UNIVERSITÀ DEGLI STUDI DI PADOVA

Dipartimento di Fisica e Astronomia “Galileo Galilei”

Dipartimento di Matematica “Tullio Levi-Civita“

Master Degree in Physics of Data

Final Dissertation

Computer Vision models for Multi-Object visual tracking: evaluations and real-world applications.

Thesis supervisor	Candidate Name
Prof. Lamberto Ballan	Andrea Nicolai
	Mat. 1233407

Academic Year 2021/2022

Abstract

Within the Artificial Intelligence framework, the Multi-Object Tracking problem lies with detecting targets from videos and reconstructing their trajectories in space, and it is commonly exploited for surveillance tasks. To provide a common and accepted benchmark for algorithms proposed by the research community, *MOTChallenge* was proposed [1]. In this work, after a formalization of the main concepts underlying the MOT problem, namely how to properly define the problem and what metrics are involved, we study and select two of the State-Of-The-Art trackers according to such a benchmark: *ByteTrack* [2] and *FairMOT* [3]. Then, we modify ByteTrack to account for visual cues, in a fashion similar to FairMOT, training it on the annotated MOT17 dataset. Finally, with the network trained for the MOT20 competition [4], we perform the tracking of players during a football match, using as input the video recorded by a static camera placed in the center of the field [5]. The authors also provided players' data coming from XYZ sensors worn by the home team. An algorithm is implemented to preprocess the video, correct the radial distortion, and project the tracklets from the image into pitch coordinates, finally assigning the detected players and their tracklets to the trajectories made available by the sensor. While the use of the re-identification feature does not seem to improve the tracker performance, our algorithm is found to be able to assign a tracklet, on average, to the $\sim 60\%$ of the trajectory of sensors.

A mia mamma Denise,
che c'è sempre stata e sempre ci sarà.
A Marnie,
che mi ha accompagnato durante questo percorso.
A tutti quelli che credono in me e mi hanno dato fiducia.
A tutti quelli che mi vogliono bene.

Contents

Introduction

Artificial intelligence has started to play a prominent role in the whole society, helping industries develop their business, researchers to make new discoveries, and administrations to better understand people's needs, thus building smarter and safer cities. In particular, Multi-Object tracking was mainly designed and developed to address surveillance problems. However, MOT algorithms have also found application in many other fields, such as biology [6], self-driving vehicles [7], and *sport analytics* [8]. To help the research community by providing a fair performance evaluation and a common benchmark of the proposed algorithms, *MOTChallenge* [1] was created, together with some public datasets on which algorithms can be trained. Especially with referral to sport analytics [8], broadcast videos, data obtained from wearable sensors, and statistics provided by official associations have been fused to provide a deeper and better quality inspection of athlete performance, giving advantage to teams that exploit these techniques on those who do not. However, obtaining an insightful evaluation of any event, whether it is a match, the performance of a player, or assessing technical and physical skills of a player, requires a lot of data, and consequently a conspicuous budget in terms of money and human resources and eventually large hardware capabilities.

However, it is technically possible to perform such an analysis using Artificial Intelligence methods, thus decreasing the amount of resources needed. In fact, using a really low budget, consisting of a fixed camera mounted on a pole recording the match, one might be able to extract the statistics of the game or the training depending on the application, reconstructing the trajectory of players throughout time. This is actually different from other approaches that rely on broadcast images: as they are not readily available for non-professional teams with limited budget.

Therefore, the goal of this work is to provide an easy but correct pipeline to address such a problem, fusing available data from multiple sources and handling eventual problems that one can encounter in a rigorous approach. As a first step, we proceed to properly define and formalize the MOT problem [9], what are the observables, and how and from what perspectives it can be solved. Later, we will introduce the common metrics accepted and used by the research community [10] to provide an evaluation of

MOT algorithms. We mention as examples: CLEARMOT (MOTA, MOTP), IDF1, and the most basic one, which is the Jaccard Index [11], which takes the form of the Intersection over Union when dealing with spatial region information. Afterwards, some applications related to physics science are described: in particular, we mention how people's behavior can be modeled through a formalism that belongs to physics [12]. In such a perspective, the exploitation of visual data coming from surveillance cameras in public places, and the following reconstruction of trajectories, can be a huge source of data to assess the quality of such models.

Subsequently, we discuss the one-stage detector architecture, whose task is to detect and locate objects represented in an image. After a brief overview of them and a short presentation about the YOLO family [13], focusing in particular on the functioning of the latest implementation *YOLOX* [14], we present the dataset we used to train our network: $MOT17$ [15]. Then, our focus moves mainly on two different tracking algorithms, namely *ByteTrack* [2] and *FairMOT* [3]. These two algorithms have been chosen due to their different approaches to solving the MOT problem, moreover, the first ranks first in $MOT17$ and $MOT20$ competitions. Then, since *ByteTrack*, which performs the best, does not handle any visual information, we try to add it a head with reidentification tasks, in such a way that also *visual* cues would be exploited, and we compare two different ways to train our "hybrid" network.

Finally, with the original *ByteTrack* network trained for the $MOT20$ competition, we start tracking players during a football match, whose videos are publicly available together with data from a XYZ sensor [5]. With the aim of exploiting all the information we have, we fuse the position information in pitch coordinates with the one obtained by the tracking algorithm. The main difficulty relies on the fact that after a player has been detected and located, its image coordinates need to be transformed to the physical world ones. However, this can occur only after the image has been pre-processed and the radial distortion has been corrected. Moreover, the data coming from the sensor might need some manipulation in terms of handling timestamps and filtering out players outside of the visibility of the camera, which obviously cannot be detected. The last step taken consists of resampling the timestamps associated with the video in such a way to be comparable with the sensor data, and, through the *Kendall* τ correlation [16], determining whether a quantity is delayed in time with respect to the other and, eventually, reshifting it. In addition, we check whether the use of different image resolution might have an impact on detector performance: we try to use two videos at different resolutions as inputs, and see whether the count of correct and incorrect detections change.

In the end, having reconstructed the players' tracklets on the pitch, we implement a simple algorithm through which one can assign them to sensor trajectories, providing a *score* of how long players are successfully tracked.

Thesis Organization

This thesis is divided into 5 chapters. In the first one, we provide a general overview of what an MOT problem is and different techniques and how and what data are manipulated to solve it. We discuss the metrics according to which a tracking algorithm is better than another, finally mentioning some physics application linked to trackers. In the second chapter, we discuss the tools we use, namely one-stage detectors, especially focusing on YOLOX. Then, we introduce the MOT datasets and explain what the core concepts of the tracking algorithms considered the State-Of-The-Art in Multiple-Object Tracking: ByteTrack and Fairmot. In the third chapter, we focus on the application to a more challenging problem, introducing our dataset, in particular the video and sensor specifications. We proceed to present the whole preprocessing made and the algorithm for assigning tracklets obtained thanks to bounding boxes to those of the sensor. Furthermore, we explain how a head with reidentification task is added to ByteTrack and trained, in a fashion similar to that of FairMOT. In the fourth chapter, we compare the results between our "hybrid" model with two different losses on $MOT17$ dataset, we understand whether different video resolutions have an impact on detection performance and speed inference, and we discuss the results of our assignation algorithms. In the fifth chapter, we conclude by discussing what could have been done better and what could be future outcomes of this work.

The Multiple-Object Tracking problematic

Among the many Artificial Intelligence challenges in Computer Vision, Multi-Object Tracking (abbr. MOT), sometimes known as Multi-Target Tracking (abbr. MTT), is one of the most challenging. However, if it is resolved, it can yield extremely helpful information. In fact, the tracker output allows researchers to model behaviors and interactions between the targets under investigation. In this context, MOT algorithms have been used to successfully track pedestrians [2], animals [17], vehicles [7], or even cells at microscopic scales [6]. However, the most well-studied targets of MOT are pedestrians [9], especially with referrals to video surveillance tasks or scene understanding. Alternatively, this framework can find applications in other fields that are more of our interest, such as sports analysis of television broadcast images [18] [19].

Let us now formalize the MOT problem following the framework used by Luo et al. [9], and how it can be solved using different approaches. Then, we will introduce the metrics used to provide an evaluation of these algorithms [20] [10] [21], concluding by discussing how physics can successfully model pedestrian behavior via the *social force model* [12] [22].

1.1 Formalization of the problem

In this section, we will provide a formal description of such a problem, according to the efforts made by W. Luo et al. [9] in deriving a unified formulation from previously existing works [23]. Due to the enormous quantities of different approaches available, we will delve into those of interest to us while referring to the literature of interest to others.

MOT can be regarded as a multivariable estimation problem. Given a subsequent series of frames (e.g., a video), the aim is to estimate the state of the i-th object in the t-th frame. We refer to this quantity as $\mathbf{s}_t^i$. By generalization, the state of all the M_t objects at time t is: $\mathbf{S}_t = (\mathbf{s}_t^1, \mathbf{s}_t^2, ..., \mathbf{s}_t^{M_t})$. For the i-th object appearing for the first time in frame i_s, and whose last appearance is in frame i_e, its sequential state is denoted as: $\mathbf{s}_{i_s:i_e} = (\mathbf{s}_{i_s}, \mathbf{s}_{i_{s+1}}, ..., \mathbf{s}_{i_e})$. Consequently:

$$\mathbf{S}_{1:t} = \{\mathbf{S}_1, \mathbf{S}_2, ..., \mathbf{S}_t\} \tag{1.1}$$

is the set of all sequential states of *all* objects starting from the first frame, whose index is 1, through the t-th frame.

According to the paradigm of *Detection-Based-Tracking* (DBT) [9], which is the most widely used and the one used in our work, the vector of M_t **observations** (i.e., detected objects) at t-th time is $\mathbf{O}_t = (\mathbf{o}_t^1, \mathbf{o}_t^2, ..., \mathbf{o}_t^{M_t})$, where $\mathbf{o}_t^i$ denotes the detection of the i-th single object at time t. Similarly to before, the set of all sequential detections of all objects starting from time $t = 1$ throughout time t is:

$$\mathbf{O}_{1:t} = \{\mathbf{O}_1, \mathbf{O}_2, ..., \mathbf{O}_t\} \tag{1.2}$$

The final goal is therefore to compute the **optimal** sequential states for all objects, usually modeled by performing the **Maximum a posterior** (MAP) $\hat{\bullet}$ estimation from the *conditional* distribution of the states given the set of observations made:

$$\hat{\mathbf{S}}_{1:t} = \arg\max_{\mathbf{S}_{1:t}} P(\mathbf{S}_{1:t}|\mathbf{O}_{1:t}) \tag{1.3}$$

One usually exploits **probabilistic inference** perspective, namely, according to the following steps:

- **Predict**:
$$P(\mathbf{S}_t|\mathbf{O}_{1:t-1}) = \int P(\mathbf{S}_t|\mathbf{S}_{t-1})P(\mathbf{S}_{t-1}|\mathbf{O}_{1:t-1})\mathrm{d}\mathbf{S}_{t-1} \tag{1.4}$$

- **Update**:
$$P(\mathbf{S}_t|\mathbf{O}_{1:t}) \propto P(\mathbf{O}_t|\mathbf{S}_t)P(\mathbf{S}_t|\mathbf{O}_{1:t-1}) \tag{1.5}$$

where one can distinguish between **Dynamic Model** $P(\mathbf{S}_t|\mathbf{S}_{t-1})$ and **Observation Model** $P(\mathbf{O}_t|\mathbf{S}_t)$. On the contrary, one can exploit **deterministic** algorithms (e.g., Hungarian Algorithm [24] [25]) to solve such a problem to directly maximize the *likelihood function* $L(\mathbf{O}_{1:t}|\mathbf{S}_{1:t})$ over the observation set $\{\hat{\mathbf{O}}_{1:n}^n\}$:

$$\hat{\mathbf{S}}_{1:t} = \arg\max_{\mathbf{S}_{1:t}} P(\mathbf{S}_{1:t}|\mathbf{O}_{1:t}) = \arg\max_{\mathbf{S}_{1:t}} L(\mathbf{O}_{1:t}|\mathbf{S}_{1:t}) = \arg\max_{\mathbf{S}_{1:t}} \prod_n P(\hat{\mathbf{O}_{1:t}^n}|\mathbf{S}_{1:t}) \tag{1.6}$$

The main difference between the two approaches is based on the results that can be obtained, which are *deterministic* or *stochastic*. Using stochastic methods, the output of the tracker is not unique, given the same output. As an example of a **stochastic** method, we can mention *particle filters* [26], where, depending on the randomness of the way the particles are generated, the results can vary. In contrast, when the output is constant and the algorithm is run several times, we refer to it as **deterministic**. Usually, data association takes advantage of deterministic optimization methods, such as the aforementioned Hungarian algorithm, as we will do throughout this work.

In the DBT paradigm, objects are first detected and then linked to trajectories. Let us now discuss how the detection is performed, and different approaches used in the last years.

1.1.1 Object detection

The first step for MOT algorithms is the actual detection of objects. Multiple approaches to the problem are possible, and a broad classification of these is:

- **Appearance based**: makes use of image-processing techniques to directly recognize objects from videos. However, a possible complication arises when objects are occluded, that is, not being visible.
- **Motion based**: sequences of images are used for object recognition, without any cues or feature embedding. However, this approach may fail in detecting objects in complex scenarios, such as noisy backgrounds or crowded scenes.
- **Deep Learning based**: either one or both previous approaches can be combined in different ways. Such methods have been adopted by State-Of-The-Art algorithms in recent years, thanks to the advancements in terms of computational power and because they are found to perform the best [27].

Furthermore, tracking devices can be classified according to their **initialization**, which depends mainly on how and whether detection is made. **Detection-based** trackers accept as input a sequence of frames, thus performing the detection and building of some object hypotheses. An alternative to this approach is the so-called **Detection-free** tracking: the number of objects to be tracked, and their locations, is fixed and *manually* set at the *first* frame. Therefore, the algorithm will follow them in subsequent frames. However, this approach is used more rarely, as it is unable to track new incoming objects and cannot terminate out-of-field trajectories. For these reasons, we will adhere to the DBT paradigm. In this approach, **tracking** itself occurs by linking the detection hypotheses into trajectories. However, it should be noted that the object detector, although it may rely on a common architecture, must be *trained* in advance to deal with specific targets. In other words, it is highly domain-specific (e.g., pedestrians, vehicles). In addition, the final performance is largely affected by the goodness of the detector employed: clearly, if an object is not detected, it cannot be related to any trajectory.

Depending on whether future observations are used, one can distinguish between *online* and *offline* trackers. **Online** trackers are also known as *casual* trackers, since they use *only of past* frames to predict the future state. On the contrary, **Offline** trackers can use both past and future information, pursuing global optimization throughout the sequence. Finally, depending on the throughput of the tracker, that is, how many *frames per seconds* can be handled, *online trackers* can be used for **real-time** tasks creating trajectories on the fly.

1.1.2 Object tracking

As already stated, a MOT focuses on the **detection** of *multiple objects* in individual frames, which means recovering identity information over time, that is, **finding trajectories**. However, attention should be paid both to how to compute the similarity between distinct objects and to how to assign a unique id to every instance of such objects across frames.

Appearance Modeling

Computing the affinity between several objects might take advantage of building an appearance model. However, the task is different from Single Object Trackers (*SOT*): there, a unique object needs to be clearly distinguished (i.e. detected) from the background and consequently tracked. [28] However, in MOT, several objects that share common visual features might appear; therefore, one would not consider appearance modeling as a *core* component in MOT. Formally, one would like to compute the similarity S_{ij} between two different observations i,j:

$$S_{ij} = F(\mathbf{o}_i, \mathbf{o}_j) \tag{1.7}$$

where $\mathbf{o}_i$, $\mathbf{o}_j$ are **visual representations** of different observations, while $F(\cdot, \cdot)$ is a function that measures their similarity.

Visual representations can be grouped into **local** or **region** features. As the name would suggest, local features refer to some peculiar croppings or segmentations of the image that would help to recognize and subsequently track different objects. Despite the fact that their employment may turn out to be efficient, their presence is strongly affected by eventual occlusions. On the contrary, the *region* features exploit information from a wider range, which might even be the whole picture or the bounding box that contains an object. It can take the form of a color histogram, in the case of a direct comparison between two pixel regions, or eventually a more elaborated analysis like the gradient-based representation (Histogram Oriented Gradients *HOG* [29]) or, even more complex, pursue a covariance matrix analysis. The former one is helpful for modeling object shapes and can handle "rigid" transformations, defecting, however, when the pose changes or the object becomes occluded. In contrast, covariance matrix analysis might overcome these problems at a higher computational cost.

After having extracted the visual representations modeled above, one has to compute some measure of similarity between them. Here, it is possible to either exploit single information or alternatively fuse **multiple** cues by using multiple strategies (e.g., concatenation, summation, boosting). However, exploiting **single cues**, there are several approaches to compute similarities, which could also depend on the domain. When dealing with color histograms, it is common to exploit their *Bhattacharyya* distance [30] $B(\cdot,\cdot)$, which becomes a similarity measure of the kind $S(\mathbf{T}_i, \mathbf{T}_j) = e^{-B(\mathbf{T}_i,\mathbf{T}_j)}$.

Another easy and eventually more natural measure to compute the similarity between two vectors of an inner product space is the **cosine similarity** [31]. Roughly, it determines whether two vectors are pointing in the same direction, by measuring the cosine of the angle between them. Due to its simplicity to calculate, being a simple scalar product, it is widely used in Natural Language Processing and Computer Vision.

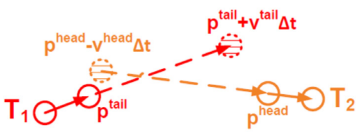
Motion Modeling

Objects can be identified across several frames by computing the similarity introduced before. However, only relying on their appearance might lead to poor performance for a MOT algorithm: similar-looking objects might be confused one with another. Indeed, one shall also be able to model the dynamic behavior of objects, providing an estimate of their future positions throughout sequential frames, thereby reducing the search space.

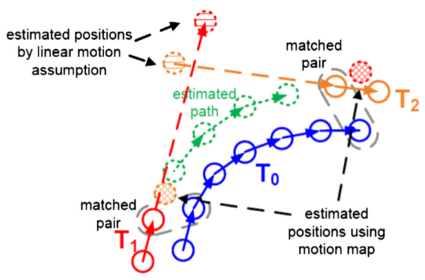
A common assumption, which suffices in most cases, is the **linear motion model** where the velocity is assumed to be *constant* (see Fig. 1.1a). There are several ways to achieve it: however, we will mention the exploited in the tracking algorithm we are going to use [2], which makes means of the so-called **Kalman Filters** [32] [33]. The assumption of **position smoothness** places a direct constraint on the observed and estimated positions: smoothness, in this case, can be modeled by fitting the **estimated** position using a *Gaussian distribution*, having as mean the **observed** position $\mathbf{p}$. Let us assume that we have a set of trajectories and a time interval Δt, and focus on a couple of them at different time instants, namely the *tail* of $\mathbf{T}_i$ and the *head* of $\mathbf{T}_j$. To assess the probability of them belonging to a unique trajectory, we compute the similarity assuming the linear motion model, that is, only considering the linear term which is proportional to velocity:

$$P_m(\mathbf{T}_i, \mathbf{T}_j) = \mathcal{N}\left(\mathbf{p}_i^{\text{tail}} + \mathbf{v}_i^{\text{F}}\Delta t \ , \ \mathbf{p}_j^{\text{head}} \ , \ \sum_j^B\right) * \mathcal{N}\left(\mathbf{p}_j^{\text{head}} + \mathbf{v}_j^{\text{B}}\Delta t \ , \ \mathbf{p}_i^{\text{tail}} \ , \ \sum_i^F\right) \tag{1.8}$$

where $\mathbf{p}$ denotes the position, and $\mathbf{v}^B$, $\mathbf{v}^F$, respectively, the backward and forward velocities. In other words: the displacement $\Delta\mathbf{p}$ is fitted to a Gaussian distribution with zero mean. However, in case such a distribution did not represent these displacements properly, one would drop the constraint on the distribution to be Gaussian and use the so-called **Unscented** Kalman filters [34].



(a) Linear motion model assumption.


estimated positions by linear motion assumption
matched pair
T_2
estimated path
T_0
matched pair
T_1
estimated positions using motion map

(b) Non linear motion model assumption, if the linear one were used, then the motion would not be predicted correctly.

Figure 1.1: Images taken from [9]

The natural extension to linear models is to consider also acceleration, that is, also the non-linear terms in the Taylor expansion for the position $\mathbf{p}$ (see Fig. 1.1b). For a small Δt we have that:

$$\mathbf{p}(t + \Delta t) \approx \mathbf{p} + \mathbf{v}\Delta t + \frac{1}{2}\mathbf{a}(\Delta t)^2 + \dots \tag{1.9}$$

Such an expansion would replace the linear terms like $\mathbf{p} + \mathbf{v}\Delta t$ in the equation (1.8), leading to a non-linear model. This is the idea underlying the so-called **Extended** Kalman filters [35].

1.2 Model performance metrics

We now want to introduce and discuss some of the metrics used to measure how good a tracker is. To compute such values, one has to compare the output returned by the detector being evaluated with the ground truth set provided. Clearly, the more similar these results are, according to some metrics chosen, the better the detector would perform. However, the *choice* of the **metrics** for such complex representations, namely scoring the similarity between sets of trajectories, is a well-defined problem. Having in mind the metrics **properties** and what they measure, we can determine which detectors to be preferred over another for a particular aspect. Finally, the use of a *shared* metrics allows communities to evaluate their tracking algorithms on common ground.

Let us now introduce what is the Jaccard Index [11] and how it can be used to measure the similarity of two generic sets, since it is fundamental to understand the metrics we will discuss later. After having introduced it, we will present the three types of metrics used and accepted by MOTChallenge [1], [15] [4].

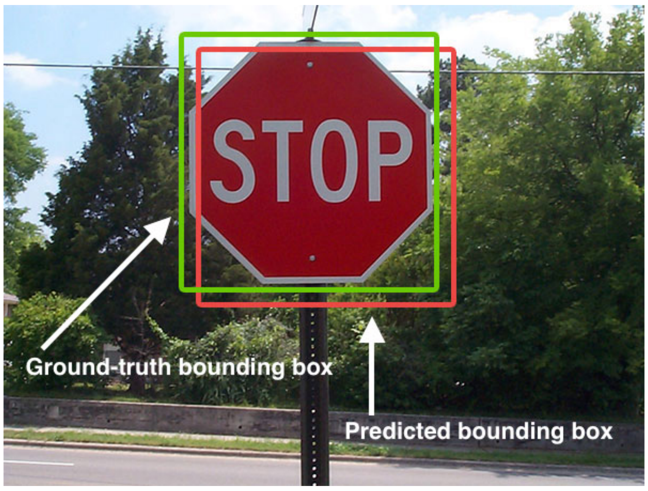


Figure 1.2: The goodness of a detector is computed by comparing the Predicted bounding box with the Ground-truth one. (image from [36])

Jaccard Index - IoU

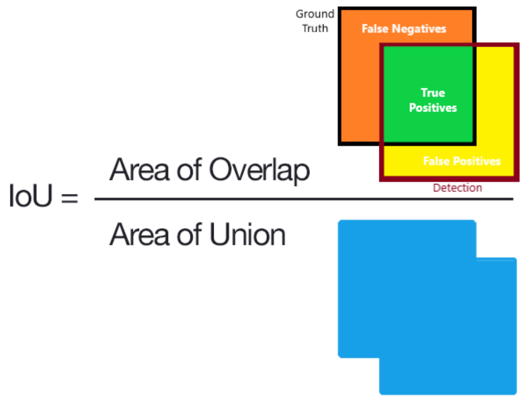
Generally, the Jaccard index is the most simple measure to assess the similarity between two generic sets. Formally, it is defined as:

$$\text{Jaccard index} = \frac{|TP|}{|TP| + |FN| + |FP|} \tag{1.10}$$

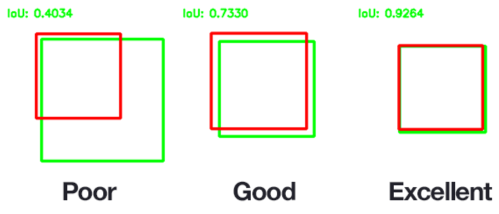
where:

- **TP** (True Positives): the set of detections shared between the ground truth and detection sets;
- **FN** (False Negatives): the set of objects which are present in the ground truth set, but have not been detected;
- **FP** (False Positives): the set of objects which are *not* present in the ground truth set, but are present in the detections;

For a visual representation of such a relation, one may want to look at Fig. 1.3a. Especially in the Computer Vision domain and for **detection** tasks, we often refer to the Jaccard Index index as **Intersection over Union** (IoU), when it denotes a *spatial* overlap between regions, which can take, for example, the form of bounding boxes (see Fig. 1.2) or masks. Therefore, a detection is considered as "*missed*" when the overlapping area is less than a certain threshold. A common value for this threshold is 0.5.



(a) Visual equation for the Jaccard Index (IOU).



(b) The closer the IoU to 1, namely the two areas overlapping almost perfectly, the better the detection.

Figure 1.3: Images taken and readapted from [36]

1.2.1 CLEAR MOT

Let us now provide an explanation of the CLEAR MOT [20] metrics for MOT algorithms, which provide event-based (i.e., frame by frame) tracking assessment. They consist basically of two different measures: Multi-Object Tracking Accuracy (**MOTA**) and Multi-Object Tracking Precision (**MOTP**). As the name suggests, the former measures how accurately an algorithm can track targets. A *bijective* mapping between *proposed* and *Ground Truth* detections is performed for every frame: in the case of matching, this would count as True Positive (TP). In contrast, any non-detected target is counted as a False Negative (FN), while any detection without a corresponding match is considered a False Positive (FP). As a common condition, if we define the spatial overlap between the proposed bounding box and the Ground Truth one with $\mathcal{S}$, the matching would occur whenever this value exceeds a certain threshold α, that is, $\mathcal{S} \geq \alpha$.

