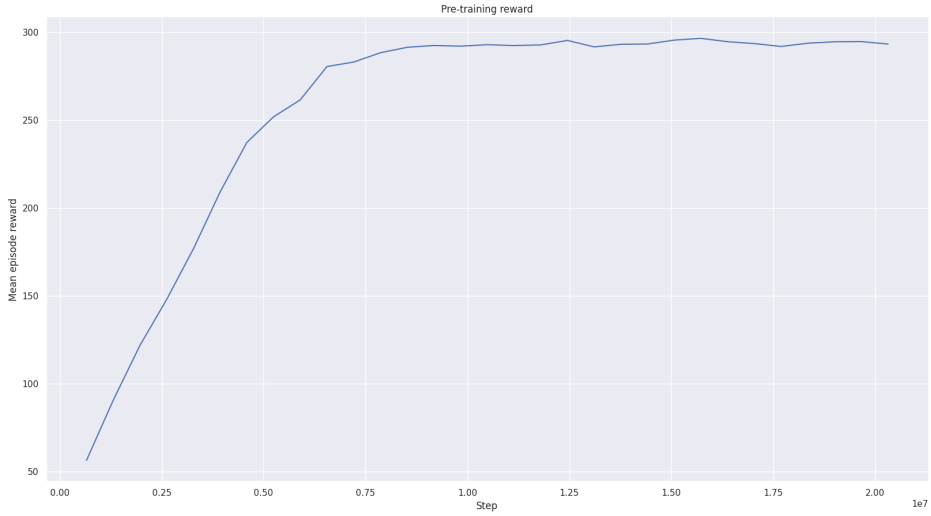
Dataset	MOTA↑	MOTP↓	FP↓	FN↓	IDs↓	Precision↑	Recall↑
`testdata`, Cam 1, Baseline	**94.6 %**	0.212	150	**194**	25	97.8 %	**97.1 %**
`testdata`, Cam 1, RL	94.1 %	**0.209**	**93**	286	25	**98.6 %**	95.8 %
`testdata`, Cam 2, Baseline	88.6 %	0.234	198	**144**	**10**	**97.8 %**	**97.1 %**
`testdata`, Cam 2, RL	**91.4 %**	**0.219**	**96**	157	11	96.8 %	94.9 %
`testdata`, Cam 3, Baseline	**97.8 %**	**0.209**	**37**	**57**	15	**99.2%**	**98.8%**
`testdata`, Cam 3, RL	97.7 %	0.214	42	65	**7**	99.1 %	98.7 %

Dataset	MOTA↑	MOTP↓	FP↓	FN↓	IDs↓	Precision↑	Recall↑
`simdata`, Cam 1, Baseline	**92.4 %**	**0.231**	**62**	**153**	31	**98.0 %**	**95.3%**
`simdata`, Cam 1, RL	83.8 %	0.237	322	176	**24**	90.4 %	94.5 %
`simdata`, Cam 2, Baseline	**97.0 %**	**0.226**	**19**	**52**	28	**99.4 %**	**98.4 %**
`simdata`, Cam 2, RL	96.1 %	0.239	38	80	**13**	98.9 %	97.6 %
`simdata`, Cam 3, Baseline	**82.3 %**	0.223	**98**	377	33	**96.2 %**	86.9 %
`simdata`, Cam 3, RL	79.8 %	**0.220**	180	**372**	**28**	93.3 %	**87.0 %**