A tracker, however, must also be evaluated with respect to the re-identification task, that is, how well it can assign a unique id throughout a unique trajectory. The quantity that measures this ability is the number of **Identity Switches** ($IDSW$). We define that an IDSW has happened whenever a tracker *wrongly* swaps targets' identities, or whenever a lost (e.g., occluded) track for some time is reinitialised with a different id. Using the above formalism: an Identity Switch is a True Positive which has a proposed ID that is different from the proposed ID of the previous True Positive (i.e., that has the same Ground Truth ID). IDSWs only measure association errors compared to the single previous True Positive, whereas we do not consider errors when an *ID Transfer* occurs, namely, when the same proposed ID swaps to a different Ground Truth ID.

Finally, we can show explicitly how to compute **MOTA**:

$$\text{MOTA} = 1 - \frac{|FN| + |FP| + |IDSW|}{|G.T.\ Detections|} \tag{1.11}$$

Here, we can note that it is simply the **sum** of *detection* errors (FNs and FPs) and the number of *association* errors (IDSW). Note that MOTA $\in [0, 1]$: the closer to 1, the better the MOT algorithm. Finally, MOTA is structured similarly to the Jaccard Index. 1.10.

Let us now introduce the other metric, which explicitly measures the so-called **localisation** error. Recalling the spatial overlap $\mathcal{S}$, the Multi-Object Tracking Precision (**MOTP**) is:

$$\text{MOTP} = \frac{1}{|TP|}\sum_{TP}\mathcal{S} \tag{1.12}$$

That can be interpreted as the average similarity score on the set of True Positives. Consequently, a precise tracker has MOTP close to 1, given that MOTP $\in [0, 1]$. These metrics (1.11,1.12) can be defined either as we have shown here or, alternatively, in their complement to 1. In such a case, the problem would become a minimization problem with the ideal lower bound being 0. The main difference between MOTA and MOTP is that MOTA refers to the *accuracy* of detection and measures how good the detector and the assignemnt rule is, whereas *MOTP* measures its precision, that is, how much the detection boxes overlap with the ground truth ones. Training might involve trying to minimize both MOTA and MOTP simultaneously.

1.2.2 IDF1

Let us now introduce another metric used in the MOTChallenge. **IDF1** [21] is a bijective mapping at the level of **trajectories**, and not at the level of *detections* as before. We define IDentity True Positives (*IDTPs*) as the number of matches on the overlapping part (where $\mathcal{S} \geq \alpha$) of trajectories matched together. In contrast, the trajectories belonging to, respectively, the sets of ground truth and proposed routes that have not been matched are named IDentity False Negatives (*IDFNs*) and IDentity False Positives (*IDFP*s). Formally, we define the following scores:

$$\text{ID-Recall} = \frac{|IDTP|}{|IDTP| + |IDFN|} \tag{1.13}$$

$$\text{ID-Precision} = \frac{|IDTP|}{|IDTP| + |IDFP|} \tag{1.14}$$

$$\text{IDF1} = \frac{|IDTP|}{|IDTP| + 1/2|IDFN| + 1/2|IDFP|} \tag{1.15}$$

The matching of trajectories is usually performed with the Hungarian Algorithm and, if correctly done, it concurs to maximize IDF1, that is, minimize the number of unmatched trajectories coming from the Ground Truth (i.e., ideally tracking *all* the good ones), and the Proposed Trajectories sets (i.e., we do not track more trajectories than the ones effectively present). Note that these metrics *only* refer to correctly identify targets, while localization (i.e., precision) is not involved.

1.2.3 Mostly Tracked, Partially Tracked, Mostly Lost

The last metric we discuss is used to quantify how much of a ground-truth trajectory has been successfully tracked by the proposed tracking algorithm. This metric is not usually employed for ranking algorithms in official competitions, but can still provide a quick snapshot of how well a tracking system operates. By convention [37], we can classify the tracked trajectory according to:

- **MT** (Mostly Tracked): the target has been tracked for at least 80% of its lifespan.
- **PT** (Partially Tracked): the target has been tracked between 20% and 80% of its lifespan.
- **ML** (Mostly Lost): the target has been tracked less than 20% percent of its lifespan.

1.3 Physics-based applications

Using the tracker output, one may be able to model the influence of an object on other objects, that is, build a **mutual** motion model. It can be exploited, for example, when *visual information* is too chaotic due to the large number of objects present in the picture and some more abstract ”order” needs to be found. If the objects being tracked are pedestrians, which are actually the most common targets for Multi-Object Trackers, fluidodynamics formalism has been proved to be efficient. In fact, Henderson [38] had success comparing equation *Navier-Stokes* with empirical observations of pedestrian movements.

Being able to model the movement of the crowd, together with some hypotheses on how people behave in emergencies and normal situations, allows architects to design more efficient and safe escape paths in buildings. The first attempt to apply such approaches to designing constructions dates back to 1979 [39]. Instead considering **vehicles** trajectories, it is possible to build some traffic models [40], to design and build roads that are better tailored to drivers' needs, thus avoiding traffic jams, improving traffic planning quality, and building smarter cities.

1.3.1 Social Force models

We now focus on one of these approaches, which formally describes pedestrian motion flows in a simple way. In such models, whose name is **social force models**, we consider each individual motion to be *dependent* on other individuals [12] plus environmental factors. According to the behavior of other people, the pedestrian is considered an agent that adjusts its velocity, acceleration, and path according to the observation of other individuals: these are, in fact, observables we can extract from the output of a tracker.

In other words, we assume that a pedestrian is subject to **social forces**, which are not exerted *directly* by the environment and other pedestrians, but are a measure of motivations and other internal mind processes that lead a person to move toward a certain point [41]. They act as a response to certain environmental and sensorial stimuli, leading to *behavioral changes* whose aim is to maximize some utility function. We are going to discuss now *normal* situations, i.e., nonchaotic or emergential, since reactions are well predictable and rather automatic, hence more easily predictable.

Recalling that from a tracker output one can retrieve an individual α trajectory $\vec{r}_\alpha(t)$ and velocity $\vec{w}_\alpha(t)$ as a function of time, one can model pedestrian behavior according to the equation of motion $\frac{d\vec{w}_\alpha}{dt}$. These “forces” that contribute to determining the motion of a certain pedestrian α can be divided into different components.

First, an individual wants to reach its *destination* $\vec{r}^0_\alpha$ as comfortable as possible, that is, taking the shortest path. This consists of passing through certain *gates* located along the path that a pedestrian would cross if not disturbed. One of these paths consists of a polygon whose edges are $\vec{r}^1_\alpha, \vec{r}^2_\alpha ... \vec{r}^n_\alpha := \vec{r}^0_\alpha$. The **desired** direction $\vec{e}_\alpha(t)$, given the *actual* position at time t being $\vec{r}_\alpha(t)$, will take the form:

$$\vec{e}_\alpha(t) := \frac{\vec{r}^k_\alpha - \vec{r}_\alpha(t)}{||\vec{r}^k_\alpha - \vec{r}_\alpha(t)||} \tag{1.16}$$

Therefore, the pedestrian will steer to the next *nearest* point after having passed through a certain gate.

We assume, moreover, that there exists a certain **desired velocity** v^0_α that can be subjected to deviation due to the necessary deceleration or acceleration processes to avoid other people or a change in the state of the environment. We define the *relaxation* time to reach such a velocity as τ_α. Finally, one can model this acceleration term as follows:

$$\vec{F}^0_\alpha(\vec{v}_\alpha, v^0_\alpha \vec{e}_\alpha) := \frac{1}{\tau_\alpha}(v^0_\alpha \vec{e}_\alpha - \vec{v}_\alpha) \tag{1.17}$$

Second, clearly the motion of a pedestrian α is affected by other pedestrians, in particular by the *desired speed* and the *pedestrian density.* Therefore, we introduce the

private sphere of an invidividual: if two individuals get closer than such a distance, they feel uncomfortable. From a psychological approach, this can be interpreted as a *territorial effect.* Such an *effect* can be modeled as a **repulsive** potential originating from other pedestrians β:

$$\vec{f}_{\alpha\beta}(\vec{r}_{\alpha\beta}) := \nabla_{\vec{r}_{\alpha\beta}} V_{\alpha\beta}[b(\vec{r}_{\alpha\beta})] \tag{1.18}$$

It is reasonable to assume that such a *repulsive* potential $V_{\alpha\beta}(b)$ is a monotonic decreasing function of b, having equipotential lines of ellipsoidal shape, whose major axis is parallel to the pedestrian's direction of motion. In fact, a walker would need more space along its desired direction. Let us define b the semi-minor axis of the ellipse

$$2b := \sqrt{(||\vec{r}_{\alpha\beta}|| + ||\vec{r}_{\alpha\beta} - v_\beta \Delta t \; \vec{e}_\beta||)^2 - (v_\beta \Delta t)^2} \tag{1.19}$$

where $\vec{r}_{\alpha\beta} := \vec{r}_\alpha - \vec{r}_\beta$. Moreover, we assume that a step taken by β: $s_\beta = v_\beta \Delta t$ is of the order of a step width.

It is observed that people tend to keep distance from *borders* of buildings, walls, streets, sidewalks, etc. This effect of a generic border B can be modeled by a *repulsive effect* that can be described by:

$$\vec{F}_{\alpha B}(\vec{r}_{\alpha B}) := -\Delta_{\vec{r}_{\alpha B}} U_{\alpha B}(||\vec{r}_{\alpha B}||) \tag{1.20}$$

where $\vec{r}_{\alpha B}$, that is, the shortest distance between the pedestrian position and the closest point of the border, and we introduce a repulsive and monotonically decreasing potential $U_{\alpha B}(||\vec{r}_{\alpha B}||)$.

As a third contribution, one may want to consider the effect of attractions made by other people on a certain individual. These *attractive effects* $\vec{f}_{\alpha i}$ at places $\vec{r}_i$ are modeled by attractive, monotonically increasing potentials $W_{\alpha i}(||\vec{r}_{\alpha i}||, t)$ dependent on time [1]:

$$\vec{f}_{\alpha i}(||\vec{r}_{\alpha i}||, t) := -\Delta_{\vec{r}_{\alpha i}} W_{\alpha i}(||\vec{r}_{\alpha i}||, t) \tag{1.21}$$

We shall consider, in addition, that all the contributions introduced above hold only for situations that are perceived by the pedestrian along its direction $\vec{e}_\alpha(t)$ of motion: in other words, all that occurs located behind it is assumed to have a weaker influence quantified by a factor $c \in [0, 1]$. Hence, we introduce the *effective angle of sight* 2ϕ to module direction-dependent weights:

$$w(\vec{e}, \vec{f}) := \begin{cases} 1 & \text{if } \vec{e} \cdot \vec{f} \geq ||\vec{f}|| cos\phi \\ c & \text{otherwise} \end{cases} \tag{1.22}$$

[1] a possible source of interest can be, as an example, street musicians, whose interest eventually vanishes over time

Taking into account such weights, the repulsive and attractive forces are given by:

$$\vec{F}_{\alpha\beta}(\vec{e}_\alpha, \vec{r}_\alpha - \vec{r}_\beta) := w(\vec{e}_\alpha, -\vec{f}_{\alpha\beta})\vec{f}_{\alpha\beta}(\vec{r}_\alpha - \vec{r}_\beta) \tag{1.23}$$

$$\vec{F}_{\alpha i}(\vec{e}_\alpha, \vec{r}_\alpha - \vec{r}_i, t) := w(\vec{e}_\alpha, -\vec{f}_{\alpha i})\vec{f}_{\alpha i}(\vec{r}_\alpha - \vec{r}_i, t) \tag{1.24}$$

One can construct the equation for the total motivation of a pedestrian. Moreover, all the forces we have discussed so far act on a certain individual α at the same time instant t: it is reasonable to assume, then, that their total effect will be simply the sum of all their effects:

$$\vec{F}_\alpha(t) := \vec{F}^0_\alpha(t)(\vec{v}_\alpha, v^0_\alpha \vec{e}_\alpha) + \sum_\beta \vec{F}_{\alpha\beta}(\vec{e}_\alpha, \vec{r}_\alpha - \vec{r}_\beta) + \sum_B \vec{F}_{\alpha B}(\vec{r}_\alpha - \vec{r}^\alpha_B) + \sum_i \vec{F}_{\alpha i}(\vec{e}_\alpha, \vec{r}_\alpha - \vec{r}_i, t) \tag{1.25}$$

So, the *non-linearly coupled* **Langevin equations** for the **social force** model are:

$$\frac{d\vec{w}_\alpha}{dt} := \vec{F}_\alpha(t) + \text{fluctuations} \tag{1.26}$$

Where the fluctuation term is a stochastic term derived from ambiguous situations leading to different behavioral alternatives, such as avoiding an obstacle either on the left or on the right with no difference in the utility function. Alternatively, these fluctuations might be accidental or deliberate deviations from the expected behavior in a given situation.

Finally, to complete the model, we need to define a **relation** between the *actual velocity* $\vec{v}_\alpha(t)$ and the *desired velocity* $\vec{w}_\alpha(t)$. We assume that there exists a *maximal acceptable speed* $v^{\max}_\alpha$ for every pedestrian α, therefore, the motion occurs as:

$$\frac{d\vec{r}_\alpha}{dt} = \vec{v}_\alpha(t) := \vec{w}_\alpha(t) g\left(\frac{v^{\max}_\alpha}{||\vec{w}_\alpha||}\right) \tag{1.27}$$

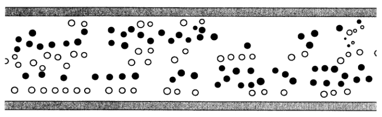
where:

$$g\left(\frac{v^{\max}_\alpha}{||\vec{w}_\alpha||}\right) := \begin{cases} 1 & \text{if } ||\vec{w}_\alpha|| \leq v^{\max}_\alpha \\ v^{\max}_\alpha / ||\vec{w}_\alpha|| & \text{otherwise} \end{cases} \tag{1.28}$$

Let us now briefly mention and discuss some interesting results of macroscopical behaviors found by [12] who actually run the simulation. It is assumed that *desired speeds* v^0 are distributed as Gaussian ($< v^0 >= 1.34; \sigma_{v^0} = 0.26$) [m/s], and the maximum speed is set to $v^{\max}_\alpha = 1.3 v^0_\alpha$. Repulsive potentials are assumed to be decreasing exponentially:

$$V_{\alpha\beta}(b) = V^0_{\alpha\beta} e^{-b/\sigma} \qquad U^0_{\alpha\beta}(||r_{\alpha B}||) = U^0_{\alpha B} e^{-||\vec{r}_{\alpha B}||/R} \tag{1.29}$$

with $V^0_{\alpha\beta} = 2.1m^2/s^{-2}$, $\sigma = 0.3$ and $U^0_{\alpha B} = 10m^2s^{-2}$ and $R = 0.2m$. For simplicity, attractive potentials were not considered. Finally, setting $\Delta t = 0.5$, and the relaxation time $\tau_\alpha = 0.5s$, while the field of view is $2\phi = 200°$ and influence $c = 0.5$.


(a) After a certain density, one can observe the emergence of macroscopic patterns, such as the formation of lanes with people going towards the same direction.

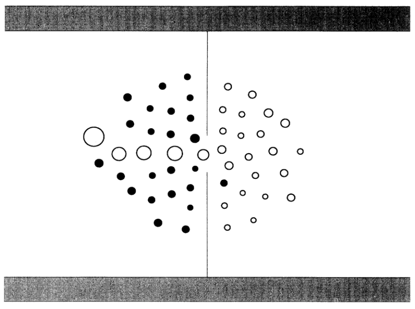

(b) If a narrow path is present, once a pedestrian has passed through, a queue of people following it will form.

Figure 1.4: The radius of a circle is proportional to the pedestrian's actual velocity, and the color refers to what side is located the individual's destination. Images taken from [12].

One can observe from Fig. 1.4a, that people going towards the same direction appear to organize in lanes having individuals sharing the same destinations: different colors refer to different directions. On the other hand, if there is a narrow door and two groups of people need to go through it (see Fig. 1.4b), when an individual has reached the other side, the other pedestrians intending to move in the same direction will be able to follow it. However, this flow may be stopped by the pressure of the incoming and opposing groups after some time, and again, once an individual has passed through the door, he will eventually be followed by others.

Headed Social Force Model

Another further improvement of such a model is the so-called **Headed Social Force Model** [22]. In such a model, the force $\vec{f_i}$ acting on an individual i according to the original Social Force Model is the contribution of two terms:

$$\vec{f_i} = \vec{f}^0_i + \vec{f}^e_i \tag{1.30}$$

Where the first model $\vec{f}^0_i$ describes long-term objectives, such as the reach of the desired destination, while the second model takes into account *temporal* and *short-term* corrections originated by the interactions between the individual and other people or the individual and the environment. However, unlike the SFM, the motion is generated by three main distinct terms. (see Fig. 1.5)

With reference to Figure 1.5, the first terms $\vec{u}^f_i$ and $\vec{u}^0_i$ are the ones driving the

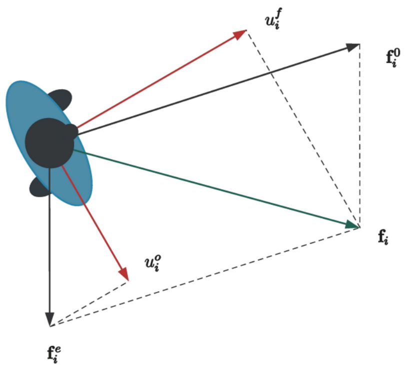
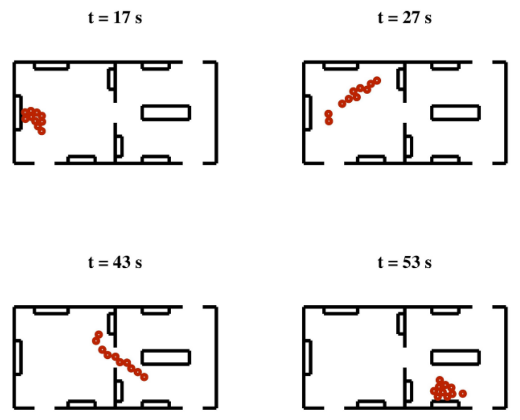


Figure 1.5: Decomposition of forces for the Headed Social Force Model. (image from [22])

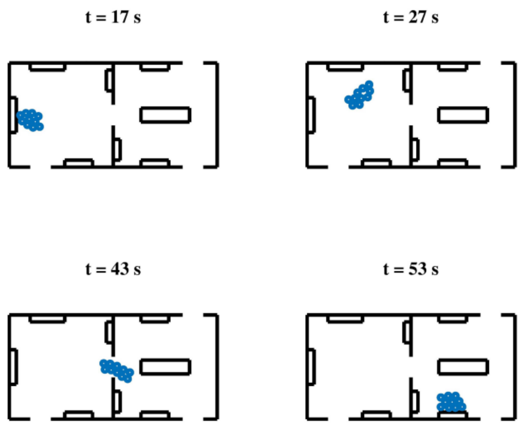
translational dynamics, and essentially they are the projection of, respectively, $\vec{f}_i^0$ and $\vec{f}_i^e$ along the direction of motion. Moreover, another term introduced is **torque** driving *rotational dynamics*, proportional to the projection of the term $\vec{f}_i^0$ perpendicularly with respect to the direction of motion. A final and additional term is introduced to ensure *group cohesion*, which is present when some people are moving together.

This model can be used to simulate people's behavior in different daily situations, such as when the metro train doors open and individuals need to give pace to others to get on or off the wagon. A similar result to the one shown in Fig. 1.4b is obtained. Another possible scenario that one would like to simulate is the one depicted in Fig. 1.6: where we are interested in understanding how a group behaves during a visit to the museum, depending on the presence of an attractive force between individuals of the same group (i.e., friends visiting the same museum).

This result, practically, would help in designing museum spaces in such situations where large gatherings of people have to be avoided, or in creating museums tailored to visitors' needs by achieving a realistic modeling of their behavior. The data used can be taken from surveillance cameras, thus tracking individuals and creating their trajectory. Another possible goal might be **scene understanding**, that is, figuring out whether people are behaving "normally" or if some emergency situation is happening.



(a) If a social attractive force is not considered, the group will be more disperse.



(b) If a social attraction force is included, as if one was visiting the museum with friends, the group would be more cohesive.

Figure 1.6: Scenario of a journey at the museum: people need to reach and see all the artworks that are located in different rooms, accessible through narrow doors. Images taken from [22]

Methodology: detectors, benchmarks and MOT algorithms

After we have introduced and discussed the MOT problem, and discussed several ways to solve it, in this chapter, we will focus on the most common networks used to perform detections, which is a fundamental step of the Detection-Based-Tracker paradigm. We will present an overview of the functioning of a one-stage detector prototype, namely the latest implementation of YOLOX [14] of the YOLO series [13] [42] [43] [43] [44]. Then, we will discuss how and according to what benchmark platforms different trackers can be compared [15], focusing particularly on *ByteTrack* [2]: the State-of-the-Art for trackers as of January 2022, according to the MOT benchmark [1] [15] [4]. Finally, another MOT algorithm is presented: *Fairmot* [3], which implements a head devoted to re-identification tasks and introduces a new "fair" approach for multitask training.

2.1 Detectors

In the context of MOT algorithms, object detectors play a particular role, as they have the task to locate and classify objects in images by highlighting the (possibly) rectangular-shaped bounding boxes containing them. In second place, after the bounding boxes have been detected, a *classification label* is assigned to them, together with a *confidence score* between 0 and 1: the closer to 1, the more confident the detector is about the output.

There are mainly two types of detectors: **one-stage** and **two-stage**, depending on how the object detection task is performed. Two-stage detectors follow the traditional pipeline of first locating objects (i.e., *region proposal*) and, in the second stage, classify them: this approach is, however, slower compared to their one-stage counterparts. Instead, **one-stage detectors** when fed with an image *directly* predict the bounding boxes over the input: the direct consequence is that the detection speed results increased. In fact according to [23], the YOLOv1 network, which is a one-stage detector, has an inference time of 45fps without batch processing using a Titan X GPU. Under

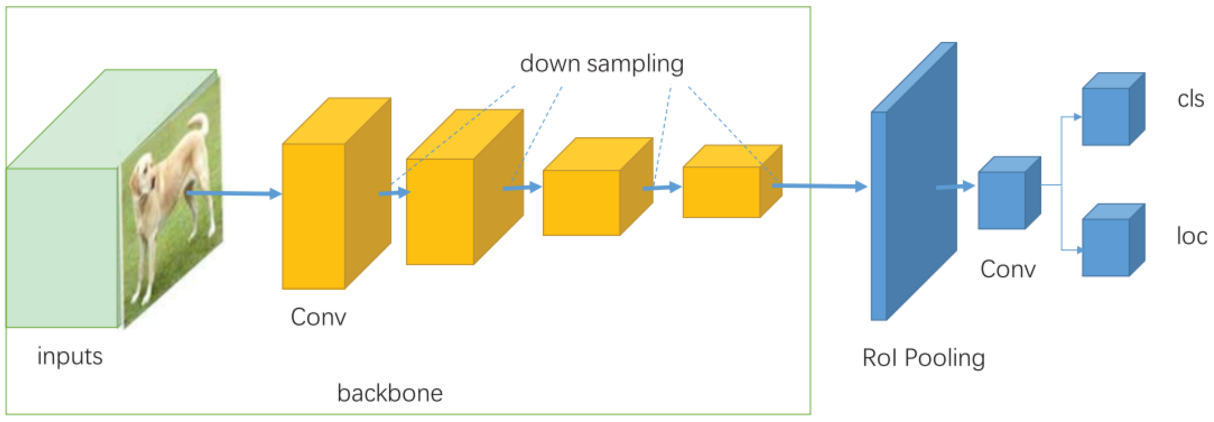


Figure 2.7: The basic architecture of a one-stage detector, which relies of a Backbone performing downsampling and the *head*, devoted to bounding box regression and classification. (image from [27]).

the same conditions and on the same GPU, the two-stage detectors Fast RCNN and Faster RCNN, respectively, have a processing time of 0.5 fps and 5 fps. The basic architecture of such detectors is depicted in Fig. 2.7. Since we want to take into account the velocity of our algorithm, we will now focus on *one-stage* detectors, in particular those belonging to the YOLO family.

Given the bounding box and the classification label outputs, one might want to train these networks in such a way that the absolute L_1 distance between the center of the *proposed* bounding box, and the bounding box dimensions[2] (x, y, w, h) are as close as possible to the ground truth bounding box. The overlap between the two can be computed through the Jaccard index, which for this case takes the form of the IoU distance of equation (1.10) (see Fig. 1.3a). The loss function that considers, respectively, the maximization IoU and the minimization of L_1 is known as **focal loss** [45]. Finally, the *total* loss function is obtained by adding the one resulting from the classification task, weighted for the regularization terms λ_i that balance the trade-off between the different loss terms:

$$\mathcal{L}_{tot} = \lambda_{cls}\mathcal{L}_{cls} + \lambda_{L_1}\mathcal{L}_{L_1} + \lambda_{IoU}\mathcal{L}_{IoU} \tag{2.31}$$

It is common to divide the network into two sub-networks, the **backbone** and the **head**, which have two completely different tasks. The backbone usually takes the input image and performs feature extractions, which, in turn, will be fed to the *head* detector. Often, it is common to use already trained backbones on datasets such as Microsoft COCO [46], to efficiently perform the downsampling of input images for

[2]A bounding box is uniquely defined by its top left (x, y) coordinates and its height h and width w. Alternatively, depending on the formalism, one can also use the coordinates of the top left (x, y) and bottom right (x', y') point coordinates.

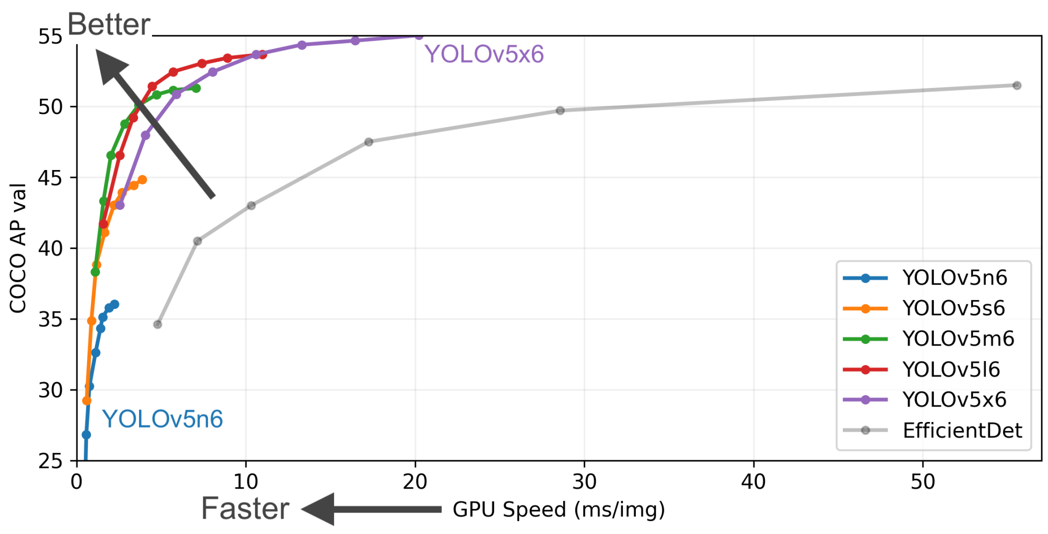


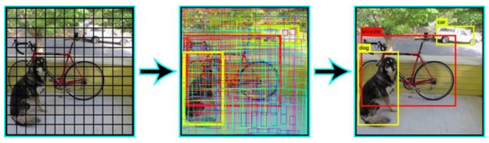
Figure 2.8: Performances in terms of precision and speed inference of the YOLOv5 detector. The amount of parameters follows the order of the list of networks in the legend in an incremental order (e.g., m has more parameters than n, but less than x). (image from [49]).

different classes of objects. For example, well-known backbones that can be employed in previous detectors are AlexNet [47] or ResNet [48]. Usually, the number of parameters plays a discriminant role in whether a detector can be used in a mobile application due to the limited space: for this reason, several versions of the networks are available as *full* or *light* models. Clearly, according to the benchmarks (e.g., see Fig. 2.8) the fewer the parameters, the lower the accuracy performance, but the faster the inference speed.

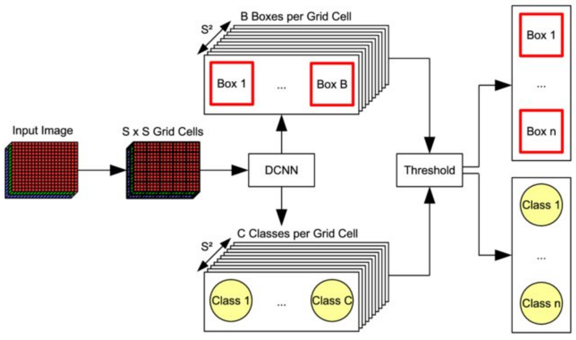
2.1.1 YOLO detectors family

Among the class of one-stage detectors, perhaps the most famous and important is the family of **YOLO** detectors (*You Only Look Once*). Historically, the first one was proposed by Redmon et al. [13] after Faster-RCNN [50]. The main reason why YOLO has become popular is that it allowed the community to detect objects in images in real time, for example using video streams produced by webcams, thus enlarging its possible fields of use.

Advancements in both training methodology and network designs, led to the evolution and growth of YOLO as a simple algorithm, generating an entire class of algorithms. For example, **YOLOv2** [42] added batch normalization during training thanks to a layer before each of the convolution layers. Furthermore, it increased the input image resolution allowed (448×448), which was doubled from the first generation YOLO (224×224) by modifying the backbone. In addition, it created a *reference* bounding box for every groups of bounding boxes related to the same object, as a result of their convolution. Another improvement is to *cluster* the *bounding boxes size* and *aspect*



(a) Visual processing of the functioning of a YOLO detector. Image from [51].



(b) Architecture of a YOLO detector: Image from [52].

Figure 2.9: Images depicting the functioning and of a YOLO detector. The input image is divided into a $S \times S$ grid: concurrently, for every grid cell, a bounding box regression and a classification tasks are performed and the results combined according to a certain threshold.

ratio by means of the k-means clustering method, to obtain a better prior. Moreover, the training approach changed: the network is re-trained starting from YOLOv1 using both higher and lower resolution features, by stacking adjacent features into different channels. In order to make the network robust to images of various sizes, every ten randomly selected batches it was fed with a new image of dimensions different from the usual input (448×448). According to the ablation study, all of these steps were shown to produce an improvement in the mean Accuracy Precision (mAP). The next model, that is **YOLOv3** [43], used as backbone Deep CNN Darknet 53 to extract features from images. To cope with more complex scenarios, it introduced multi-label classification with overlapping patterns. Finally, the three training feature maps that have different scales were used to predict the boundary box, allowing the detection of small objects.

Latest advancements in training techniques, architecture design, different loss functions, exploitation of different backbones and data processing in recent years have led to different generations of detectors of the YOLO family, such as YOLOv4 [53] or Analytics YOLOv5 [49] (whose accuracy and inference speed are presented as an example in Fig. 2.8). However, in the following section, we will discuss the State of the Art in detection tasks in 2021 introducing YOLOX [14], which, while taking inspiration from the third generation of YOLO, implements new training and architecture features.

2.2 YOLOX

Let us now focus on the detector employed by the algorithm we used for our work, namely **YOLOX**, which was first proposed by Zheng et al. [14] in 2021.

The results obtained (see Fig. 2.10) allow us to consider it as the State-of-the-Art detector both in terms of accuracy and speed for real-time tasks, as for late 2021, and its code is publicly available at [54]. The design of YOLOX, which uses as backbone

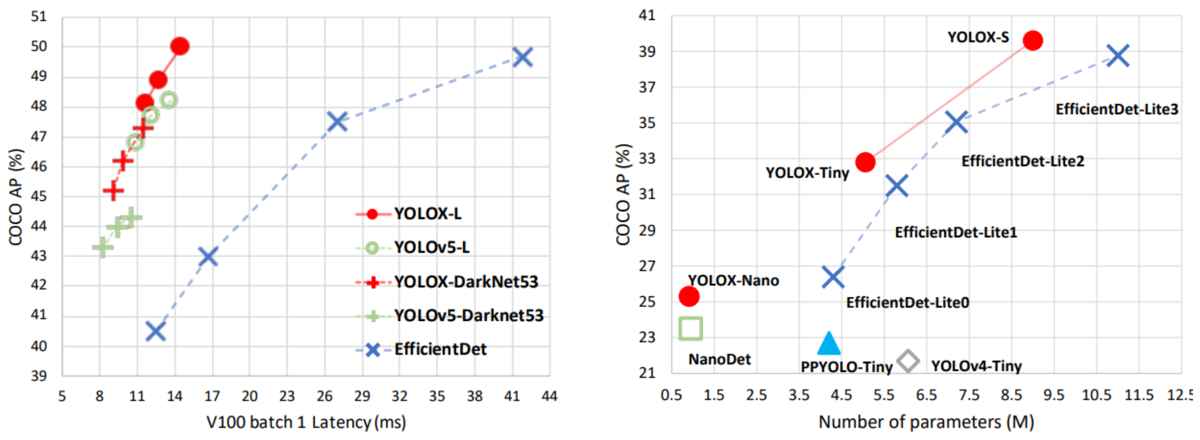


Figure 2.10: Results in terms of Speed-Accuracy (left) for a visual comparison of YOLOX with other detectors. The higher the Average Precision (AP) and the less the 1-batch Latency, the better performances. On the right it is shown the Size-Accuracy curve for *lite* models, that can be mounted on mobile devices due to the less memory needed for storing parameters. (image from [14]).

CSPDarkNet [55], takes inspiration from that of YOLOv3, which is still one of the most widely used one-stage detectors due to the limited amount of resources needed: since 2018 the research community has been able to create some industrial software support and apps that make use of YOLOv3.

With referral to YOLOX, the main improvements in detector design mainly deal with its architecture and training process. In particular, the conflict between *classification* and the regression tasks is addressed by **decoupling** the two *heads* devoted to *classification* and *regression* of the bounding boxes. We refer to **head** as the final part of the network, which takes as input the feature extracted by the backbone (and eventually a *feature pyramid*) and returns the bounding box, its labeling, and the confidence score. This decoupling affects the training convergence speed, which occurs faster, and is essential for the end-to-end version of YOLO.

Furthermore, YOLOX is designed to be a **anchor-free** detector, that is, the number of bounding boxes predicted for each location passes from 3 to 1: the head directly predicts the four parameters that uniquely define the bounding box, namely the coordinates on the top left and its width and height (see Fig. 2.11). This relieves the burden of conducting clustering analysis in advance to determine what anchor-boxes are the most important for the task, which is said to be highly domain specific. Moreover, heads not implementing the anchor-boxes mechanism show less complexity, thus reducing the number of parameters to be optimized: other design hyperparameters are also avoided to be heuristically tuned (e.g., the ones related to clustering). Other *strategies* employed in the YOLOX detector are related to **training** and augmentation techniques:

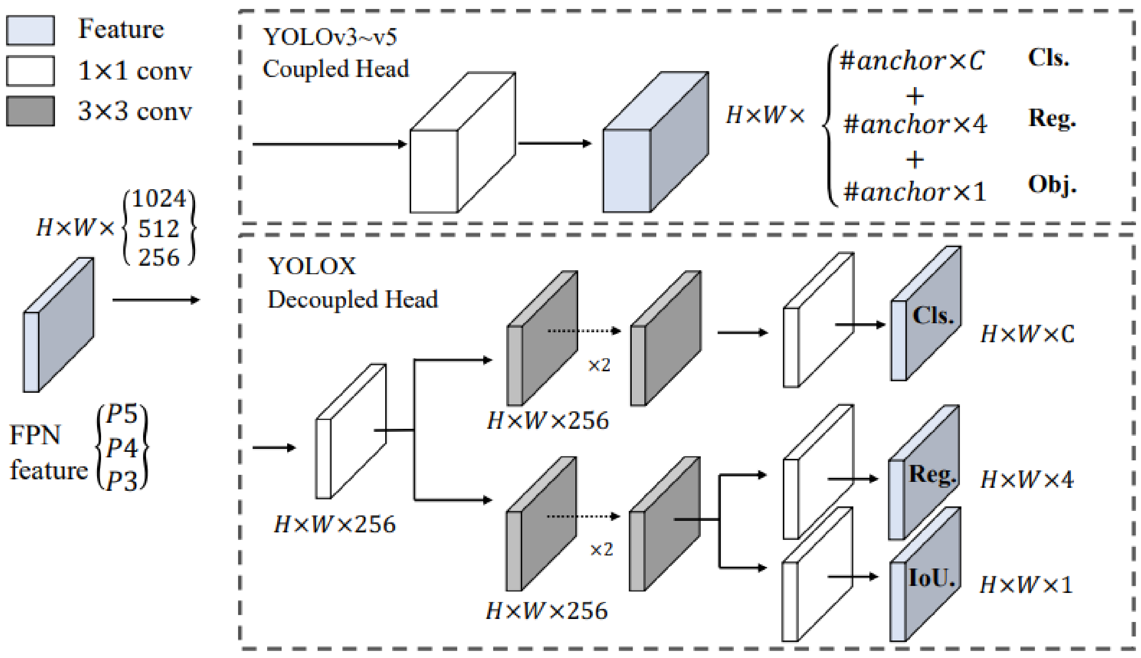


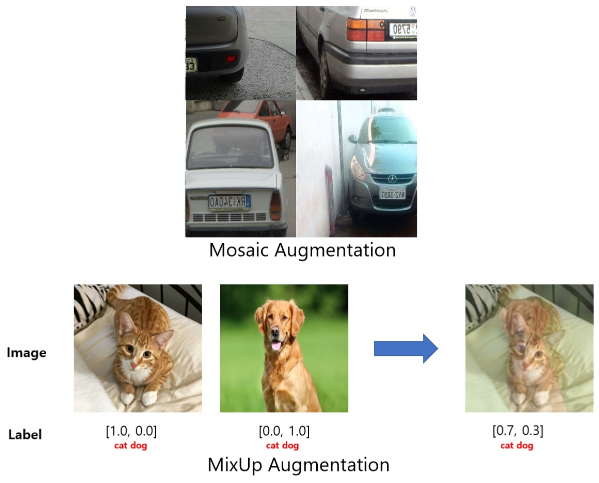
Figure 2.11: The heads, namely the parts of the detector devoted to classification and regressions are *decoupled* in YOLOX, while are not in other YOLO implementations starting from v3. Moreover, YOLOX produces as output "unique" detection boxes, defined by their top left coordinates, width and height. (image from [14]).

Mosaic, i.e., a mosaic is created by stitching different images from the training set, and *MixUp*, that is, weighted combinations of random image pairs from the training data (see Fig. 2.12a) are generated. The detector, due to the strong augmentation techniques, is decided to be trained from scratch: it is not more beneficial to make use of the pretraining on *ImageNet* to specialize the network to the image domain.

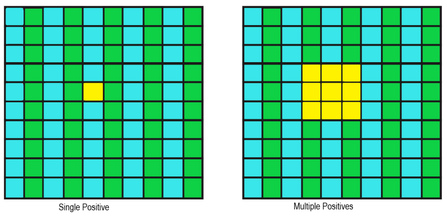
Another technique implemented is to consider as *positive* samples not only the centers of the object-related bounding boxes, but also the adjacent cell grids, in such a way that a grid 3×3 is considered a positive sample. The authors refer to this approach as **Multiple Positive** (see Fig. 2.12b).

The last important advancement in object detection concerns the development of an *advanced label assignment* technique, namely **SimOTA**. Usually, anchor-free methods directly relate the center of the bounding box region of any ground-truth object to the corresponding positives. To take advantage of all the properties of the object for positive and negative assignment, dynamic assignment methods have been implemented, such as *SimOTA*: it first calculates the degree of pairwise matching for every prediction-ground truth pair. As an example, this cost c_{ij} between the ground truth gt_i and the prediction p_j can be calculated as:

$$c_{ij} = L_{ij}^{cls} + \lambda L_{ij}^{reg} \qquad (2.32)$$



(a) Data augmentation techniques. Images from [56] and [57].



(b) Instead of considering as positive sample only the center location for each object, a larger grid (3×3) is considered as positive sample.

with λ balancing the two terms and L_{ij}^{cls} L_{ij}^{reg} being, respectively, the classification and regression losses. Finally, for every gt_i, the top k predictions with the lowest cost within a fixed center region are considered positive samples, whereas the other grid predictions are negatives. The value of k varies for different ground truths and is automatically returned by the $SimOTA$ algorithm itself. We have now shown how, starting from a well-known architecture, namely YOLOv3, the new State-of-the-Art one-stage detector for the YOLO family has been implemented. It employs many of the advancements developed by the research community in the last years, such as anchor-free paradigm or the use of decoupled heads for accomplishing the two different tasks of classification and bounding box regression.

2.3 The MOT challenges

After having presented the detector we are going to use in our work, let us now show according to what metrics and benchmarks we will consider ByteTrack, the State of the Art among Multi-Object Tracker algorithms. In the recent past, it is worth noting that the Computer Vision community relied on several centralized benchmarks for many tasks, including object detection, pedestrian detection, 3D reconstruction, optical flow, short-term tracking of single objects, and stereo estimation. Providing a common method to evaluate newly proposed techniques, they have proven to be extremely helpful in advance of State of the Art research in the respective research fields. However, there used to be a **lack** of a **shared benchmark** technique accepted by the research community, except for the well-known PETS [58] (*Performance Evaluation of Tracking and Surveillance*) which primarily addressed surveillance applications. Typically, the proposed methods were specifically tuned for this dataset, which resulted in overfitting in real-world tasks.

Furthermore, being able to **properly evaluate** *single-camera MOT results* is **not trivial**: there may be little confusion in the definition of the ground truth, such as how to handle partially occluded/cropped/reflected objects and how to properly define their bounding boxes. Additionally, before the first version of MOTChallenge, there was no commonly accepted and non-parametric metric to evaluate the goodness of a MOT algorithm. Finally, it was difficult due to the lack of predefined test and training data, making it difficult to fairly compare the proposed methods.

To address all these issues, **in 2014**, the **first MOTChallenge** [1] benchmark was released, which showed the following three main components. It consisted of a recollection of publicly available new datasets, a centralized evaluation method, and, last, an infrastructure that allowed the community to present new datasets, propose evaluation metrics, and share new annotations to improve ground truth quality. Due to the propulsion provided by the research community, the consistency of the annotations over the sequences was improved by computing the *degree of occlusion* and *cropping* of bounding boxes, resulting in the release of MOT17Challenge [15]. Moreover, in the latter, three sets of public detections of three different **object detectors** are provided: participants in the competition are required to evaluate their proposed tracker on *all* these detection sets, hence taking the *overall average*. A *good tracker* is **robust**, that is, it should not (ideally) depend on *quality* of *detections*.

The latest of these challenges is the MOT20Challenge, which shows some differences, especially in the labeling of ground truth boxes: they denote whether a given target is a standing/sitting pedestrian or any other object (e.g., a vehicle, a bike, etc.), although during the evaluation part only the moving objects and **pedestrian** class are considered. This is done since some trackers might rely on motion cues; therefore, one does not want to penalize it if not tracking a sitting pedestrian, or if following a bike-vehicle trajectory, which might not be the task the detector had been trained for. Generally, together with the release of new MOTChallenges, the complexity of the scenarios shown by the training and evaluation datasets has increased. More specifically, the crowd density in the sequences has become larger, several viewpoints were considered, different frame rates, and light conditions were tested (e.g., some video sequences at night). Typical frames from the data set are shown in Fig. 2.13

The training and evaluation datasets consist of sequences recorded by **single static cameras**: the images are converted to JPEG format. Detection and annotation files are simple CSV (Comma Separated Value) files. A general *detection* (DET) and a *ground truth* (GT) **annotations files** exhibit the following structure as shown in Table 2.1.

It is worth mentioning that, following the MOTChallenge approach and idea, new standardized benchmarks were released to tackle image segmentation (e.g., MOTS), the tracking of *any* object (e.g., TAO), autonomous driving related tasks (e.g., KITTI),

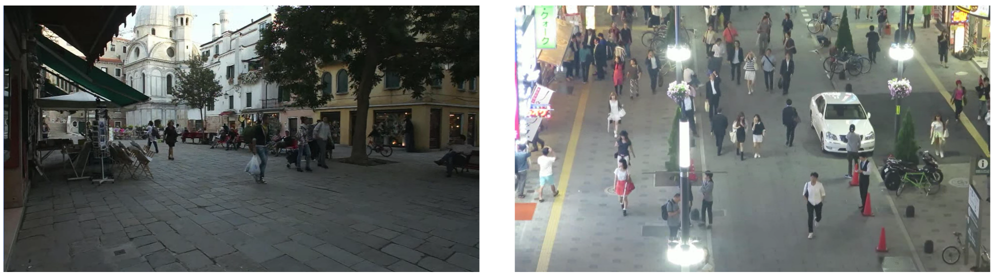


Figure 2.13: Frames coming from a typical MOT17Challenge dataset. (image from [15]).

Position	Name	Description
1	Frame number	Number denoting the frame at which the target is present
2	Target id number	**DET**: fixed to -1. **GT**: Unique ID for pedestrian trajectories.
3	Bounding Box left	X-axis coordinate of the top-left corner of the bounding box.
4	Bounding Box top	Y-axis coordinate of the top-left corner of the bounding box.
5	Bounding Box width	Width in pixels of the bounding box.
6	Bounding Box height	Height in pixels of the bounding box.
7	Confidence Score	**DET**: the confidence of the detector for the target being a pedestrian **GT**: ’1’ if entry has to be considered, ’0’ otherwise. It acts as a flag.
8	Class	**GT**: denotes the type of object annotated. (’1’ for pedestrians)
9	Visibility	**GT**: visibility ratio due to cropping or partial occlusion.

Table 2.1: Typical annotations file structure for MOTChallenge.

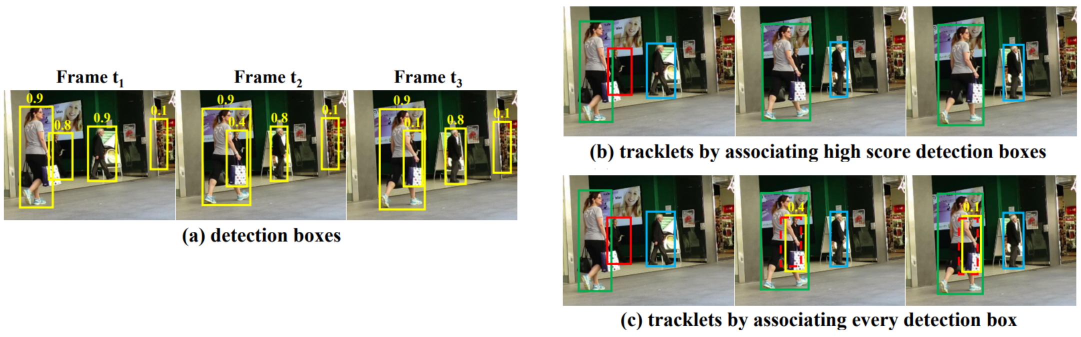

Frame t_1
Frame t_2
Frame t_3
0.9
0.8
0.9
0.1
0.4
0.8
(a) detection boxes
(b) tracklets by associating high score detection boxes
(c) tracklets by associating every detection box

Figure 2.14: An example of associating every detection boxes. *(a)* Output of a detectors, namely detection boxes and their confidence scores. *(b)* Association *only* of detection boxes *above* a certain *threshold* (i.e., 0.5). Same colors represent the same identity. *(c)* Association of detection boxes provided by ByteTrack algorithm. Dashed boxes represent the predicted boxes thanks to the Kalman Filter. The remaining two score detection boxes are matched to the past tracklets. (image readapted from [2]).

vehicle tracking (e.g., DETRAC), etc.

2.4 ByteTrack

According to the results claimed by Zhang et al. [2] in late 2021, the new **ByteTrack** has become the state-of-the-art tracker algorithm when provided with additional training data. It relies on the **Tracking-by-Detection** paradigm, that is, in a first step perform the detection of objects for subsequent frames and, secondly, perform their association into a *tracklet*. Therefore, the part of the network devoted to detection is fundamental and requires high performance: for this reason, the latest YOLOX, presented in the previous pages, has been used.
The main idea behind ByteTrack is that also the *low confidence* detection boxes are important; therefore, they should not be eliminated. In fact, they sometimes might indicate the existence of a *partially occluded* object or an object confused with the background, leading to poor tracking performance in terms of missing detections and fragmented trajectories. This is what is represented in Fig. 2.14: when an occlusion occurs, the confidence of the bounding box decreases, thus being thresholded out from detector output and losing the tracklet. On the contrary, exploiting low score detection boxes allows to assign them to tracklets using, for example, the position and motion information. However, visual information can also be exploited in such a way as to favor long-term recovers.

Once the detection boxes have been proposed by the detector, the tracker needs to compute the similarity between tracklets and these boxes, finally matching them according to some similarity metrics. Some useful cues for association are **location**, **motion**, and **appearance**. Depending on the algorithm, this information can be combined in different ways. For example, **ByteTrack** relies only on *location* and *motion*, completely neglecting the appearance. Having computed the **similarity**, one has to perform the matching according to the similarity score. This can be done, for example, with the mean of the Hungarian Algorithm.

ByteTrack, after performing the detection, divides the bounding boxes according to a *high* and *low* score threshold, thus obtaining two different sets. Initially, the **high score** detection boxes will be **matched to tracklets**. However, some tracklets will remain unmatched due to some changes in detection box size or eventual occlusions. Finally, the remaining detection boxes (low score) and these unmatched tracklets are associated: using such an approach, we can recover objects with low score detection confidence, hence filtering out the background.

Let us now present the **algorithm** in pseudocode (see Algorithm 1) implemented by ByteTrack. It accepts as **inputs** a *sequence of frames* $\mathcal{V}$, an *object detector* DET, the *Kalman Filter* KF to predict the future position of detection boxes, and three parameters: respectively τ_{high}, τ_{low} being the *detection* thresholds, while ϵ being the *tracking* threshold. For a matter of space, in the pseudocode the **track** rebirth, which is devoted to preserve the identity of the tracks in case of *long range* associations, is not listed: if a tracklet remains unmatched after the two associations, it is put in the set of *lost* trajectories $\mathcal{T}_{lost}$. For every frame, detection boxes are tried to be assigned to these tracks, however, if a given trajectory exists in such a set for a certain number of frames (usually 30), it is definitely removed. Detection thresholds are those that define the values according to which detection boxes can be assigned to the *high* or *low* score sets. On the contrary, the *tracking* threshold is the minimal value for which two consecutive and unmatched *high* score detection boxes are assigned to a *new* trajectory. Note that, as a similarity metric, it is not mandatory to use IoU, but others can also be employed that, for example, account for visual cues also. However, it is reasonable to not use appearance information, since low-score detection boxes usually contain severe occlusions, the image is blurred, or simply are detections that incorrectly refer to background.

The state-of-the-art performance for ByteTrack is obtained by equipping it with YOLOX as detector, whose initial weights have been pre-trained on the MS-COCO dataset. The default values for, respectively, τ_{high}, τ_{low}, and ϵ are 0.6, 0.1, and 0.7. The matching is rejected if IoU is less than 0.2, and the buffer size for the lost tracklets is 30 frames. Some visual results of ByteTrack in the MOT-17 dataset are shown in Fig. 2.15.