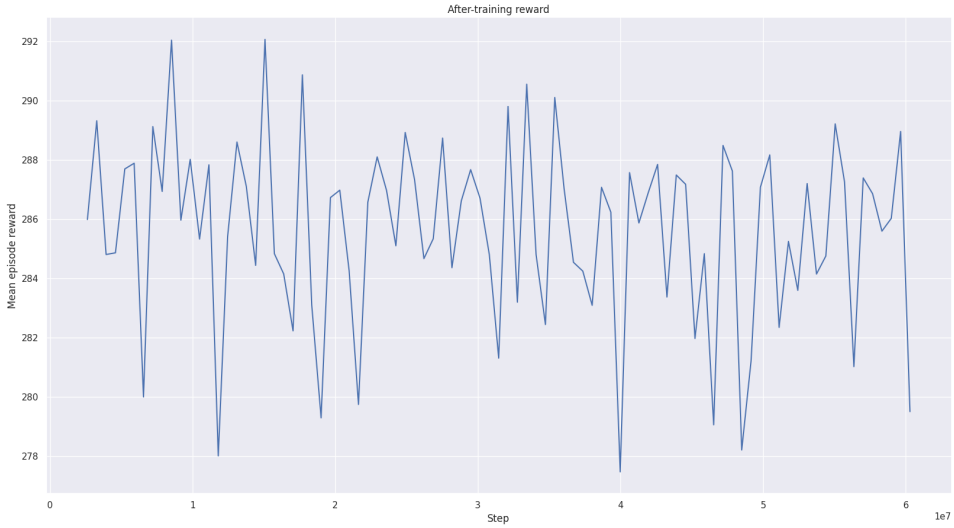
Note that the evaluation on the simdata set in the table above, we used an RL agent trained on the trainingdata set due to time constraints. The RL agent was trained on Camera 2 (we only include it for completeness).

A.3 Local RL agent training results

In the following section we show the training results of the pre-training and 'post-training' in each camera of the training data set.

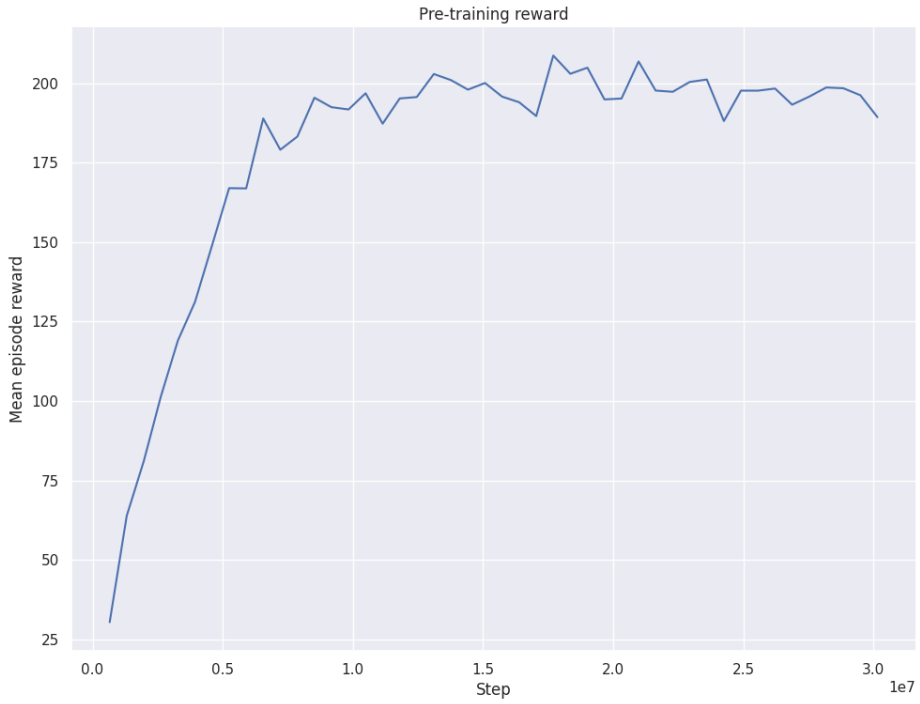


(a) Average rewards during pre-training per episode for Camera 1.



(b) Average rewards during training per episode for Camera 1.