Algorithm 1 ByteTrack pseudocode. Track rebirth, which stores a buffer of untracked tracklets for some instants, is not shown for simplicity. (taken from [2])

```
Input: Video sequence $\mathcal{V}$, Detector DET, Kalman Filter KF, detection score threshold
    $\tau_{high}$, $\tau_{low}$, and tracking score $\epsilon$ threshold
Output: Tracks $\mathcal{T}$ of subsequent bounding boxes $\mathcal{D}$ with fixed id.
 1: INITIALIZATION: $\mathcal{T} \leftarrow \emptyset$
 2: for frame $f_k$ in $\mathcal{V}$ do
 3:     $\mathcal{D}_k \leftarrow \mathrm{DET}(f_k)$
 4:     $\mathcal{D}_{high}, \mathcal{D}_{low} \leftarrow \emptyset$
 5:     for $d$ in $\mathcal{D}_k$ do
 6:         /* Find detection boxes and their score */
 7:         if d.score $> \tau_{high}$ then
 8:             $\mathcal{D}_{high} \leftarrow \mathcal{D}_{high} \cup \{d\}$
 9:         end if
10:         if d.score $> \tau_{low}$ then
11:             $\mathcal{D}_{low} \leftarrow \mathcal{D}_{low} \cup \{d\}$
12:         end if
13:     end for
14:
15:     /* Predict the new position of detBoxes */
16:     for $t$ in $\mathcal{T}$ do
17:         $t \leftarrow KF(t)$
18:     end for
19:
20:     /* High score detBoxes associations*/
21:     Associate $\mathcal{T}$ and $\mathcal{D}_{high}$ using $IoU$ distance
22:     $\mathcal{D}_{remain} \leftarrow$ not matched boxes in $\mathcal{D}_{high}$
23:     $\mathcal{T}_{remain} \leftarrow$ not matched tracks in $\mathcal{T}$
24:
25:     /* Low score detBoxes associations*/
26:     Associate $\mathcal{T}_{remain}$ and $\mathcal{D}_{low}$ using $IoU$ distance
27:     $\mathcal{T}_{remain'} \leftarrow$ not matched tracks in $\mathcal{T}_{remain}$
28:
29:     /* delete unmatched tracks */
30:     $\mathcal{T} \leftarrow \mathcal{T}\ \mathcal{T}_{remain'}$
31:
32:     /* Initialize new tracks */
33:     for $d$ in $\mathcal{D}_{remain}$ do
34:         if d.score $d > \epsilon$ then
35:             $\mathcal{T} \leftarrow \mathcal{T} \cup \{d\}$
36:         end if
37:     end for
38:
39: end for
40: Return $\mathcal{T}$
```

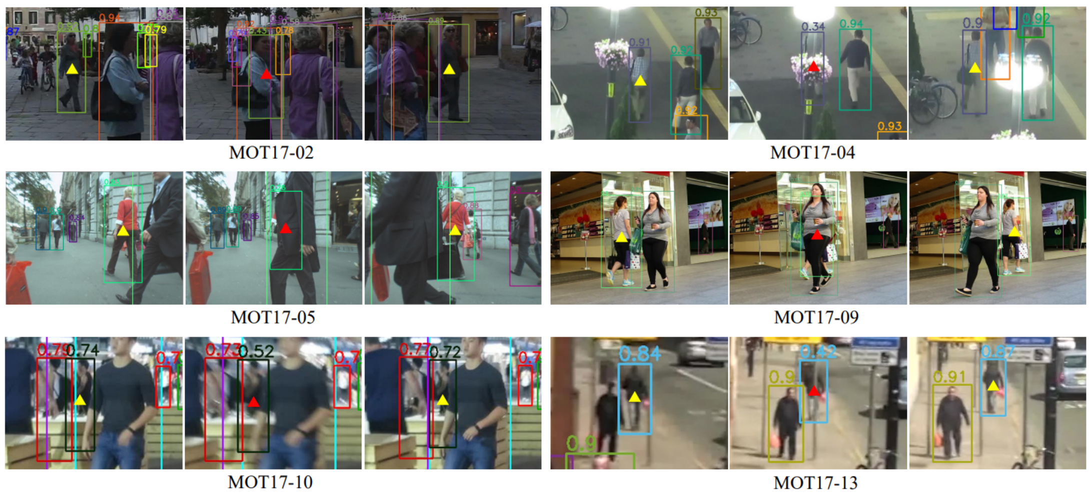


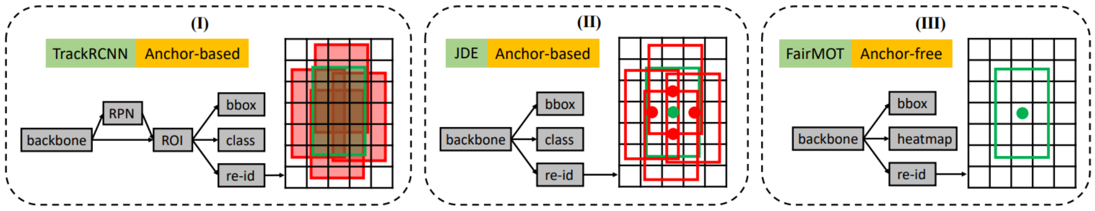
Figure 2.15: Visualization results of ByteTrack coming from MOT17 validation set. Note as also in case of occlusion or motion blur the algorithm behaves correctly. Yellow triangles denotes high score detection boxes, whereas red triangles low score and boxes with the same color refer to a unique trajectory. (image from [2]).

2.5 FairMOT

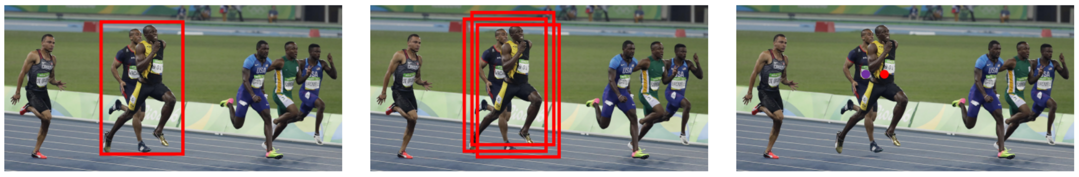
In the following section, we discuss the main ideas behind another MOT algorithm released in 2020: **FairMOT**. According to Zhang et al. [3], Multi-Object Tracking can be formulated as a *multi-task* learning of **object detection** and **re-ID** in a single network, thus allowing joint optimization of the two assignments. However, it is found that, if not trained properly, the two heads (see Fig. 2.17) devoted to these different tasks tend to compete, leading to biased and poor results. Before the release of *FairMOT*, the **detection** of objects through bounding boxes and **re-identification** (abbr. ReID), namely their *association* to trajectories while extracting features from the image regions corresponding to each bounding box, were treated as two different tasks. This produced inaccurate results. An advantage of treating them separately is that different and specifically tailored models can be implemented for these two tasks without making any compromise. However, the processing turns out to be very slow and not useful for real-time applications.

There are three main reasons why one-shot trackers get degraded association performance:

- **Anchors**: they were originally designed for object detection but are not suitable for extracting reID features. Indeed, anchor-based one-shot trackers overlook the reidentification task: when the detections are not correct, it turns out to be useless to extract reID features from the bounding box. This results in favoring the detection task at the expense of the reID. Moreover, if several detection boxes



(a) *Comparison of the existing one-shot trackers and FairMOT*



(b) *One anchor contains multiple identities* (c) *Multiple anchors response for one identity* (d) *One point for one identity*

Figure 2.16: (a) TrackRCNN and JDE are *anchor-based* detectors, namely they propose more than one bounding boxes (in red), while the correct one being the green one. The feature extraction process occurs at a second stage, at might occur using a whole region (TrackRCNN), using several points (JDE) or a single point in the center (*FairMOT*). (b) The red anchor contains two different instances (i.e., people) of the same class (i.e., person). When extracting ReID features, they refer to two conflicting istances. (c) Three red anchors referring to different image patches predict the same identity. (d) FairMOT extracts reID features only at the center of the two objects (image from [3]).

are proposed (as in anchor-based detectors), they might refer to a single identity, and a single anchor might refer to multiple identities, especially in crowded scenes. (see Fig. 2.16)

- **Feature sharing between detection and reID**: since they are two different tasks, they need different features. In fact, the reID task needs more *low-level features* to discriminate between instances of the same class, while the detection features need to be *similar* for different instances to provide a proper classification.

- **Feature dimension**: it is found empirically that learning low-dimensional reID features for joint detection and reID allows better results to be achieved in terms of tracking accuracy and efficiency.

To address these issues, *detection* and *reID* tasks are treated at the same level, thus avoiding the past paradigm of "detection first, reID secondary". The proposed solution is the one depicted in Fig. 2.17, which consists of two *homogeneous* branches devoted to **detection** and **reID feature extraction** tasks, and which are trained at the same time. In particular, the detection branch relies on *anchor-free* styles, which estimate the objects' center and sizes, represented as position-aware measurement maps. Whereas the reID branch attempts to extract *pixel features* to characterize the object in a *single*

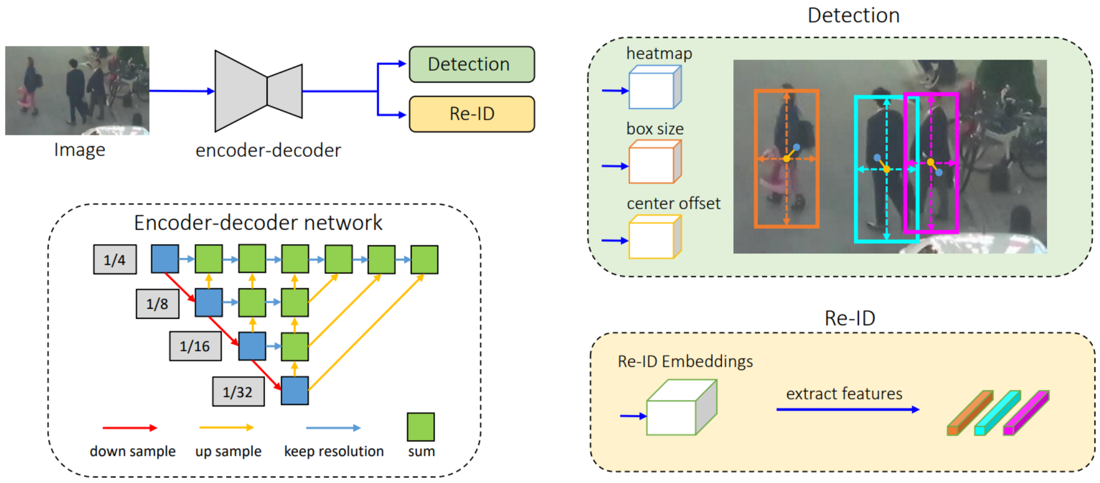


Figure 2.17: FairMOT network architecture. The input image is fed to an encoder-decoder network to extract its features at different levels of abstraction (*stride* = 4). Then, two heads are added, whose tasks are to *detect* objects and to *extract re-ID* features. The features at the *center* of the objects are the ones exploited for tracking. (image from [3]).

point (i.e., its **center**), therefore FairMOT can perform long-range association thanks to appearance features, thus proposing a new way to handle occlusion cases.

Let us now discuss how we can effectively balance the losses of two different tasks that in our case are *detection* and *reID features extraction*. We denote the detection loss by $\mathcal{L}_{det}$ and the Re-ID loss by $\mathcal{L}_{ReID}$. Given the center $(\tilde{c}_x^i, \tilde{c}_y^i)$ of the i-th bounding box, the re-ID feature vector is $\mathbf{E}_{\tilde{c}_x^i, \tilde{c}_y^i}$. This is mapped using the *softmax* operation to a class distribution vector $\mathbf{P} = \{\mathbf{p}(k), k \in [1, K]\}$. Let $\mathbf{L}^i(k)$ be the *one-hot* representation of the ground-truth labels class, with K being the total number of identities in the training data. Finally, we can define **re-ID loss** as:

$$\mathcal{L}_{reID} = -\sum_{i=1}^{N}\sum_{k=1}^{K} \mathbf{L}^i(k) \; \log(\mathbf{p}(k)) \tag{2.33}$$

In order to **train** *jointly* the two branches devoted to detection and re-ID, the approach proposed by Kendall et al. [59] is used: **uncertainty loss** is introduced to automatically balance the losses for the two tasks:

$$\mathcal{L}_{total} = \frac{1}{2}\left(\frac{1}{e^{w_1}}\,\mathcal{L}_{det} + \frac{1}{e^{w_2}}\,\mathcal{L}_{reID} + w_1 + w_2\right) \tag{2.34}$$

where w_1, w_2 are *learnable* parameters to achieve such a balance. Thanks to this approach, FairMOT is considered a valid tool to perform the MOT task and ranks among the top 5 in both the MOT20 and MOT17 challenges.

Experiments: MOT in sport analytics

Let us now discuss a particular application for the algorithms we have just presented, which is different from the usual surveillance tasks. In fact, we want to build a tracker that is capable of detecting soccer players on the field and reconstructing their trajectories in space, given some videos recorded by a *single static camera.* The dataset we are going to use is an open dataset made openly available by Pettersen et al. [5] It consists of elite soccer player movements captured by an XYZ sensor and the corresponding videos.

In this chapter, we will show how pre-processing has been performed to reduce the radial distortion of the image according to [60], the tools used to accomplish this task, and how we design the algorithm to associate the tracklets in the pitch coordinates found by the homography [29], with the players' trajectories. These trajectories are provided by XYZ sensors worn by players and, via a simple transformation, can mapped into pitch coordinates. Considering the data of the sensors as ground-truth, we can obtain a hint of the goodness of our tracking and assignation algorithm.

3.1 Dataset description

There are several open datasets related to **Multi-Object Tracking**, but they treat mainly as targets pedestrians or vehicles [58] [7] [15]. We are interested in finding an application different from the usual ones, that is, **track players on the football pitch**, in such a way that using a single fixed camera we are able to retrieve players' trajectories in the *real world.* In this case, we will treat the data coming from sensors as our ground truth.

The dataset we will use consists of body sensor traces and the corresponding videos from several professional soccer games captured in late 2013 at the Alfheim Stadium in Tromsø, Norway. In particular, we consider the videos coming from a single game, namely, the one played on November 3rd, 2013 between Tromsø IL and Strømsgodset

IF (Norway).

3.1.1 Sensor data

Camera	Basler acA1300-30gc	Basler acA2000-50gc
Resolution	1280 × 960	1920 × 1080
Frame rate	30 fps	25 fps
Lens model	3.5 mm *Kowa-LM4NCL*	8 mm *Azure-0814M5M*
Use	single wide-angle videos (1280 × 960)	stitched panoramic video (4450 × 2000)

Table 3.2: Cameras technical specifications for the videos available in the dataset [5]. We will use the camera whose technical specifications are written in the first column.

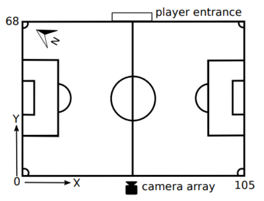
The sensor data we will use are recorded using the ZXY system and are obtained by sensor belts worn by players on their lower torso for the entire duration of the match. The signal is received by 11 stationary radios mounted on poles or on the tribune roof surrounding the stadium. Every receiver has approximately 90° field of view, in such a way as to avoid occlusions and/or signal blocking, since the pitch presents overlapping zones. The belt is furnished with an **accelerometer** in ZYX directions, as well as a *gyroscope*, a *heart-rate sensor* and a *compass*. Due to privacy concerns, heart rate estimation is not provided, and players are anonymized: the *tag* denoting every sensor is unique, but different from the actual shirt number. The authors claim that every attempt to recognize players is forbidden; therefore, we will only reconstruct their trajectories on the ground.

The data provided by the sensor are sampled at a frequency of $20Hz$, and are available in several CSV (Comma Separate Values) files; we are going to use the ones with interpolated timestamps, that is, the one that has already been preprocessed by the authors of the dataset. Their pre-processing has mainly involved resampling the entries according to a common timestamp, while keeping the sensor resolution of $20Hz$, to simplify the operation of queries. A sample of such a dataset is that in Fig. 3.20.

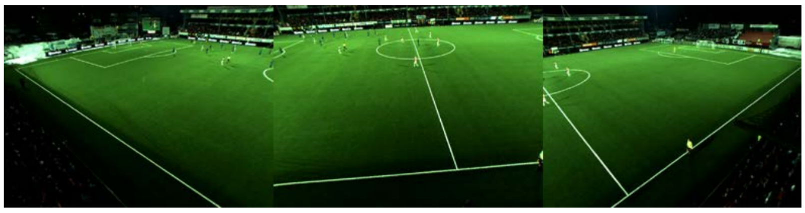
The logs that we will handle consist of the following information[3]:

- **timestamp** *(string)*: local Central European Time (CET) time encoded as ISO-8601 format string;
- **tag_id** *(int)*: the unique identifier for the sensor;
- **x_pos** *(float)* $[m]$: x-direction player's relative position in the field. Valid values are such that $0 < x < 105$. The authors claim a precision of 1 meter for the sensor;

[3]specifically for *x_pos* and *y_pos*, one the origin is the one in Fig. 3.18a



(a) Location of the camera array in the pitch XYZ coordinates.



(b) The view from the three cameras array. In this work we will use the central one, since we want to track what happens in the middle of the pitch.

Figure 3.18: Images from [5].

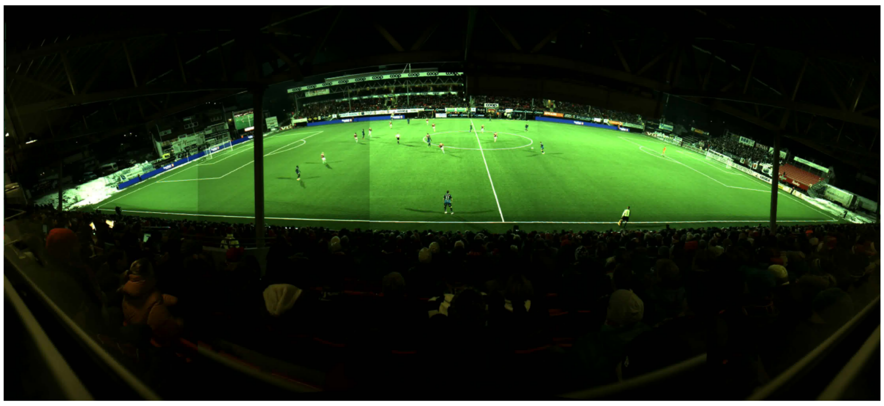
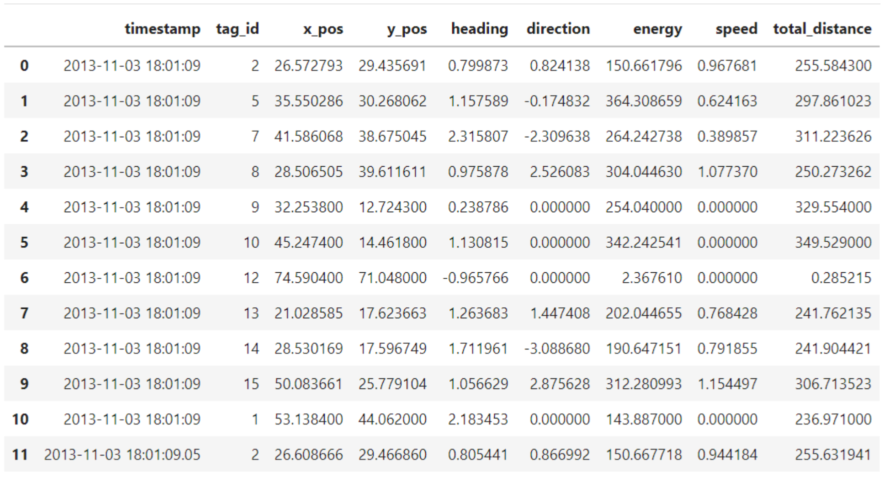


Figure 3.19: High-Quality cylindrical panorama video available in the dataset. Camera specs for this video are available in the Table 3.2, second column. We will not use this video for our analysis, due to its strong radial distortion and the presence of the audience, which may disturb the MOT algorithm if we want to track players on the field. (image from [5]).



	timestamp	tag_id	x_pos	y_pos	heading	direction	energy	speed	total_distance
0	2013-11-03 18:01:09	2	26.572793	29.435691	0.799873	0.824138	150.661796	0.967681	255.584300
1	2013-11-03 18:01:09	5	35.550286	30.268062	1.157589	-0.174832	364.308659	0.624163	297.861023
2	2013-11-03 18:01:09	7	41.586068	38.675045	2.315807	-2.309638	264.242738	0.389857	311.223626
3	2013-11-03 18:01:09	8	28.506505	39.611611	0.975878	2.526083	304.044630	1.077370	250.273262
4	2013-11-03 18:01:09	9	32.253800	12.724300	0.238786	0.000000	254.040000	0.000000	329.554000
5	2013-11-03 18:01:09	10	45.247400	14.461800	1.130815	0.000000	342.242541	0.000000	349.529000
6	2013-11-03 18:01:09	12	74.590400	71.048000	-0.965766	0.000000	2.367610	0.000000	0.285215
7	2013-11-03 18:01:09	13	21.028585	17.623663	1.263683	1.447408	202.044655	0.768428	241.762135
8	2013-11-03 18:01:09	14	28.530169	17.596749	1.711961	-3.088680	190.647151	0.791855	241.904421
9	2013-11-03 18:01:09	15	50.083661	25.779104	1.056629	2.875628	312.280993	1.154497	306.713523
10	2013-11-03 18:01:09	1	53.138400	44.062000	2.183453	0.000000	143.887000	0.000000	236.971000
11	2013-11-03 18:01:09.05	2	26.608666	29.466860	0.805441	0.866992	150.667718	0.944184	255.631941

Figure 3.20: Some typical entries for the sensor dataset we will use and that are provided by [5]. Sensors are worn by players on their lower torso for the entire duration of the match. For every timestamp, there is a unique row corresponding to a player identified by a sensor with a given tag (tag_id).

- **y_pos** *(float)* $[m]$: y-direction player's relative position in the field. Valid values are such that $0 < y < 68$. The authors claim a precision of 1 meter for the sensor;
- **heading** *(float)* $[rad]$: orientation of the player's head with respect to the y-axis, where 0 is the y-axis;
- **direction** *(float)* $[rad]$: orientation of player's movement with respect to the y-axis, where 0 is the y-axis;
- **energy** *(float)* $[undefined]$: estimated energy consumption since last sample. The value is based on the step frequency as measured by the on-board accelerometer. There is no unit for it, since it might vary from player to player.
- **speed** *(float)* $[m/s]$: player's speed;
- **total_distance** *(float)* $[m]$: cumulative distance traveled by a given player.

However, we will focus only on the first four of them, namely, *timestamp*, *tag_id*, *x_pos* and *y_pos*. In addition, for this dataset the **index** is set to be the **timestamp**. As can be seen in the figure 3.18a, the pitch measures (68×108) meters.

3.1.2 Video data

The camera array is positioned in the center of the field, as shown in Fig. 3.18a, and its view is the one of Fig. 3.18b. It is reasonable to have more player density and, therefore, more targets to track in the middle of the field: hence, we chose to use only the camera pointed to that direction, namely with referral to Fig. 3.18b, the one in the middle. In the dataset, such a camera is called camera 1, and its technical specifications can be found in Table 3.2. Despite the presence of a High-Quality cylindrical panorama video that actually frames the whole pitch, it includes a strong radial distortion and the audience is visible, which might disturb the tracking algorithm. For these reasons (see Fig. 3.19), this video is not used, although the camera specs are present in the second column of 3.2. The videos coming from three different cameras are shutter synchronized, therefore allowing to compare, merge, and analyze the information obtained by simultaneous images belonging to different views. Note that the cameras actually frame the entire soccer field.

All the videos captured are stored using the same codec with the same parameters, except for the resolution and frames per second (fps). For every camera, the entire video has been divided into 3-second fragments, which have been encoded in H.264 using *libx264* (see table 3.3). Therefore, a single fragment contains exactly $n_{frames} = 3 \cdot fps$ which, in our case, will be a total of 90 frames. The file name is written in such a way that it contains a timestamp of the first frame in the fragment, with an accuracy of nanoseconds. This permits the synchronization of the videos with the data of the XYZ

profile	high
preset	ultrafast
tune	zerolatency
Colorspace	YUV 420 planar
GOP	30 x fps
Files	xxxx YYYY-MM-DD hh:mm:ss.000000000.h264

Table 3.3: Parameters that the authors of the dataset [5] have used, in the H264 encoding, to create the 3-second video fragments of the football match.

sensor detailed in Section 3.1.1.

3.2 Preprocessing

Let us now discuss how the preprocessing has been made, namely, the steps taken to be able to track players' trajectory to the pitch coordinates starting from the image ones. Then, we will show that, despite the authors claim to have completely deleted the radial distortion, a little is still present when making the homography and therefore will be corrected. Finally, we will describe how the sensor data and the returned sensor coordinates have been transformed into pitch coordinates. The last step we take is to re-sampling the timestamps every 50 ms.

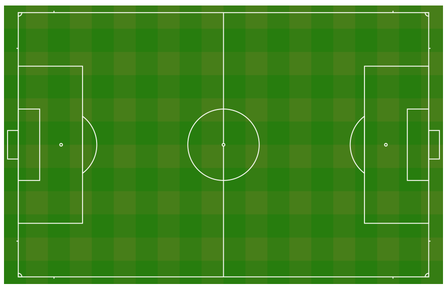
Homography

In Computer Vision field, usually, when dealing with two images of the same planar surface in space, one may want to compute a transform linking the two figures: the **planar homography** (see Fig. 3.21b). It is common to not have access to the *depth* coordinates of pixels in a photographic image, making it not possible to reconstruct the z-depth in the real-world coordinates. This is the reason why we can reasonably assume that all points lie on the same surface $\hat{\mathbf{n}}_0 \cdot p + c_0$. We can compute the mapping equation from the points of one image to another with different perspectives:

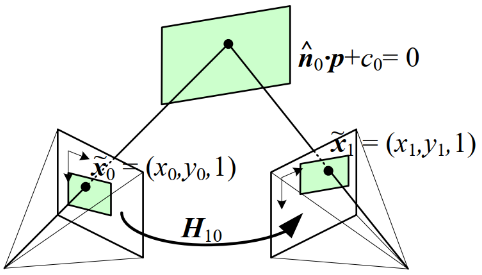
$$\tilde{\mathbf{x}}_1 = \tilde{\mathbf{H}}_{10}\tilde{\mathbf{x}}_0 \tag{3.35}$$

Where $\tilde{\mathbf{H}}_{10}$ is a general homography 3×3 matrix, and $\tilde{\mathbf{x}}_1$ and $\tilde{\mathbf{x}}_1$ are now 2-dim homogeneous coordinates (that is, 3-dim vectors). In this matrix, $h_{33} = 1$, and there are 8 degrees of freedom. To find their best estimates *at least* 4 point correspondences suffice for the task.

Using the formalism described above, we want to find the transformation that relates the trajectories in the space of **camera-pixel coordinates** to those of the image represented in Fig. 3.21a. It is clear that, with reference to equation (3.35), all points lie on the same surface, that is, the football field. To address this problem, it is

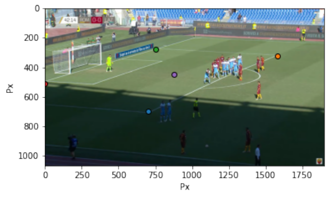


(a) Image of a generic football pitch, onto which we want to project the trajectory of a player in the real world.

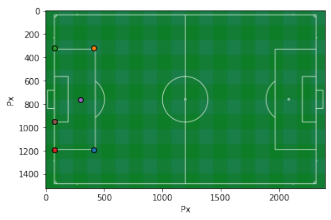

$\hat{\boldsymbol{n}}_0 \cdot \boldsymbol{p} + c_0 = 0$
$\tilde{\boldsymbol{x}}_0 = (x_0, y_0, 1)$
$\tilde{\boldsymbol{x}}_1 = (x_1, y_1, 1)$
$\boldsymbol{H}_{10}$

(b) The coordinates in the images $\tilde{\mathbf{x}}_0$, $\tilde{\mathbf{x}}_1$ of point $\mathbf{p}$ lying on a plane, are linked through a *planar homography* transformation $\mathbf{H}_{10}$. Image taken from [29].

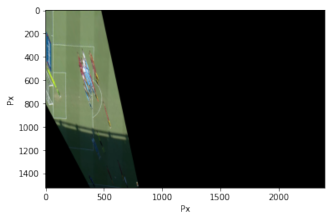
Figure 3.21



(a) A broadcast image of a soccer game. Source points to be mapped to the pitch are visible. Frame from [62].



(b) The pitch with the colored target points visible.

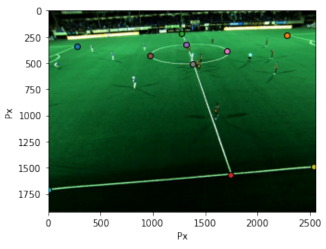


(c) The result of the source image after applying the homography transform.

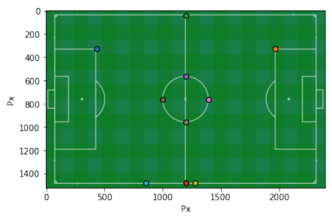
Figure 3.22: Pipeline to be followed when we want to apply the homography. In the source image we select some significant source points, while in the target image we select their correspondent ones. The same color denotes a source-target pair.

common [19] to find some points of interest that can be uniquely defined and recognized on a football pitch. As a first step, we extract a sample frame from a Youtube video showing the target area of the field and highlight the *source* points of which we want to find the transformation. The frame showing these points is that in Fig. 3.22a. The second step is to find the correspondent target points in Fig. 3.22b, finally using the Python wrapper library openCV2 [61] to find such a transformation matrix. The output image after we have applied such a transformation can be seen in Fig. 3.22c. Note that we have used a generic broadcast image, which we expect to be as high quality as possible, to assess the correctness of the homography approach.

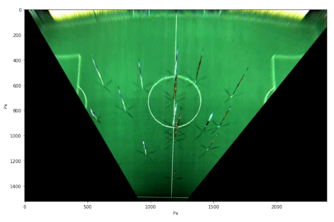
Instead,let us apply the same method to a frame from our dataset. We will show, as before, the source image highlighting our point of interest and their correspondence on the football pitch. Finally, the output image is the one visible in Fig. 3.23. The result we obtain is sufficiently satisfactory, and the image from the video can be mapped to the pitch coordinates. Therefore, one can expect to be able, once the players in the



(a) The first frame of the video we have analyzed throughout our work. Source points to be mapped to the pitch are visible.



(b) The pitch with the colored target points visible.



(c) The result of the source image after applying the homography transform. A little radial distortion is visible.

Figure 3.23: Pictures showing the pipeline followed by the homography procedure. We select several source points, which are visible in different colors, and we map them to their corresponding target points. The pairs source-target are distinguishable since they are colored in the same way. The frames are taken from [5].

field have been tracked and we have made some hypotheses to provide an estimate their position, to plot their trajectory on the pitch. However, a little **radial** distortion is present, especially in the upper left and upper right corners of the image, which we want to further correct.

Radial Distortion

Ideal imaging models assume that cameras obey a *linear* projection, where straight lines in the world result in straight lines in images. Many *wide-angle* lenses exhibit a **radial distortion**, whose noticeable effect is a curvature in the projection of straight lines. In the warped Fig. 3.23c , we see that its coordinates are displaced *away* from *image center* by an amount that is *proportional* to their radial distance (**barrel** distortion). The usual and simplest models deal with this problem using *low order polynomials*, however, since we have not been able to compute the *calibration parameters* useful for undistorting the image, we follow the approach proposed by Devernay et Faugeras [60].

They essentially propose a method to perform the calibration automatically, without relying on camera specifications. Using their approach, any camera can be considered a *pinhole* camera after applying the inverse of the distortion function to the features of the image. We will use their distortion model designed for *fish-eye* lenses, based on the way images are actually produced. In fact, the distance between every point in the image and the principal point is *roughly proportional* to the angle between the point in the real world, the optical center, and the optical axis. The angular resolution is circa proportional to the image resolution along an image radius: the farther we are from the center, the wider and more distorted the image, and visually speaking, the fewer details are clear. The proposed model *FOV* accounts for only one parameter

ω, which is the *field-of-view* of the corresponding *ideal* fish eye lens. This angle might not correspond to the real camera field-of-view, since the fish-eye optics might follow some other models. Formally, the corresponding **distortion** and its **inverse** functions are, respectively:

$$r_d = \frac{1}{\omega} \arctan\left(2r_u \tan\frac{\omega}{2}\right) \tag{3.36}$$

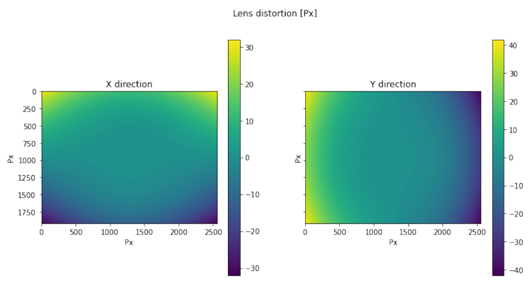
$$r_u = \frac{\tan(r_d\omega)}{2\tan(\omega/2)} \tag{3.37}$$

The correspondent code to compute the mapping between the *distorted* and the *undistorted* image coordinates is the one of 3.1.

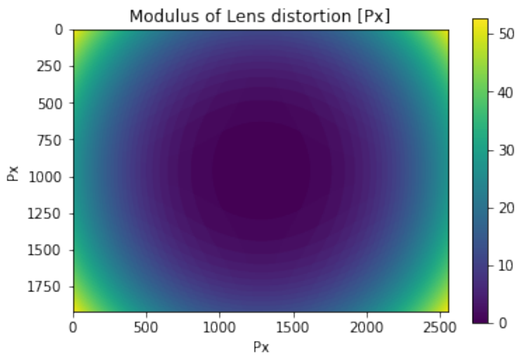
```python
import numpy as np
import pandas as pd

def dist(x,y):
    return np.sqrt(x**2 + y**2)

def correct_fisheye(src_size, dest_size, dx, dy, factor):
    # convert dx,dy to relative coordinates
    rx, ry = dx-(dest_size[0]/2), dy-(dest_size[1]/2)
    # calc theta
    r = dist(rx,ry)/(dist(src_size[0],src_size[1])/factor)
    if 0==r:
        theta = 1.0
    else:
        theta = np.arctan(r)/r
    # back to absolute coordinates
    sx, sy = (src_size[0]/2)+theta*rx, (src_size[1]/2)+theta*ry
    # done
    return (int(round(sx)),int(round(sy)))
```

Listing 3.1: Code to compute (sx sy) namely the coordinates of the distorted images corresponding to a given (dx dy) of the distorted source and the sizes of the source and target images. Code from [63]

We want now to have a hint of how much pixels are distorted for a fixed value of the unique parameter ω (see Fig. 3.24). It is observed that, as expected, the farther we are from the center of the image, the more distorted the image is, according to a radial symmetry. In fact, at the corners, the modulus of this distortion in pixels unit even reaches $50Px$ for a $1920 \times 2560Px$ image. We set the unique parameter ω to $\omega = 0.65$ after a trial-and-error procedure to rectify the lines of the football pitch as much as possible after homography was applied. The results we have obtained using this approach are those shown in Fig. 3.25. Note that the radial distortion is now almost absent, and the output image exhibits straighter lines. Using the code 3.1 and equation (3.37), we can find the relation between the distorted image coordinates

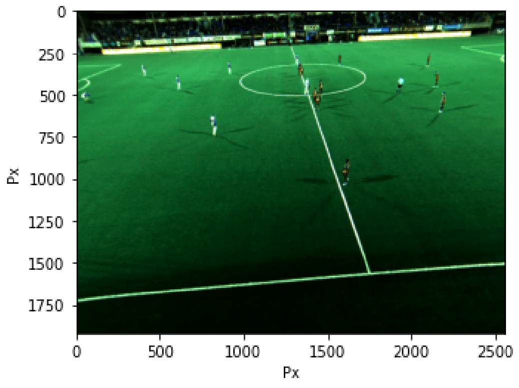


(a) Difference in pixel of the distortion for a $\omega = 0.65$ along XY axis.

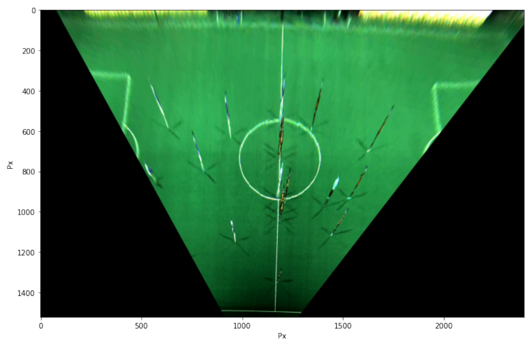


(b) Modulus of the distortion in pixel for $\omega = 0.65$

Figure 3.24: Distortion, measured in Pixel units, for a $1920 \times 2560Px$ image according to the model of equation (3.37) by [60]. The parameter ω is set to $\omega = 0.65$.



(a) Output undistorted image using $\omega = 0.65$.



(b) Undistorted image combined with the homography. Despite lines are not perfectly straight, the radial distortion has almost vanished.

Figure 3.25: Output images after that we have applied the correction for the radial distortion listed in code 3.1 and using the parameter $\omega = 0.65$, respectively, before and after applying the homography.

(x_d, y_d) and the undistorted image coordinates (x_u, y_u). In other words, we obtain a certain function $f\ s.t.(x_u, y_u) \to f(x_u, y_u) = (x_d, y_d)$. We want to explicitly populate the target image pixel by pixel, that is, using two loops over the width W and height H, and denoting the actual index with ii, jj, we find which pixel will be the correspondent in the distorted image. Formally, for every $ii \in [0, H]$ and for every $jj \in [0, W]$:

$$\text{Undistorted Image}[ii, jj] \ \leftarrow \ \text{Distorted Image}[s_x, s_y] \tag{3.38}$$

where (s_x, s_y) are found with the algorithm 3.1. In this way, we can create a **Python dictionary** of such mappings where, given a certain pixel (s_x, s_y) of the distorted image, we compute its correspondent in the *undistorted* coordinates (x_u, y_u). This is done instead of inverting the function (3.37), which is not trivial.

However, we have noted that this is not a bijective relation between the two sets. Defining a fixed pixel in the target image whose coordinates are (x_u, y_u), we have found that there may be more than one pixel in the source image (x_d, y_d) that are mapped to it, as well as some of the pixels (x_u, y_u) that do not have an image. To deal with these problems, if a given (x_d, y_d) has more than one image, we take the arithmetic mean of them to obtain a one-to-one mapping. Moreover, to deal with the eventual absence of $f(x_u, y_u)$, we drop the single-pixel discretization and iterate over "quarters" of pixels, namely with steps of 0.25 pixels for ii and jj indexes. Recall now that our final goal is to relate pixels in the *distorted* coordinates to their correspondents in the *undistorted* image. Using the two tricks above, namely taking the mean if a certain (x_d, y_d) has more than one image and using the quarter-pixel resolution, one can efficiently cover all pixels from the original distorted image. The final Python dictionary object storing this mapping is saved as a *pickle* binary file.

Sensor preprocessing

Among all the data provided by the sensor (see Fig. 3.20), we are only interested in a part of it. That is why we extract only the following columns:

- **timestamp**: which will be our index;
- **tag_id**: the unique identifier of every player, we want to formulate an algorithm that assigns the trajectories of the detection box (output of the MOT) on the football pitch to them;
- **x_pos**, **y_pos**: we want to transform these coordinates in meters to the ones on the football pitch, which will be in Pixel.

We want now to find the mapping between the players' position in meters, namely (x_{pos}, y_{pos}), and their position on the football pitch in pixel coordinates. This mapping, recalling that the pitch has dimensions $(W_m, L_m) = (105{\times}68)[m]$, is obtained as follows.

As a first step, we define the *Length* and *Width* of the pitch in the pixel coordinates as L_{Px}, W_{Px}. Then, we can define the ratio between them in both directions, thus obtaining a *proportion factor* $R_{x,y}$, which is dimensionally $[Px/m]$:

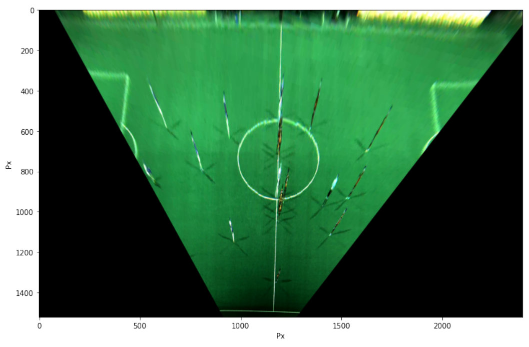
$$R_x = \frac{W_{Px}}{W_m} \qquad R_y = \frac{L_{Px}}{L_m} \tag{3.39}$$

Then, the mapping between the (x_{pos}, y_{pos}) and their correspondent (x_{Px}, y_{Px}) is:

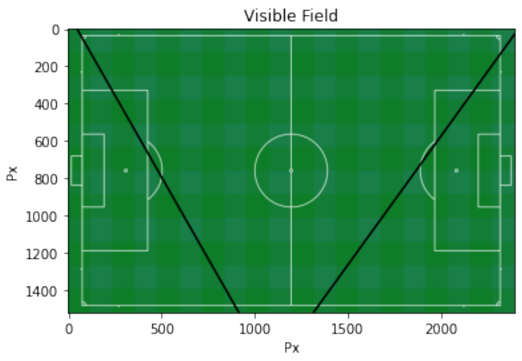
$$(x_{Px}, y_{Px}) = \left(\frac{x_{pos}}{R_x} + TL_x \ , \ \frac{L_m - y_{pos}}{R_y} + TL_y \right) \tag{3.40}$$

where (TL_x, TL_y) $[Px]$ is the top left corner of the pitch image (see Fig. 3.21a). Note that $R_x \approx R_y \approx 21[Px/m]$, therefore 1 meter in the real world is about 21 pixels in the image of the football pitch.

We now want to filter out the data provided by the sensor in such a way as to neglect rows referring to players that cannot be seen by the camera and, therefore, detected by our tracker. To achieve such a result, we need to find a parameterization in the Pixel coordinates for the **field of view** of the central camera. This is done visually by comparing the two images shown in 3.26, with the result shown in the picture on the right side.



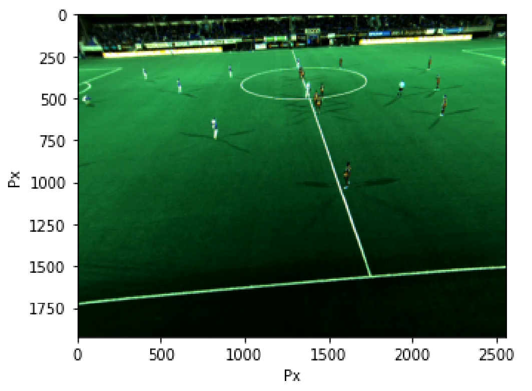
(a) Image without the radial distortion and after the homography.



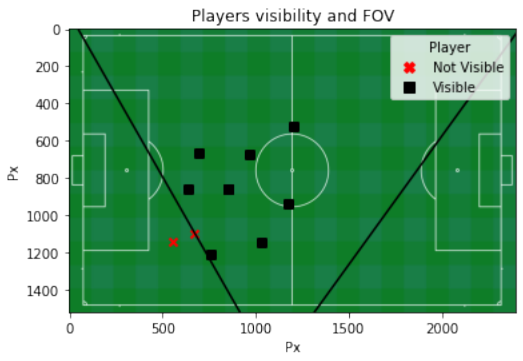
(b) Field of view of the central camera taken into account in our study.

Figure 3.26: For every time instant, after we have applied the homography, we can define a field of view of the camera. When a player is inside this visibility cone, he is potentially detected and consequently tracked. We show here a visual comparison between the undistorted image, after applying the homography and the parametrization we have used to define such a visibility cone.

Then, the following step is to define a **mask** for the player-sensor dataset, which is *true* when a player is *inside* the field of view of the camera. Depending on their visibility, which is a function of their actual position in the pitch, several rows are filtered out. In this way, we obtain the **final** sensor dataframe that we will use for this



(a) Players in the first frame of the video being analyzed and taken from [5]. The home team, that is the team of which we have XYZ sensor data available, wears white jersey.



(b) Players are either denoted by a red cross or a black square depending on their visibility. This image refers to the first frame of video in. The player positions are obtained through the XYZ sensor data.

Figure 3.27: For every time instant we create a mask that denotes whether a player is visible. Then we keep only visible players. In these images, the players of the home team, that wear white jersey, are represented in the pitch with a red cross when they are not visible, while with a black square if they are inside the visibility cone. The players' positions are obtained through the sensors data, after applying the transformations of equation (3.40).

research.

We want now to compute the systematic error that we introduce if we did not consider the radial distortion, that is, how the radial distortion, if present, affects the computation of the position when we have applied the homography. Visually, one obtains the results shown in Fig. 3.28. We can see that, as expected by comparing Fig. 3.25 and Fig. 3.23, the largest correction is obtained at the corners of the image, that is, the furthest region from the center of the starting image. However, the error is of the order of meters and is therefore comparable to that of the sensors, that is, **not** negligible for our research.

Tracker output preprocessing

Let us now discuss how we obtained the **tracklets**, that is the trajectories of the bounding box of players after they have been detected and subsequently tracked by our algorithm. For the tracking part, we have used (*ByteTrack* [2]) Multi-Object Tracker, in a pretrained version for the MOT20Challenge [4] competition. When its input is a video, it returns the outputs shown in Fig. 3.29: it always consists of a text file with the following entries (see Fig. 3.29b):

- **frame**: the frame number;
- **id**: an integer that uniquely identifies a bounding box;

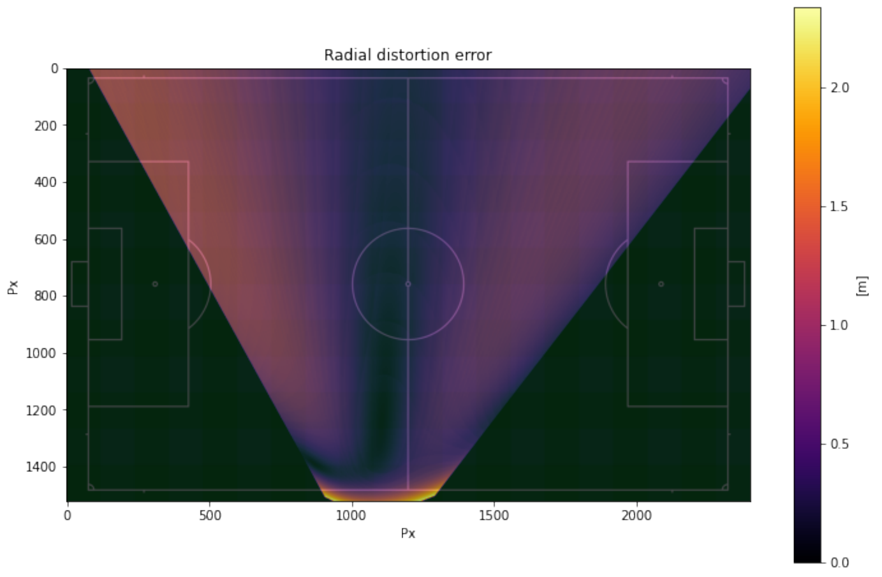


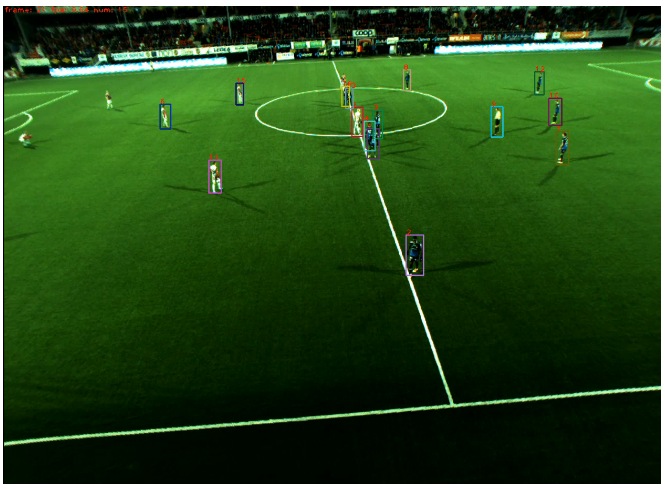
Figure 3.28: Euclidean distance, in meters, between the target pixels when we have applied the correction to the radial distortion, and the one without the radial distortion. The correction to the radial distortion has been applied using Algorithm 3.1, to a 1920×2560 Px image, with parameter $\omega = 0.65$ according to the model in (3.37).

- **x**: the x coordinate $[Px]$ of the top left corner of the bounding box;
- **y**: the y coordinate $[Px]$ of the top left corner of the bounding box;
- **wx**: the width $[Px]$ units of the bounding box;
- **wy**: the height $[Px]$ units of the bounding box;
- **confidence**: the confidence of the tracker that the target is a pedestrian.

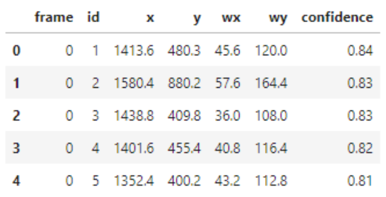
However, it is not mandatory to also have the video as output, but we decided to store it for a visual quality assessment to find the best hyperparameters for the tracking. Note that we have the *sensor* data only for the **Home** team, which wears white jerseys, whereas the tracker does not distinguish between different teams of people in the image and tries to detect anyone, including the referee and the linesmen. To assess the goodness of our preprocessing approach, we created a video that lasts two minutes, that is, using 20 video fragments of 3 seconds. Recalling that the camera records at a speed of 30 fps, according to Table 3.2, the total number of frames processed will be $120 \cdot 30 = 3600 \ frames$.

Let us assume that we can estimate any player's position using the coordinates of his corresponding bounding box. That is, we compute the player position $(x_{pl}, y_{pl}) = \mathbf{x}_{pl}$ in the image coordinates as:

$$\mathbf{x}_{pl} = (x_{pl} \ , \ y_{pl}) = (x + wx/2 \ , \ y + wy) \tag{3.41}$$



(a) A sample frame from the video that ByteTrack has returned as output. The players that are detected are assigned with a colored bounding box, above which there is a number that uniquely identifies it.



	frame	id	x	y	wx	wy	confidence
0	0	1	1413.6	480.3	45.6	120.0	0.84
1	0	2	1580.4	880.2	57.6	164.4	0.83
2	0	3	1438.8	409.8	36.0	108.0	0.83
3	0	4	1401.6	455.4	40.8	116.4	0.82
4	0	5	1352.4	400.2	43.2	112.8	0.81

(b) A sample of the *txt* file output of the tracker. The *id* column denotes the id that uniquely identifies a bounding box, whereas the (x, y) are the coordinates of the top left corners in the image pixel coordinates. wx, wy are instead, respectively, the width and the height in pixel of a detection box. Finally, the confidence is a score between 0 and 1 of how much the tracker is confident about its output. The closer to 1, the more confident.