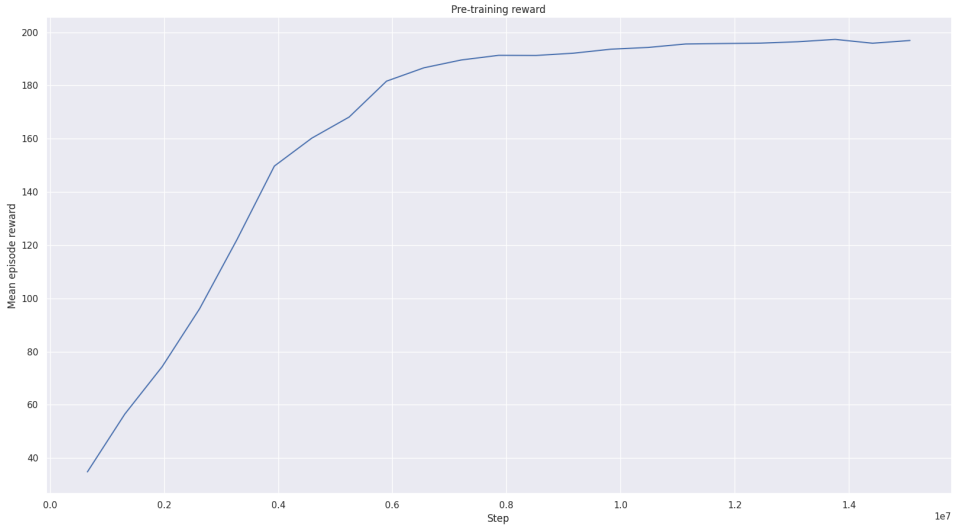
Figure 34: The figure shows how the reward increased during training. The episode lengths were in this case 300 frames both in pre-training and training.



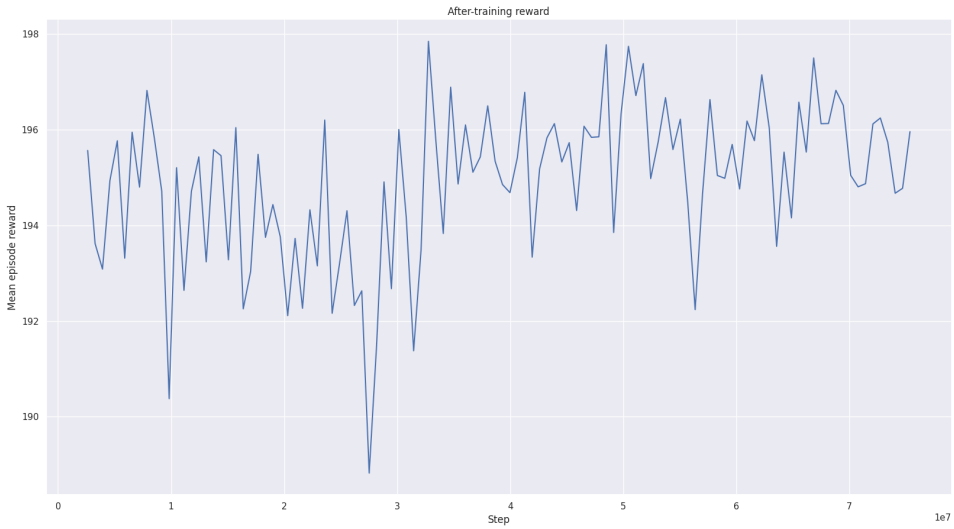
(a) Average rewards during pre-training per episode for Camera 2.

(b) Average rewards during training per episode for Camera 2.

Figure 35: Shows how the reward increased during training. Note that the pre-training were trained over 200 frames and 300 frames in training..



(a) Average rewards during pre-training per episode for camera 3.



(b) Average rewards during training per episode for camera 3.

Figure 36: Shows how the reward increased during training. Note that the training was done with an episode length of 200.

A.4 Summary of hyperparameters

Hyperparameter	Local pre-training	Local training	Global pre-training	Global training
`Learning Rate`	10^{-4}	5×10^{-5}	10^{-4}	5×10^{-5}
`Discount Factor` γ	0.95	0.95	0.95	0.95
`Clip Range`	0.2	0.2	0.2	0.2
`Batch Size`	256	256	128	128

A.5 Kalman parameters

Global Kalman parameters:

$$\mathbf{R} = \text{diag}(4, 4), \sigma_{xy} = 1$$

Local Kalman parameters:

$$\mathbf{R} = \text{diag}(1e1, 1e1, , 1e2, 1e2), \sigma_{xy} = 1, \sigma_a = 1e-4, \sigma_r = 1e-4$$

A.6 Images

Figure 6 licensed under the Creative Commons Attribution-Share Alike 4.0 International license. Source: Wikimedia Commons. Author: Adrian Rosebrock.

Illustration 10 from [1] under Creative commons licence.

Master's Theses in Mathematical Sciences 2022:E32
ISSN 1404-6342

LUTFMA-3480-2022

Mathematics
Centre for Mathematical Sciences
Lund University
Box 118, SE-221 00 Lund, Sweden

http://www.maths.lth.se/