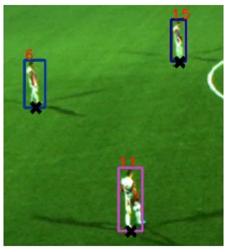
Figure 3.29: Output of ByteTrack after an input video has been preprocessed. In this case, we provide as input the video from the dataset [5].

where (x, y, wx, wy) are the parameters that univocally denote a bounding box. This assumption is reasonable since, for most of the time, a player is standing during a game. So, his position is assumed to be basically the position of his feet (see Fig. 3.30a), which are located in the median of the bottom side of the corresponding detection box. In fact, using this hypothesis, we seem to avoid errors since the feet are supposed to stay on the ground, that is, on a *plane*. Therefore, we are allowed to use the **homography matrix $\mathbf{H}_{10}$** we have estimated before. Finally, we can compute the player's position on the football pitch by applying such a change of coordinates:

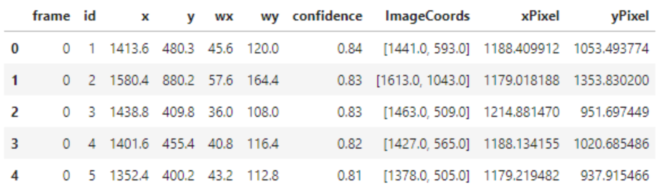
$$\mathbf{x_1} = \mathbf{H}_{10}\mathbf{x_u} \tag{3.42}$$

Here $\mathbf{x_1}$ is the vector that denotes a player's position on the pitch in pixel coordinates, and $\mathbf{x_u}$ is the output vector that we obtain after we have applied the radial distortion correction to $\mathbf{x}_{pl}$. This allows us to obtain a dataset that is similar to the one shown in Fig. 3.30b.

We are now able to draw players' trajectories on the field, since we have finally computed their position on the pitch in the *pixel* coordinates. Let us plot (see Fig. 3.30a), as an example, a couple of trajectories that belong to those that last the most ($>$ 30 seconds): the referee, which starts at the beginning and interrupts after 1500$frames$ ($\sim$ 50 seconds) and the player's tracklet denoted by the detection box with id #32 which approximately lasts 60 seconds.



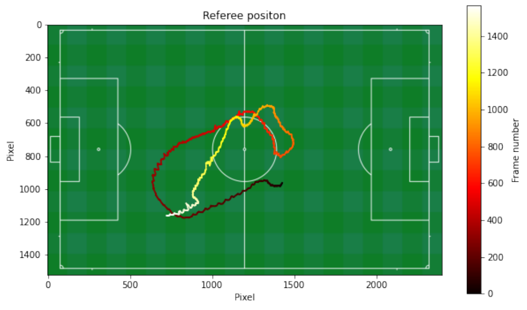
(a) A player's position is computed as the median of its bottom side of the corresponding detection box, as in equation (3.41). Such a position is denoted with a black cross.



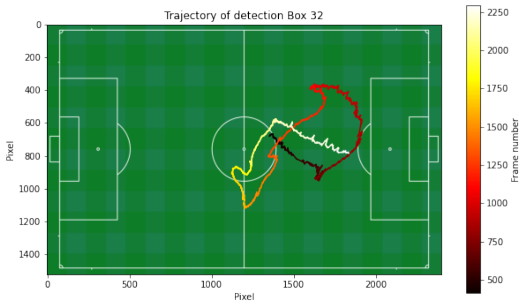
	frame	id	x	y	wx	wy	confidence	ImageCoords	xPixel	yPixel
0	0	1	1413.6	480.3	45.6	120.0	0.84	[1441.0, 593.0]	1188.409912	1053.493774
1	0	2	1580.4	880.2	57.6	164.4	0.83	[1613.0, 1043.0]	1179.018188	1353.830200
2	0	3	1438.8	409.8	36.0	108.0	0.83	[1463.0, 509.0]	1214.881470	951.697449
3	0	4	1401.6	455.4	40.8	116.4	0.82	[1427.0, 565.0]	1188.134155	1020.685486
4	0	5	1352.4	400.2	43.2	112.8	0.81	[1378.0, 505.0]	1179.219482	937.915466

(b) The pre-processed output of the tracker. Using the estimated player's position in image coordinates, we use the homography to estimate his position in the pitch coordinates. The first columns are those of figure 3.29b, whereas *ImageCoords* refers to the estimate player's position in image coordinates after the radial distortion has been corrected. While, *xPixel* and *yPixel* are the coordinates on the pitch after the homography has been applied.

Figure 3.30: After that a player's position has been estimated, by taking the median point of the lower side of the correspondent bounding box (3.41), its coordinates are first undistorted (3.38) and finally mapped to the pitch ones via the homography (3.42).

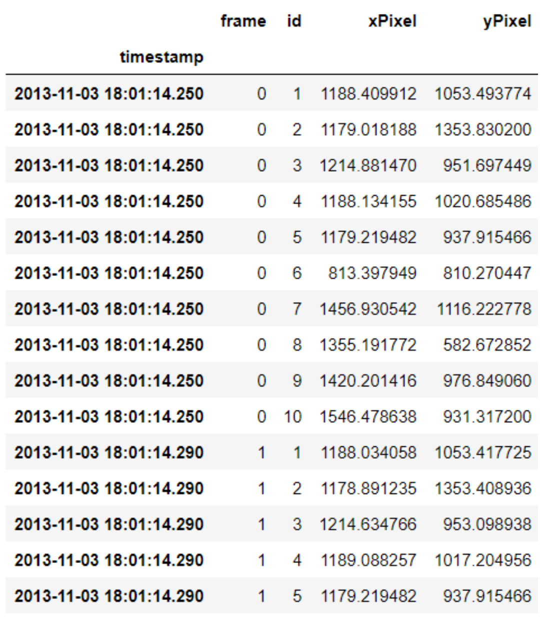


(a) Referee trajectory through time. His tracklet does not cover the whole video, and stops around frame 1500.



(b) The trajectory followed by the player tracked by the detection box id #32.

Figure 3.31: Tracklets corresponding to two different bounding boxes with different IDs. Every tracklet is obtained in a first step, by estimating the player's position from its bounding box coordinates (3.41). Then, we apply the correction to the radial distortion (3.37) and finally project its position in image coordinates to the pitch coordinates via homography (3.42). The color of the curves changes according to the frame they refer to: as time passes, the color becomes brighter.



	frame	id	xPixel	yPixel
timestamp				
2013-11-03 18:01:14.250	0	1	1188.409912	1053.493774
2013-11-03 18:01:14.250	0	2	1179.018188	1353.830200
2013-11-03 18:01:14.250	0	3	1214.881470	951.697449
2013-11-03 18:01:14.250	0	4	1188.134155	1020.685486
2013-11-03 18:01:14.250	0	5	1179.219482	937.915466
2013-11-03 18:01:14.250	0	6	813.397949	810.270447
2013-11-03 18:01:14.250	0	7	1456.930542	1116.222778
2013-11-03 18:01:14.250	0	8	1355.191772	582.672852
2013-11-03 18:01:14.250	0	9	1420.201416	976.849060
2013-11-03 18:01:14.250	0	10	1546.478638	931.317200
2013-11-03 18:01:14.290	1	1	1188.034058	1053.417725
2013-11-03 18:01:14.290	1	2	1178.891235	1353.408936
2013-11-03 18:01:14.290	1	3	1214.634766	953.098938
2013-11-03 18:01:14.290	1	4	1189.088257	1017.204956
2013-11-03 18:01:14.290	1	5	1179.219482	937.915466

Figure 3.32: The final dataset, which was the output of the tracker, after we have performed the preprocessing.

Now we want to find the **timestamp** that denotes **time** to which a certain frame belongs. Recall from the video specs that every second contains 30 frames. Therefore, we can extract the **starting time** from the name of the *first* video fragment, that is: *0056_2013-11-03 18:01:14.248366000.h264*. We further round the microseconds to its *third* decimal place, saving only the tens (i.e., *2013-11-03 18:01:14.250000*), in such a way that we can compare the *sensor* data and this dataframe, after setting the timestamp as its index. The final dataset is shown in Fig. 3.32.

Time handling

After having computed the final dataframes for the sensor data and the tracker output, our next step consists of comparing the two, exploiting the shared **timestamp index**: using it, it is possible to merge the different information from the separate datasets. In fact, the goal of this work is to **relate** the **players' position** provided by the sensor data, to their correspondent tracklet extracted from the corresponding *bounding box* for the video under investigation.

As shown above, we have been able to project the position of the player on the pitch (see Fig. 3.31) denoted with a unique detection box *id*. Thanks to a visual comparison with the trajectories provided by the sensor, we can provide a match "by hand" of a certain bounding box with a corresponding sensor. That is, for the video in analysis, we have been able to match the following pairs (tag_id $\longleftrightarrow$ detBox_id):

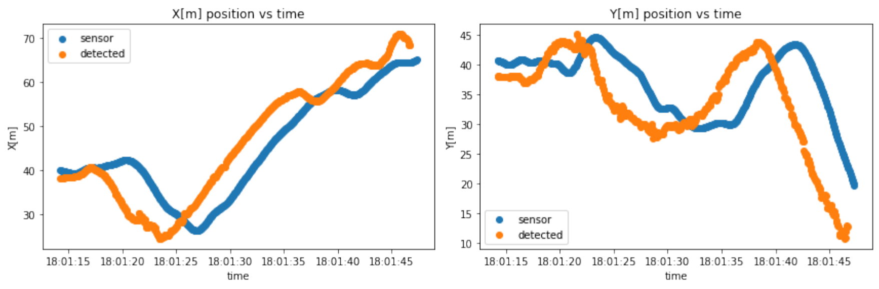


Figure 3.33: Visual comparison of XY position as function of time between the trajectory followed by the sensor with tag #15 and the detection box #5 in pitch coordinates.

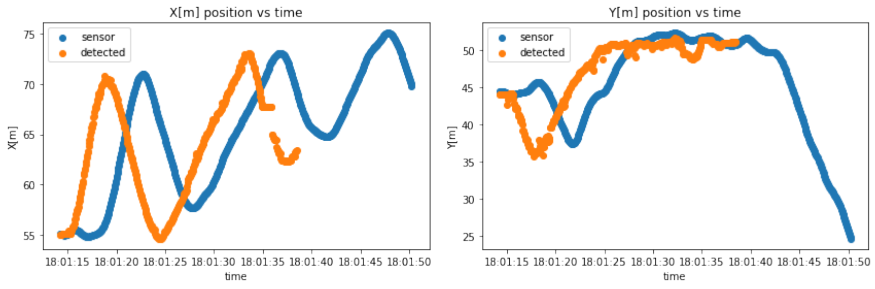


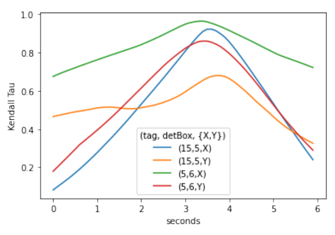
Figure 3.34: Visual comparison of XY position as function of time between the trajectory followed by the sensor with tag #5 and the detection box #6 in pitch coordinates.

- (15$\longleftrightarrow$5)
- (5$\longleftrightarrow$6)

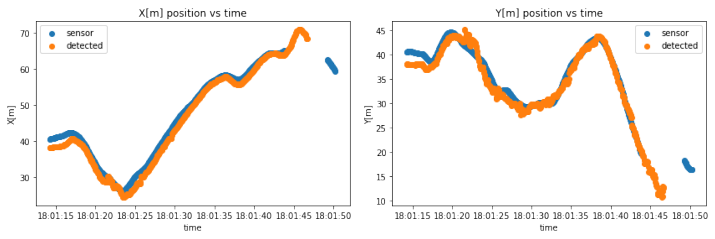
These trajectories and their components on, respectively, the X and Y axes as function of time are the ones that can be inspected in Fig. 3.33 and Fig. 3.33 for *only* the first 30 seconds. In such a time interval, sometimes the bounding box does not exist, and therefore some discontinuities are present. However, it is reasonable to assume that, despite the fact that a player is not detected, he will continue to exist and will move in such a way as to reach its destination according to a linear motion until the track disappears for good. In addition, one can easily note that there exists a **temporal shift** between the two curves: in fact, they are basically the same, but simply translated by some seconds. Let us define the *optimal* amount of shift needed as the time that *maximizes* the **Kendall's Tau** τ correlation. Kendall's Tau correlation describes how strongly two variables are correlated, assuming that we can provide them with an order and their *relation* is monotonic. Values close to 1 indicate strong agreement, while values close to -1 indicate strong disagreement [4].

Let us denote with $(x_1, y_1), ..., (x_n, y_n)$ the set of observations of the joint random variables X and Y, such that all values are unique. A generic pair of observations

[4]in other words: if one increases, the other one should always decrease ($\tau = -1$) or increase ($\tau = +1$).



(a) Kendall τ for different time shifts in range [0,6] seconds of X and Y positions for two different pairs of players trajectories (sensor data) and tracklets (detection boxes data).



(b) After the time shifting, one can see that the two curves are essentially the same one for the pair $(5 \longleftrightarrow 6)$.

Figure 3.35: Using the Kendall's τ (3.43) as a measure for the correlation between two variables, we want to find the *optimal* amount of time we must shift our sensor trajectory timestamp to maximize its correlation with the corresponding detection box tracklet. In fact, after the time shifting of $t^* = 3.5s$ the two curves, one depicting the position estimate obtained from the sensor, and the second one from the corresponding tracklet, are really close to each other.

(x_j, y_j), (x_i, y_i) with $i < j$ are said to be **concordant** if the sort order (x_i, x_j) and (y_i, y_j) agrees. In other words, if either $x_i > x_j \land y_i > y_j$ or $x_i < x_j \land y_i < y_j$ holds. If this does not hold, they are said to be **discordant**. Formally, the coefficient τ is computed as:

$$\tau = \frac{\text{number of concordant pairs} - \text{number of concordant pairs}}{\binom{n}{2}} \tag{3.43}$$

where the binomial coefficient accounts for the number of ways to choose 2 items from n items. If we consider the two random variables to be the $\{x, y\}$-position of the *detected* player and the $\{x, y\}$-position of the associated sensor, we note that the hypotheses for the use of the Kendall τ are satisfied. In fact, the two variables are two estimates of the same quantity, that is, the position of the player on the ground. Therefore, we expect the two quantities to be **concordant**.

To compute the *optimal time shift*, we try different time shifts in the range $(0, 6)$ seconds with steps of 0.1 seconds, since we expect that this optimal value belongs to this interval. Obviously, we keep the X and Y values that have a common timestamp of the two dataframes: only if this condition holds, it is meaningful for them to be compared. The results we obtain are those of Fig. 3.35a.

The optimal shift is assumed to be the mean value of the different time shifts that maximize such a correlation measure, which is reasonable from the Figure 3.35a: it is clear that all the curves exhibit a maximum around a common value of the time shift.

Denoting the optimal time shift as t^*, it is found that:

$$t^* = 3.5s \pm 0.2s$$

Therefore, we **shift back** the *timestamp* index for the *sensor* dataset of 3.5s, thus obtaining the **final sensor dataframe**.

However, tracklets can disappear for some time: the visible effects are essentially the gaps in the curves of Fig. 3.35b. As mentioned above, it is reasonable to fill these gaps by a linear interpolation. In fact, the expected behavior would be that of Figg. 3.33, 3.34. To achieve such a result for the detection dataframe, we basically performed a resample at a $50ms$ frequency, which is also the frequency of the data provided by the sensor, thus obtaining our **final detection dataframe**. Note that this solution is inspired by the interpolation performed in [2] to ensure a *continuity* for the existence of bounding boxes even when targets are occluded.

3.3 The assignation algorithm

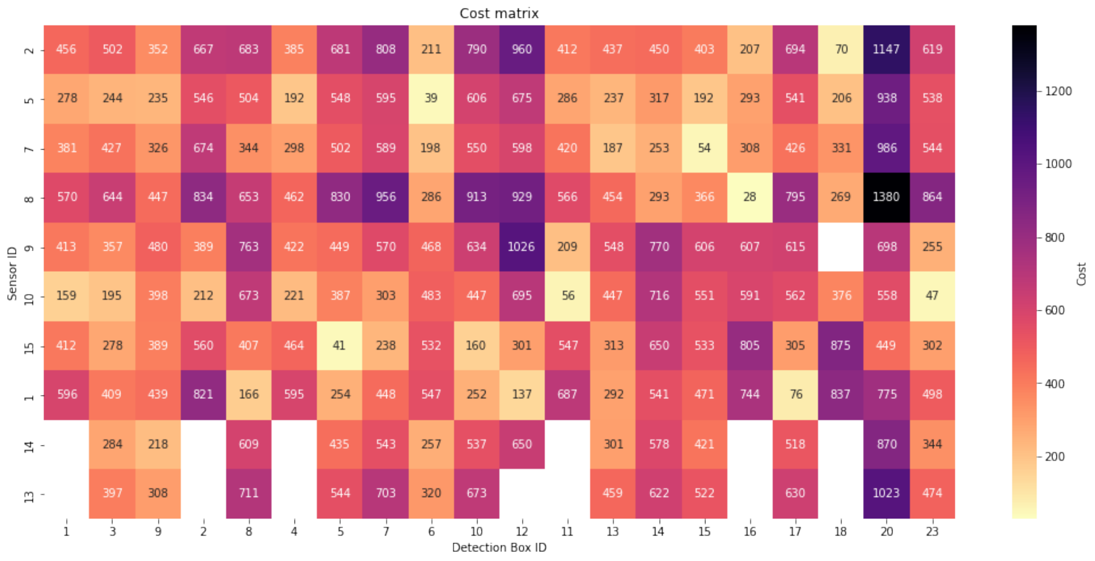
Let us now discuss the algorithm we implemented, through which we can assign the detection boxes trajectories (we will refer to them as "*tracklets*") to the sensor trajectories. Let us start again from the final version of the two dataframes: the sensor one, whose timestamp has been shifted back by 3.5 seconds, and the detection one, which has been resampled at a $50ms$ frequency.

The natural cost one can use for assignation can be the **Euclidean distance** between two positions at the same time. Let us define the position of the player estimated by the sensor at a given time instant t as $(x_{sen}^{(t)}, y_{sen}^{(t)})$, while the position estimated using the bounding box at the same instant $(x_{det}^{(t)}, y_{det}^{(t)})$. The *Euclidean distance* d between $\mathbf{x}_{det}^{(t)}$ and $\mathbf{x}_{sen}^{(t)}$ at a given time instant t is:

$$d^{(t)} = \sqrt{(x_{det}^{(t)} - x_{sen}^{(t)})^2 + (y_{sen}^{(t)} - y_{det}^{(t)})^2} \tag{3.44}$$

It is reasonable to assume that if a matching between a tracklet of a given player and its sensor is carried out successfully, then the Euclidean distance between the two trajectories will be, *on average*, lower with respect to the detection boxes corresponding to other players.

As a first step, we neglect all tracklets whose lifespan is less than a fixed *short-tracklets* threshold M_{short}. We have decided to exclude short tracklets from the computation in order to reduce the computational effort. Then, for *every* detection box, and for *every timestamp*, we compute the Euclidean distance between its position in the field and the position returned by the sensor in the visible field. Finally, we take



	1	3	9	2	8	4	5	7	6	10	12	11	13	14	15	16	17	18	20	23
2	456	502	352	667	683	385	681	808	211	790	960	412	437	450	403	207	694	70	1147	619
5	278	244	235	546	504	192	548	595	39	606	675	286	237	317	192	293	541	206	938	538
7	381	427	326	674	344	298	502	589	198	550	598	420	187	253	54	308	426	331	986	544
8	570	644	447	834	653	462	830	956	286	913	929	566	454	293	366	28	795	269	1380	864
9	413	357	480	389	763	422	449	570	468	634	1026	209	548	770	606	607	615		698	255
10	159	195	398	212	673	221	387	303	483	447	695	56	447	716	551	591	562	376	558	47
15	412	278	389	560	407	464	41	238	532	160	301	547	313	650	533	805	305	875	449	302
1	596	409	439	821	166	595	254	448	547	252	137	687	292	541	471	744	76	837	775	498
14		284	218		609		435	543	257	537	650		301	578	421		518		870	344
13		397	308		711		544	703	320	673			459	622	522		630		1023	474

Figure 3.36: The matrix showing the *assignment cost*, that is the average Euclidean distance between the position of a tracklet (*Detection Box ID*) and the the position of the sensor (*Sensor ID*), over the lifespan of a tracklet. The darker the color, the higher the assignation cost. If an entry is not present, then it means that a player is not inside the visibility cone of the camera when a tracklet exists.

the average, thus obtaining a final cost. Defining the **average euclidean distance** of a tracklet i for its entire lifespan $t = 1, ..., t_n$, with respect to the position of the player denoted with the tag_id j, we compute it as:

$$\bar{d}(i.j) = \frac{1}{n}\sum_{t=1}^{n}\sqrt{(x^{(t)}_{i,\ det} - x^{(t)}_{j,\ sen})^2 + (y^{(t)}_{i,\ det} - y^{(t)}_{j,\ sen})^2} \tag{3.45}$$

All these terms just obtained can be interpreted as the entries of a certain matrix $D(j, i)$, whose rows refer to the tag_id player, while the column denotes the detection box id. The matrix is represented in Fig. 3.36, and each entry is in Pixel unit: it can be converted to meters by dividing by the factor of $R_{x,y}$, which we recall to be $\sim 21[Px/m]$. This entry is NaN if a given detection box does not share any index with the data provided by a sensor: in reality, what happens is that the player is not visible for the entire lifespan of the detection box.

Let us now define the **assignation rule** for tracklets. For every column, that is *every detection box*, we assign the sensor which exhibits the minimum cost $\bar{d}(i, j)$ and is *less* than a fixed threshold. For our case, this simply means we keep the minimum value of every column, finally obtaining what is shown in Fig. 3.37. If we inspect such a matrix by rows, the only null values are the detection boxes that are valuable candidates for the assignment. As one can see, the pairs (tag_id $\longleftrightarrow$ detBox_id) we have mentioned in Section 3.2 pag. 52 after a simple visual inspection are present.

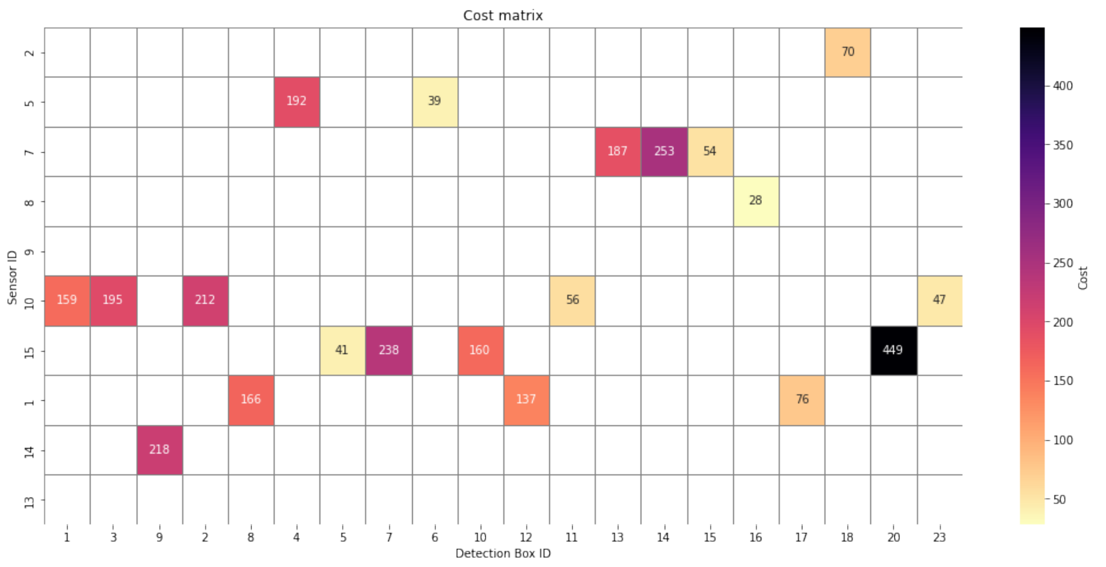


Figure 3.37: The final dataframe only with the *minimum cost* of every column. The rows refer to players' sensors id (*Sensor ID*), while the columns refer to the id uniquely identifying a detection box (*Detection Box ID*).

In fact, the row with index 15 has candidate detection boxes $5, 7, 10$. We are pretty confident that the correct assignment would be with detection box #5: in fact, it is the minimal of the three values proposed. Such an average distance, in meters, would be $\sim 2m$ which, if compared to $\sim 12m$ and $\sim 8m$, obviously is a much better value. It is also comparable to the uncertainty of the sensor: defining the *error* on the x and y-direction with, respectively, $\sigma_x = 1m$ and $\sigma_y = 1m$ the error introduced when computing the Euclidean distance can be computed as:

$$\sigma_{\mathbf{r}} = \sqrt{\sigma_x^2 + \sigma_y^2} \ \sim 1.4m \ \sim 30Px \tag{3.46}$$

assuming that, ideally, there is no error on the detection box position. Using this approach, if we consider the player with the sensor #5, a close call in this matching is the detection box #6, although the box #4 is also a candidate.

Formally, we want to assign to every sensor i the detection box j that minimizes the average distance $\bar{d}(i,j)$ *and* is less than a fixed **threshold** M_d, thus obtaining the set of detection boxes B_i that tracks a given sensor:

$$B_i = \{\ j \mid \underset{j}{\operatorname{argmin}}\ \bar{d}(i,j)\ \wedge\ \bar{d}(i,j) < M_d\} \tag{3.47}$$

However, this method leads to some problems that we need to further handle. In fact, after the assignation, the set B_i of detection boxes associated with a certain sensor i could contain *simultaneous* tracklets. To address this issue, we need to add another constraint. We know that the tracker detects a person *only once*, so it should not be

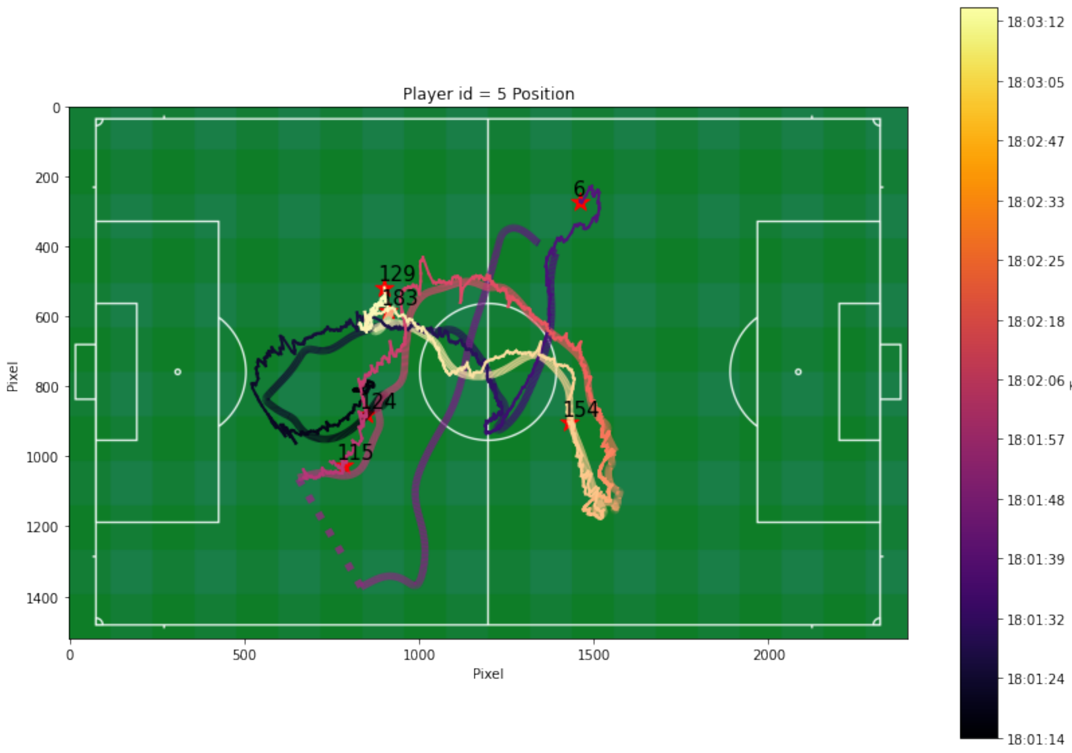


Figure 3.38: The trajectory reconstructed for the player with tag_id #5. The thicker curve is the *ground truth* (i.e., the sensor), while the thinner curves are the trajectories of the bounding boxes assigned to that sensor. Stars denote their id, and fragmented thicker trajectories are non-visible or missing data sensor.

possible for players to have two detection boxes assigned *at the same time* and for more than a given time. The condition for which two tracklets are considered *simultaneous* is that they exist for *a longer time* than a fixed *time interval* M_t. If two of them, belonging to the same set B_i, are simultaneous, then the *definitive* assignment would be the one with the least cost. In this way, the **final sets** D_i of the bounding boxes assigned to a given sensor i is obtained. This algorithm is first tested in the first 2 minutes of the video to evaluate its performance. We can see, as an example, that we are able to track a good part of the trajectory of the player wearing sensor with tag_id #5. (see Fig. 3.38). The Euclidean error between the sensor with tag_id #5 and the tracklets assigned to it, as a function of X and Y position, looks as in Fig. 3.39. We can see that, when the assignation is correct, the order of such an error is of the order of couple of *meters*, that is, comparable with respect to the one of the sensor. Finally, the **pseudocode** for the aforementioned algorithm takes the form listed in 2.

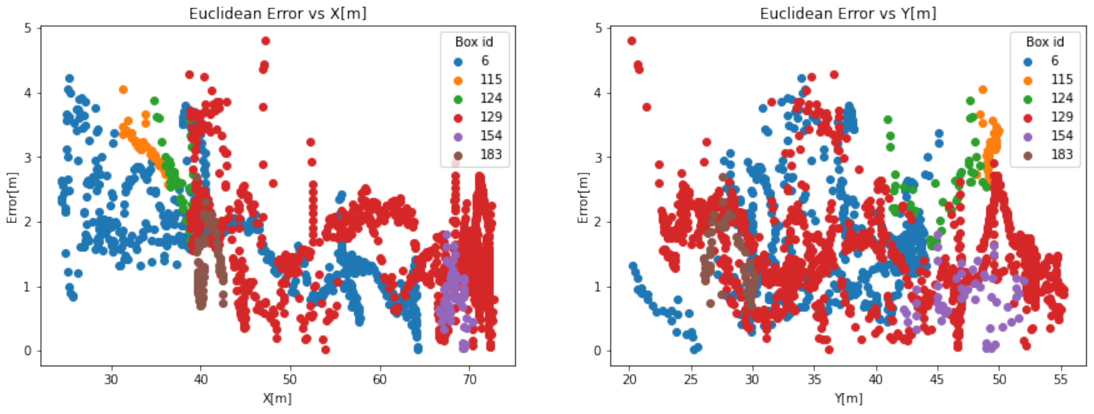
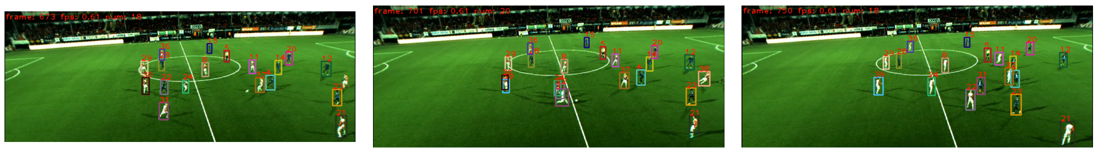


Figure 3.39: Euclidean error between the sensor #5 position and the corresponding boxes for every time instants, in function of X and Y.

Algorithm 2 Tracklet - Sensor assignment

Input: tracklets B, trajectories T, short-track M_{short}, distance M_d. simultanenous M_t thresholds
Output: Tracklets assigned to a sensor D

```
 1: $B_{long} \leftarrow$ LENGTH(B) $> M_{short}$
 2:
 3: /* Average Euclidean Distance */
 4: for DetBox $j$ in $B$ do
 5:     for sensor $i$ in $T$ do
 6:         $\bar{d}(i,j) \leftarrow$ AVERAGEEUCLIDEANDISTANCE(i,j)
 7:         CostMatrix(i,j) $\leftarrow \bar{d}(i,j)$
 8:     end for
 9: end for
10:
11: MinimaCostMatrix $\leftarrow$ KEEPCOLUMNSMINIMUM(CostMatrix)
12:
13: /* Assign DetBoxes to sensors */
14: for sensor $i$ in $T$ do
15:     $D_i \leftarrow$ ASSIGNATION(MinimaCostMatrix) $\wedge \bar{d}(i,j) < M_d$
16: end for
17:
18: /* Remove simultaneous tracklets */
19: for sensor $i$ in $T$ do
20:     for DetBox $B_j$, $B_k$,$j \neq k$ in $D_i$ do
21:         CommonIndexes($B_j, B_k$) $\leftarrow$ Indexes($B_j$) $\wedge$ Indexes($B_k$)
22:         if LENGTH(CommonIndexes($B_j, B_k$)) $> M_t$ then
23:             if $\bar{d}(i,j) < \bar{d}(i,k)$ then
24:                 $D_i \leftarrow D_i/\{B_k\}$
25:             else
26:                 $D_i \leftarrow D_i/\{B_j\}$
27:             end if
28:         end if
29:     end for
30: end for
```



(a) Frame 673. (b) Frame 601. (c) Frame 750.

Figure 3.40: The processed video is obtained as output of ByteTrack for the video of our dataset. The bounding boxes, which surround detected players, are uniquely identified by a number which is their ID. In particular, in this sequence of images, the IDs 22 and 31 are switched, this phaenomenon is known as **ID switch** and it is a well-known problem in MOT-task.

3.4 ByteTrack with re-identification head

Sometimes it happens that two detection boxes *swap* their identities, thus decreasing the performance of the tracking: *ID Switch* is a common and well-known problem in MOT task (see Fig. 3.40). However, when dealing with team sport image sequences, players wear jerseys of different colors according to the team they belong to; moreover, the two colors chosen by the regulation must be contrastive. Consequently, we think that if our tracker could also employ *visual* features, in the form of some embeddings returned by the detector, we could relieve this problem and improve the ID accuracy.

We propose here an approach inspired by [64] where, together with the usual YOLOX [14] detector head that addresses the tasks of bounding box regression and target classification, we add the head used in FairMOT [3] (see Fig. 2.16) to extract the feature embedding. The **head** architecture of this new "hybrid" detector will be essentially the same as in FairMOT, with the exception that the *detection-based* tasks are carried out by YOLOX, which theoretically should guarantee better performances.

Training is done on the dataset provided by *MOT17Challenge* [15], using two-thirds of the training set provided by the authors. On the contrary, the evaluation is performed on the remaining third. In other words: two-thirds of each of the video sequences contained in the training set are devoted to training the algorithm, while the last one-third helps to provide an estimation according to some metrics (*ClearMOT*, *IDF1*), thus allowing a fair comparison between the two methods.

There are basically two approaches for training such a "hybrid" network, where separate branches fulfill different tasks. In the first one, we take the pre-trained (or, alternatively, train ourselves) network for a specific task, and in a second step we train the other branch, which is devoted to the second task, without modifying the weights of the already trained one. This would act as a sort of *transfer learning*. In contrast, the second approach is inspired by *FairMOT* [3], where both branches are trained at the same time with the aim of minimizing a general loss with contributions from both tasks.

The difference in the code with regard to the "vanilla" ByteTrack[5] YOLOX-based tracker mainly consists in also providing the network the total number of IDs contained in the training set. This information can be retrieved from the ground-truth files, computing the number of different IDs labels for every training sequence. In addition, the original ByteTrack loss has been changed: it now includes a contribution coming from the re-identification task Let us define the **uncertainty loss**, that balances the different contributions from ReID and detection tasks, according to that of [59]:

$$\mathcal{L}_{total} = \frac{1}{2}\left(\mathcal{L}_{det}e^{-w_1} + \lambda_{ReID}\ \mathcal{L}_{ReID}e^{-w_2} + (w_1 + w_2)\right) \tag{3.48}$$

where we introduced a *regularization* term λ_{ReID} between the two losses. The contribution of the detection loss is that of ByteTrack:

$$\mathcal{L}_{det} = L_{ij}^{cls} + \lambda L_{ij}^{reg} \tag{3.49}$$

While the ReID loss is taken from FairMOT:

$$\mathcal{L}_{ReID} = -\sum_{i=1}^{N}\sum_{k=1}^{K} \mathbf{L}^i(k)\ \log(\mathbf{p}(k)) \tag{3.50}$$

Another approach, instead, is simply to sum the two contributions: in this way we are not taking into account the uncertainty. As done in (3.48), we introduce a regularization term λ_{ReID}:

$$\mathcal{L}_{total} = \mathcal{L}_{det} + \lambda_{ReID}\mathcal{L}_{reID} \tag{3.51}$$

The final score used to assign the detection boxes to tracklets is the same as that of ByteTrack for IoU. Regarding the information provided by the feature *embeddings*, we instead calculate **cosine similarity** between the embedding of the last frame of every track and the proposed detection box. The two costs c_{ReID}, c_{IoU} are then simply combined using a weighted sum according to a parameter α that by default is set to $\alpha = 0.9$:

$$C_{tot} = \alpha c_{IoU} + (1 - \alpha)c_{ReID} \tag{3.52}$$

The final assignment is performed by the Hungarian algorithm using this combined cost.

3.5 Programming tools

For such a work, we have used the Python v.3.8.10 [65] programming language. In particular, we take advantage of the versatility of the Jupyter Notebook [66] to better interact and handle the data. The libraries we have mostly used are pandas [67] and

[5]we refer to a network as "vanilla", when we have not applied any change to it.

NumPy [68] for data management, the PyTorch [69] implementations of the YOLOX detector and FairMOT, ByteTrack for the MOT task. To extract and modify images, the OpenCV2 [61] Python library was used, while data visualization was performed with Matplotlib [70] and Plotly [71].
For the training we exploit two NVIDIA Tesla T4 GPUs, while the inference is performed using only one of them.

Results and Discussion

In this chapter, we present the results that we have obtained. In particular, we discuss the performance of the MOT tracking algorithm that we have implemented. Taking inspiration from the ByteTrack algorithm, which uses YOLOX for object detection, we added a reidentification (ReID) head in such a way that it would also exploit visual cues. This head is taken from another MOT algorithm, namely FairMOT. Training is carried out on the MOT17 dataset with two different losses: the uncertainty one, which naturally offers a balance between two separate tasks [59], and the one that simply sums the contributions coming from the two tasks. The final performance is evaluated for the original ByteTrack and for the different loss choice, with the result that the original (vanilla) ByteTrack performs better in terms of MOTA and IDF1, but worse in terms of MOTP.

We then show that using input videos at higher resolution has a better impact on the quality of detections, though the throughput is slightly decreased. Finally, we see that our implemented algorithm is able to properly assign tracklets to detection boxes for most of the time, meaning that we can successfully track players' movements on the pitch when they are inside the visible to the camera recording the event. This is done first for the two minutes of the match, to assess the goodness of our algorithm, and finally for half of the first half of the game, which is around 20 minutes of video.

4.1 ByteTrack with re-identification head

Let us now discuss the results we have obtained with the additional head devoted to reidentification (abbr. ReID) tasks, using MOT17 for training and evaluating. We start from ByteTrack, where object detection is performed by the YOLOX one-stage detector, and modify it by adding the head that the FairMOT algorithm uses for ReID tasks. Our goal is to track multiple objects not only exploiting motion information, as in the original ByteTrack, but also combining it with visual cues, which are extracted in the same way as FairMOT does.

To perform such an analysis, we take the small YOLOX network (i.e., *yolox-s*) [14] in its pretrained version on the MS-COCO dataset and train it on the MOT17Challenge [15]. We split the training data, that is, the only dataset for which we have ground-truth annotations, into the actual training set and the evaluation set with proportions, respectively, of two-thirds and one-third. The tracking algorithm has been implemented in three flavors:

1. **Vanilla**: it is essentially the ByteTrack network, which does not have any re-identification (ReID) added to YOLOX and does not exploit *any* visual cues, thus relying only on motion information;

2. **Sum**: it is ByteTrack that exploits also visual information. The YOLOX detector is modified by adding a reidentification head, and is trained in order to minimize the loss (3.51), that is the simply the sum of detection and reidentification loss.

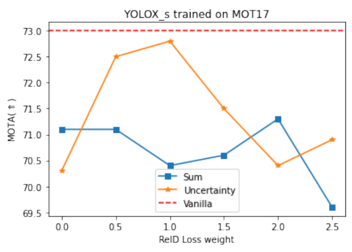
3. **Uncertainty**: it is ByteTrack that exploits also visual information. The YOLOX detector is modified by adding a reidentification head, but this time the total loss to minimize (3.48) balances the two losses related to the detection and reidentification.

We set the array that encodes visual features $\vec{x}_{vis} \in \Re^{128}$, and perform the training using different weights for the ReID regularization term $\lambda_{ReID} \in \{0.0, 0.5, 1.0, 1.5, 2.0, 2.5\}$. The training has taken approximately $\sim$ 4.5 hours in total on 2 NVIDIA Tesla T4 GPUs. Our results obtained in the evaluation set are those of Table 4.4.

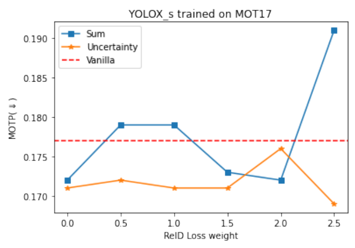
Let us now recall the $CLEAR\!-\!MOT$ metrics ($MOTA$, $MOTP$) which are defined, respectively, in (1.11) and (1.12) and are a metric for the *accuracy* and the *precision* of a MOT algorithm. The plot shows how MOTA and MOTP change in function of λ_{ReID}, which is a regularization term for the reidentification loss, and the different way of combining the two losses contributions, either summing (3.51), or using the uncertainty loss (3.48). The results are those shown in Fig. 4.41. They are compared with the *vanilla* model, that is, the usual small YOLOX model used by ByteTrack authors and trained with the same procedure. Since the vanilla model is independent of the ReID loss weight (i.e., λ_{ReID}), it is represented as a straight dashed line. From the results shown in Table 4.4, together with the Fig. 4.41, one can clearly see that the highest **MOTA** is reached when using the *vanilla* network. However, it is interesting to note that using the *uncertainty* loss leads to better results in terms of **MOTP**. Contrary to what one might expect, metrics related to trajectories and reidentification (i.e., **IDF1**, **ID SW**witches) are lower in the networks with the ReID head. This might be due to some hyperparameters that need to be tuned in a more appropriate way to efficiently combine the two costs, or, also, to the fact that using an additional head enlarges the number of parameters of the model, which, in turn, would need more data to be properly trained. It is clear that when an additional head is used, inference takes more time: this is why the *vanilla* model performs the fastest among all in terms of

TYPE	$\lambda_{\mathbf{ReID}}$	**MOTA** (⇑) [%]	**MOTP** (⇓)	**IDF1** (⇑) [%]	**ID SW.** (⇓)	**FPS** (⇑)
VAN	—	73.0	0.177	77.9	139	24.5
SUM	0.0	71.1	0.172	61.0	767	15.9
SUM	0.5	71.1	0.179	67.2	425	15.8
SUM	1.0	70.4	0.179	68.0	409	16.1
SUM	1.5	70.6	0.173	67.7	387	16.7
SUM	2.0	71.3	0.172	68.8	377	16.5
SUM	2.5	69.6	0.191	68.1	440	16.5
UNC	0.0	70.3	0.171	61.8	902	16.5
UNC	0.5	72.5	0.172	68.6	417	16.5
UNC	1.0	72.8	0.171	71.6	387	16.7
UNC	1.5	71.5	0.171	70.8	390	16.7
UNC	2.0	70.4	0.176	70.1	409	16.5
UNC	2.5	70.9	0.169	69.7	413	17.0

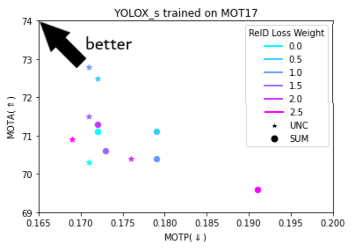
Table 4.4: The results we have obtained splitting the MOT17 dataset into the training set, two-thirds of the total samples, and the evaluation set, one-third of the total samples. The metrics we have considered are the MOTA (1.11), MOTP (1.12), IDF1 (1.15), ID SW (number of ID SWitches, see section 1.2.1), and FPS (frames per second, that is, the *throughput*) for different values of the regularization term λ_{ReID}, and choice of losses {SUM, UNC}, which are, respectively, equations (3.51) and (3.48). The model without reidentification head is denoted as VANILLA. The symbol (⇑)/(⇓) refers to a quantity such that the higher/lower, the better the model. Training is performed only using public detections $FRCNN$ to avoid overfitting, while the batch size is set to 16. For models using uncertainty loss, the parameters w_1 and w_2 are set, respectively, to -1.85 and -1.05, as they are the ones originally set in the FairMOT code [3].



(a) MOTA vs Identity Loss weight. The higher the MOTA, the better the model.



(b) MOTP vs Identity Loss weight. The lower the MOTP, the better the model.



(c) MOTP vs MOTA for different models with reidentification. The higher the MOTA, the lower the MOTP, the better the model

Figure 4.41: Comparison, in terms of MOTA (1.11) and MOTP (1.12), of the different models trained for different weights of the reidentification Loss λ_{ReID}. The training is performed only using public detections $FRCNN$ to avoid overfitting. In particular, the MOT17 dataset is split into the training and evaluation sets, which have, respectively, two-thirds and one-third of the total annotated samples. The batch size is set to 16. For models using uncertainty loss, the parameters w_1 and w_2 are set, respectively, to -1.85 and -1.05, as they are the ones originally set in the FairMOT code [3].

Network	Pretrained Weights	Match Thr.	Track Thr.	Track Buffer
yolox_x	mot20	0.90	0.60	90

Table 4.5: The parameters and thresholds used for the tracking of players in the short ($\sim$ 2 minutes) and in the long videos ($\sim$ 22 minutes). The algorithm we have used for tracking is ByteTrack [2], and is listed in 1.

velocity.

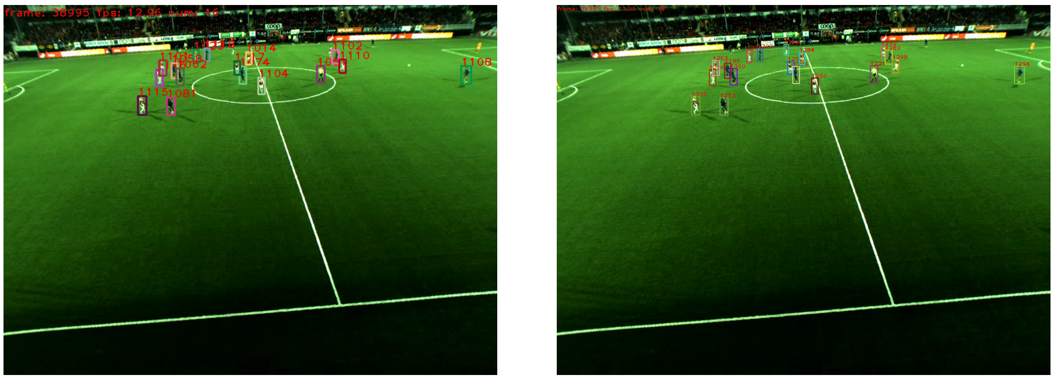
Finally, it should be noted that the approach proposed by Kendall, that is, using the uncertainty loss rather than simply summing the two contributions for detection and reidentification, generally leads to better results. Having said this, if we take into account both *MOTA* and *MOTP*, the best model with the ReID head is the one that uses **uncertainty loss** and $\lambda_{ReID} = 1.0$.

4.2 Tracking and assignation algorithm

Let us now show what we obtain when we analyze the detector performance as a function of the input video specifications, that is, how and whether it is affected by the resolution of the input video. Later, we test our algorithm to assign players tracked to the XYZ sensor data on the very first part of the match we are considering, namely, on the first two minutes of video. This has been done to provide information on the performance of the algorithm. Then, after we have found that most of the players have been successfully tracked, we run our algorithm on a longer video, that is, the very first $\sim$ 22 minutes of the football match.

Effects of video resolution

Following the instructions provided by the authors of the dataset in the related paper [5], we merge the video fragments at two different resolutions for a total of $\sim$ 22 minutes of video: the one of the camera (i.e., 1280 $\times$ 960), and an upsampled one by a factor 2 (2560 $\times$ 1920). For these two resolutions, we perform the tracking with ByteTrack using the two videos as input, choosing the set of parameters shown in the Table 4.5. We run the tracking in such a way that it would also return the “tracked” videos as output, that is, the videos showing the detection boxes. Although an obvious effect is to slow down the inference speed, we also need some visual information to assess the goodness of the detector, that is, how many people are visible in the image (*Ground Truth*), how many are detected (*True Positive*), how many are missed (*False Negative*), how many are counted twice (*False Positive*). Without visual information, as the one shown in Fig. 4.42, this benchmark would not be possible. We randomly extract 50 frames from the complete video and, by hand, notate the aforementioned quantities, finally averaging them over the total number of frames. The metrics are



(a) Low resolution video. **(b)** High resolution video.

Figure 4.42: The same frame, chosen randomly from the video we are analyzing [5], with different resolutions: the one used by the camera (1280 × 960), and the upsampled (2560 × 1920). The colored boxes, which surround players, are the bounding boxes returned as output by the detector, while the number associated is returned by the MOT algorithm that, for this case, is ByteTrack. In the top left corner it is visible the number of frame, the number of objects detected and the throughput (i.e., frames per second) of the algorithm.

computed as follows:

$$\text{Accuracy} = \frac{\#\text{ people detected}}{\#\text{ people visible}} \tag{4.53}$$

$$\text{Precision} = \frac{\#\text{ people detected}}{\#\text{ people visible} + \#\text{ people not detected} + \#\text{ double detections}} \tag{4.54}$$

The results returned by this analysis are those shown in the Table 4.6.

Resolution	**Av. Accuracy** (⇑)	**Av. Precision** (⇑)	**Av. FPS** (⇑)
1280 × 960	76.5%	75.7%	12.98
2560 × 1920	78.7%	78.2%	11.92

Table 4.6: Detection results for videos at different resolutions. The average accuracy and precision are computed as in eq. (4.53) and eq. (4.54) over 50 frames, randomly selected from the ∼ 22 minutes of video. For every frame, we annotate by hand the number of visible players, how many of them have been detected, missed, and detected more than once. The average FPS is computed as the average, over 50 frames, of the number of frames processed per second by ByteTrack (this number is visible in the top left corner of the output, see Fig. 4.42). The symbol (⇑) denotes a quantity that the higher, the better.

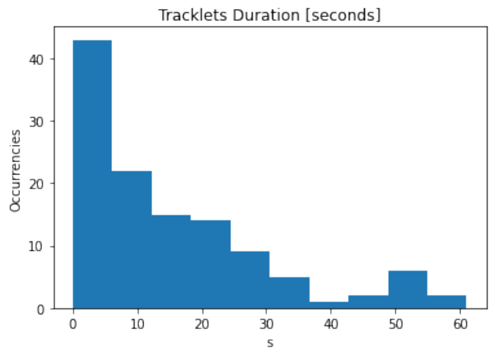
Short video

We can proceed to perform the tracking of the players using the video at a higher resolution, since we are more interested in having better performances rather than a *real-time* inference. We will now discuss the output of our assignation algorithm for the first 2 minutes of video: we want, in first place, to evaluate the algorithm before applying it to the complete one ($\sim$ 22 minutes).

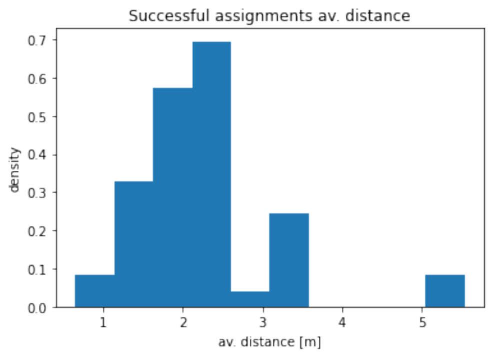
The histogram with the distribution of the length of tracklets for this first short video fragment is the one drawn in Fig. 4.43a. Using the assignation rule we have introduced, instead, leads to the following *average cost distribution* (see Fig. 4.43b), when using the set of thresholds listed in Table 4.7. The bar graph with the ratio

Short Track Thr.	**Av. Distance Thr.**	**Simultaneous Thr.**
0.5s	120Px	3s

Table 4.7: The different thresholds used in our algorithm for assignation of the bounding box to the sensors (see Algorithm 2). The short track threshold denotes the amount of seconds for which a tracklet is considered short and, therefore, should be discarded. The average distance threshold, instead, denotes the maximum average distance in Pixel between a tracklet and the trajectory of a sensor, according to which an assignation is considered valid. The last threshold denotes the amount of seconds that two tracklets must coexist to consider them as simultaneous.



(a) Tracklet duration occurrence distribution.



(b) Average assignation cost density distribution. The mean value is 2.2 ± 0.9.

Figure 4.43: Histograms, for the first two minutes of video, showing the tracklet duration counts and the average cost distribution for successful assignments of tracklets to sensor trajectories. Tracklets are found using ByteTrack, while the assignations are performed via the Algorithm 2.

of how long a player is tracked over the total time when inside the visibility cone is that of Fig. 4.44. It turns out that a player is, *on average*, tracked $80 \pm 16\%$ of the time when he is *visible* in the first two minutes of the match. We present here, besides the *successful* assignment we have already shown in Fig. 3.38, another result of the assignation algorithm, namely, the one corresponding to the player with the sensor #2 (see Fig. 4.45). The stacked plots of the positions for every player *missed* (i.e., not detected) or *unassigned* are those in Fig. 4.46 and Fig. 4.47

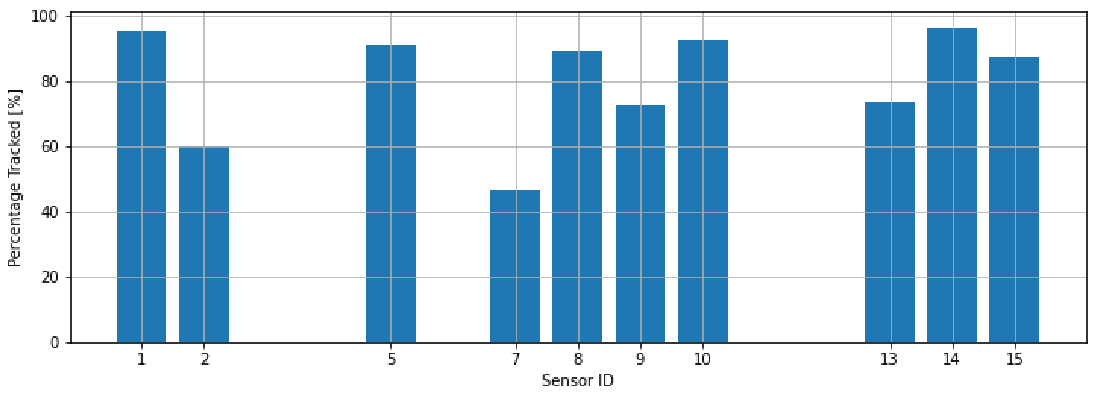


Figure 4.44: Bar chart showing the percentage of time a player, marked by the sensor ID (which is another name for the tag_id), has an assigned tracklet compared to the total time he is within the camera field of view. The graph refers to the first two minutes of the video, the tracklets are found using ByteTrack and the assignment is done through the Algorithm 2. On average, we obtain that a visible player is tracked $80 \pm 16\%$ of the time he is visible.

Long video

Since the results we have obtained on the two-minute video are satisfactory to us, we can now perform the tracking through ByteTrack, and the assignation of tracklets to sensor trajectories via the Algorithm 2 we have implemented. The parameters used are the same as before, namely the ones listed in 4.7, and for the creation of the long video, we take 450 fragments of 3 seconds, which roughly corresponds to *half* duration of a half-time in the soccer game.

The histogram showing the count distribution of the duration of the tracklets is the one in Fig. 4.48a, while the density distribution of the average costs for successful assignations is represented in Fig. 4.48. If a player is visible, that is, inside the visibility cone, we find that its trajectory is tracked for $59 \pm 11\%$ of the time. We then show the missed/non-assigned count histogram of the positions for every player in Fig. 4.50 and Fig. 4.51. Finally, we produce the *density* plot of these positions when a miss occurs, over the total times that a player is located at a certain position in the pitch coordinates (see Fig. 4.52). According to our findings, we note that **video resolution** could have an impact on the performance of our algorithm, especially in the detection part, which is fundamental for the subsequent tracking task: clearly, if a target is not detected, it cannot be tracked. It is interesting, however, that even though the camera has a given resolution we were able to perform the *upsampling* of a factor $2\times$. In doing so, the algorithm run with the same parameters returned different results, which are slightly better on the video with a *larger* resolution: both *average accuracy* and *average precision* increase by a couple of percentage points. On the other hand, this comes with a price, namely a slower inference time of $\sim 1fps$. Given that we are more interested in

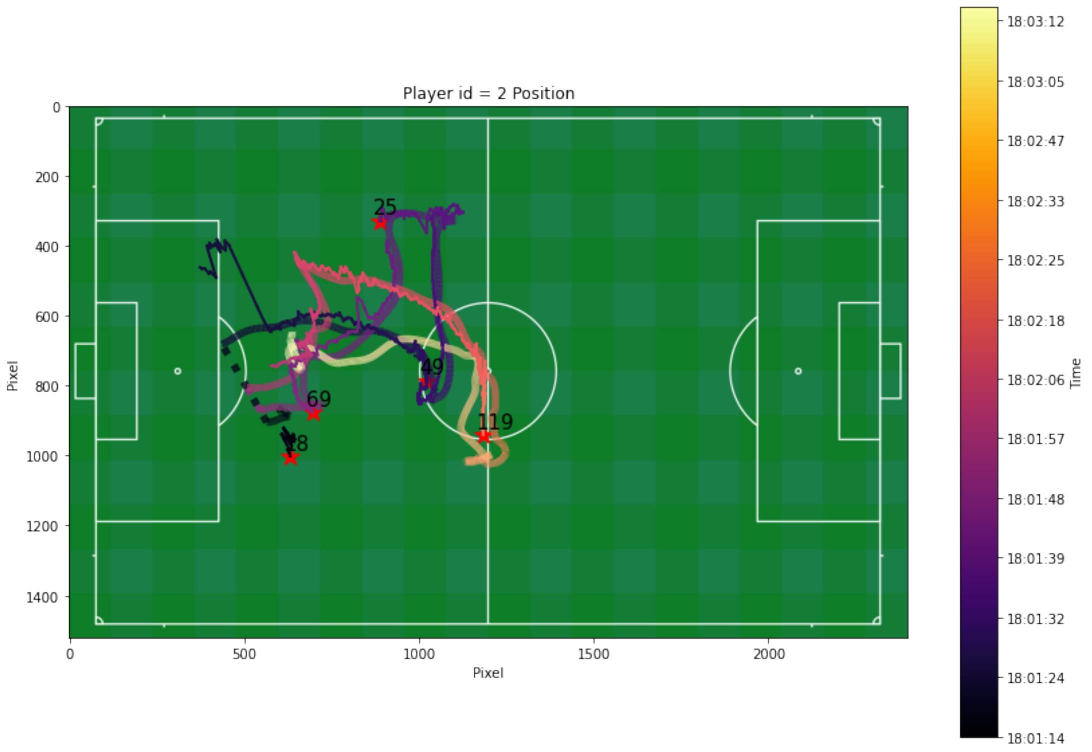


Figure 4.45: Tracklets assigned to player with tag_id #2 in first 2 minutes of video. Note the "jump" in the curve, which can be interpreted in the following way: a certain player has been detected and tracked at the beginning (the trajectory above), but some instants later his bounding box is transferred to the player wearing the sensor #2, who has just been detected. The thicker curve is the sensor data, while the thinner shows the tracklets from the assigned detection boxes. The color of the curves changes according to the timestamp: the darker the color, the earlier the time instant. The fragmentation for the thicker curve denotes that the sensor data are not available.

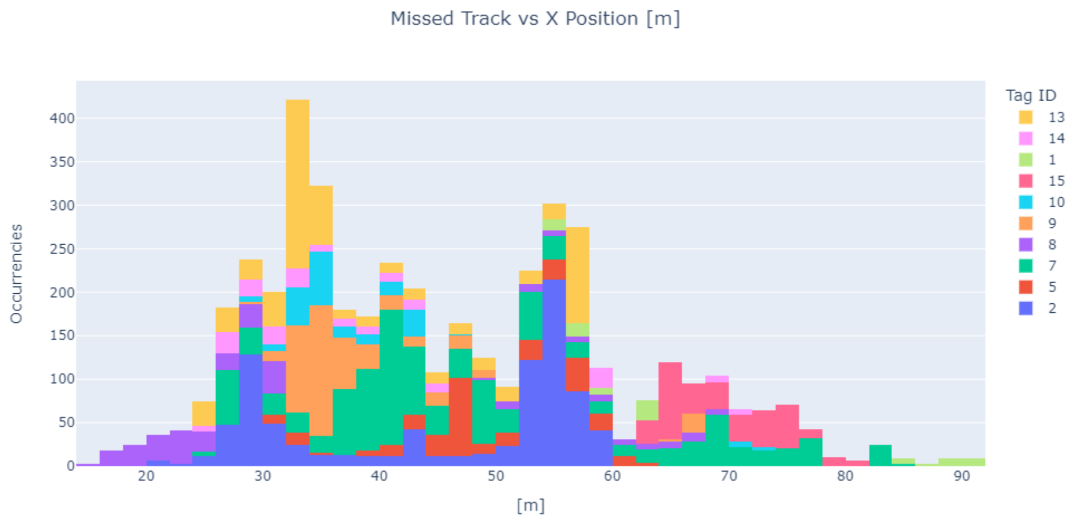


Figure 4.46: Histogram representing all positions along the X-axis of the sensors of players that either have not been detected by ByteTrack, or have no tracklet assigned, for the first two minutes of the video under investigation. Different colors refer to positions of different players, which are uniquely identified by their sensor ID (tag_id).

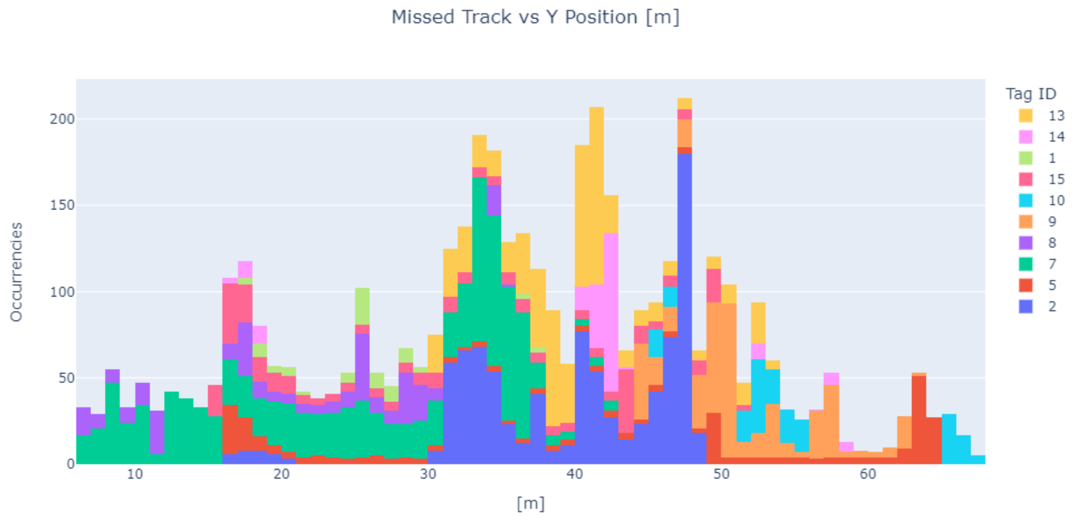
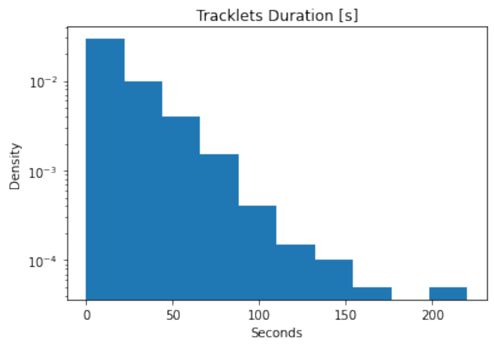
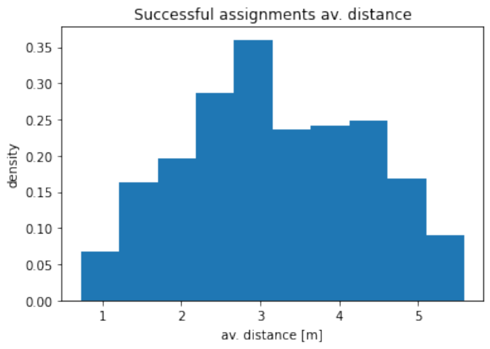


Figure 4.47: Histogram representing all positions along the Y-axis of the sensors of players that either have not been detected by ByteTrack, or have no tracklet assigned, for the first two minutes of the video under investigation. Different colors refer to positions of different players, which are uniquely identified by their sensor ID (tag_id).



(a) Tracklet duration density distribution. The exponent of the power law is $\gamma = -0.043 \pm 0.002$



(b) Average assignation density cost distribution. The mean value is 3.1 ± 1.1 m.

Figure 4.48: Histograms, for the the complete video ($\sim$ 22 minutes), showing the tracklet duration density distribution and the average cost density distribution for successful assignments of tracklets to sensor trajectories. Tracklets are found using ByteTrack, while the assignations are performed via the Algorithm 2.

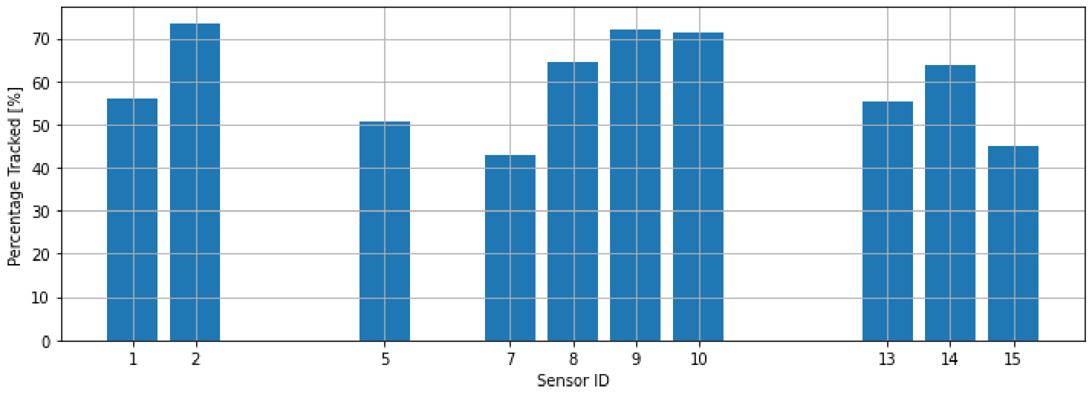


Figure 4.49: Bar chart showing the percentage of time a player, marked by the sensor ID (which is another name for the tag_id), has an assigned tracklet compared to the total time he is within the camera field of view. The graph refers to the complete video, which lasts $\sim$ 22 minutes, the tracklets are found using ByteTrack and the assignment is done through the Algorithm 2. On average, we obtain that a visible player is tracked $59 \pm 11\%$ of his trajectory.

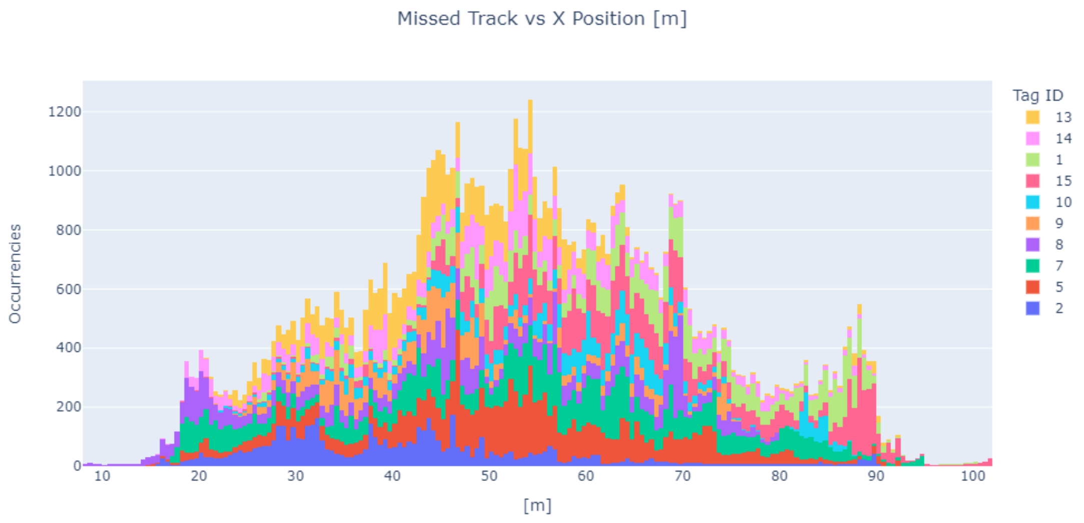


Figure 4.50: Histogram representing all positions along the X-axis of the sensors of players that either have not been detected by ByteTrack, or have no tracklet assigned, for the complete video ($\sim$ 22 minutes) under investigation. Different colors refer to positions of different players, which are uniquely identified by their sensor ID (tag_id).

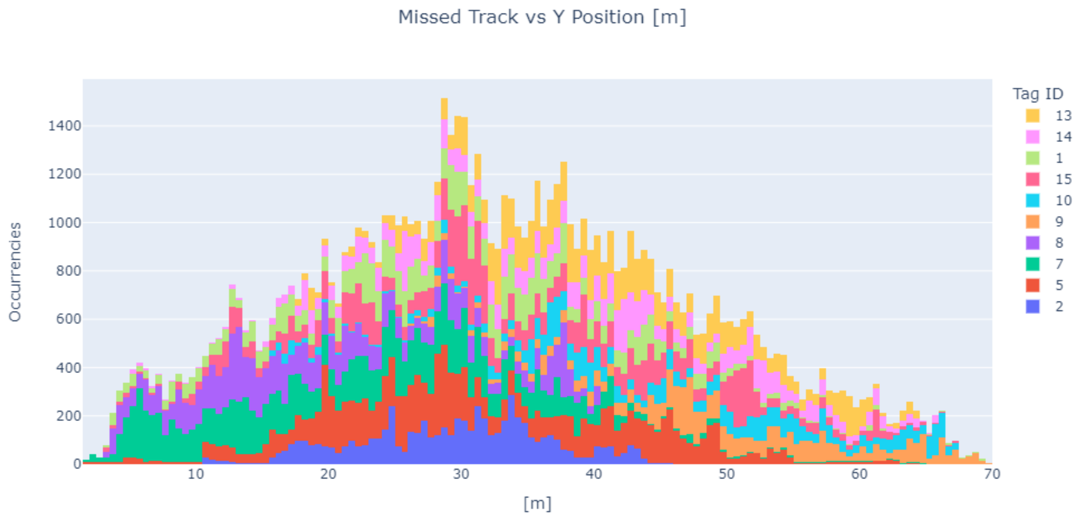


Figure 4.51: Histogram representing all positions along the Y-axis of the sensors of players that either have not been detected by ByteTrack, or have no tracklet assigned, for the complete video ($\sim$ 22 minutes) under investigation. Different colors refer to positions of different players, which are uniquely identified by their sensor ID (tag_id).

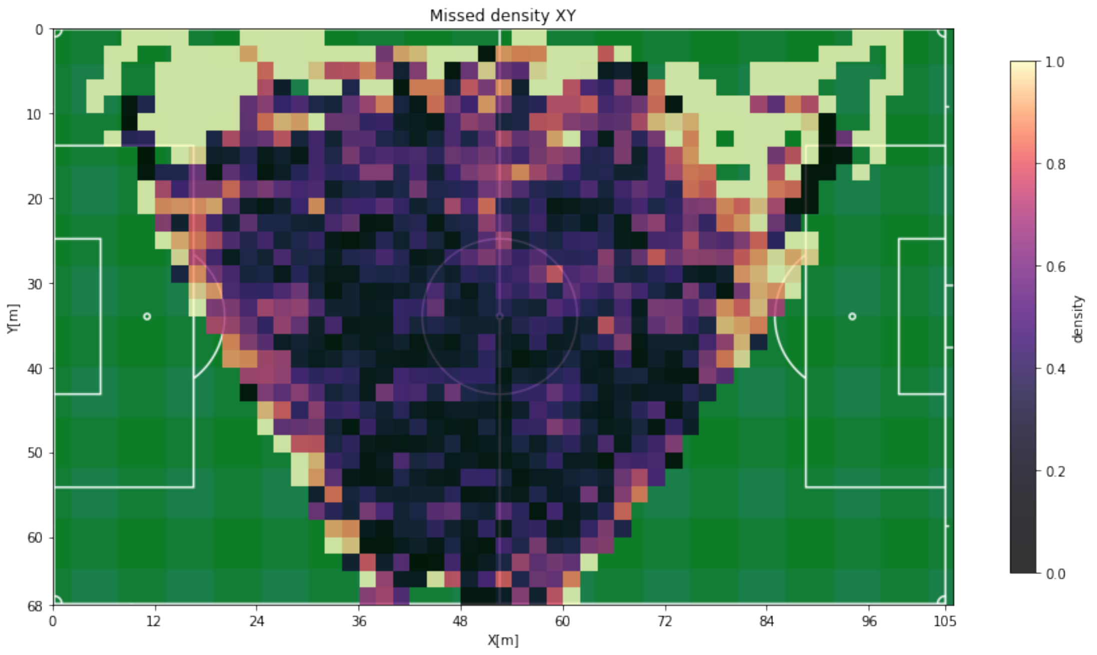


Figure 4.52: Density plot of the positions of misses, normalized to all the players position for the Home team. A "miss" is defined when a player, that is inside the visibility cone, has not been neither seen nor assigned a tracklet. This graph refers to the complete video, which lasts $\sim$ 20 minutes, the tracklets are found via ByteTrack, while the assignation of tracklets to sensors trajectories is performed using Algorithm 2.

the performances in terms of *accuracy/precision* rather than *inference speed*, we have chosen to continue with the video at higher resolution, according to Table 4.6.

With regards to the **short video** of 2 minutes, we see that most of the tracklets are very short, that is, less than 10 seconds (see Fig. 4.43a). However, there exist even tracklets that are greater than 40 seconds in length. Using the duration of such a short video as a reference, the lifespan of the tracklets can reach even the 50% of the overall duration, which is a very good result. However, we must take into account that the field of visibility sets a constraint on tracklets: if a player traverses the pitch very fast from one side to the other side, his tracklet will last a few seconds, even if he has been tracked for the entire path. Therefore, short tracklets do not necessarily imply that the results are not good, but long tracklets can be considered a signal of good results.

The average distance distribution for a *successful assignment* of Fig. 4.43b shows that most assignments are made when the average distance is less than 2.2 ± 0.9, which is comparable to sensor error. Furthermore, inspecting the visual comparisons we have presented in Fig. 3.38 and Fig. 4.45, we see that generally the assigned tracklets correspond to their target and the algorithm satisfactorily performs the task. However, in the latter image, it is to be noted that an ID swap occurs: at the beginning of the tracklet there is a straight line. Indeed, in the output video, the assigned bounding box initially is tracking a player, while some instants later is transferred to another one. However, the subsequent tracklet almost perfectly resembles the target sensor.

Despite this problem, the proposed algorithm assigns a bounding box $80 \pm 16\%$ of the time (see Fig. 4.44) to players inside the visibility cone: some of them even are tracked almost all the time. According to the metrics we have introduced in Section 1.2.3, we can state that 4 players have been **partially tracked**, while the remaining 6 have been **mostly tracked**, out of a total of 10 players of which we have the sensor position available. This is comparable to the detection performance of Table 4.6, although we must remember that the latter results refer to *all* players, without any team distinction.

The **histogram** with the sensor positions of **missed/unassigned tracklets** counts, instead, shows that our algorithm encounters some difficulties in the middle field position, but this can be explained by stating that most of the time the game develops exactly there. In fact, in the middle of the pitch we expect to observe the highest density of players for both teams. Hence, for a detector, it might be more difficult to work when the density of people is higher, whereas we may expect ID swap between two different players that are moving towards the same direction, or they make contrasts to fight for the ball or keep positions. Also: it is clear that when the density of players is higher, the *number* of misses will tend to be higher. Another interesting point is that, with reference to Fig. 4.47, we have the fewest misses closer to the camera.

Furthermore, many of them occur at the left boundary of the field of view: this can be explained first by saying that this statistics only refers to the first two minutes of the video, in which the game almost totally developed on the left side, and even one or two missed trajectories might bias the result. For a more significant result, we need to consider the data coming from the longer video.

Let us now discuss the **final results** of the **20 minutes video**, to assess the quality of our tracking algorithm. According to Fig. 4.48a, the distribution density of the duration of the tracklets appears to follow a power law with an exponent $\gamma = -0.043 \pm 0.002$. The average distance for the successful assignments (see Fig. 4.48b) does not seem to follow any particular distribution and has mean and standard deviations equal to 3.1 ± 1.1 meters. Such a mean value is slightly larger than the previous one, but is still acceptable and comparable, though twice larger, than the sensor error. Although the overall result (see Fig. 4.49) is worse than the previous one, it is satisfactory: a generic player is tracked on average $\sim 60\%$ of the time, that is, more than half of its trajectory when visible. According to the metrics we have introduced in Section 1.2.3, *all* 10 players are **partially tracked**.

Our algorithm seems to fail the most in the middle of the field according to Fig. 4.50 and Fig. 4.51, that is, where we expect the players to be the most of the time. However, these graphs do not help much if we do not integrate this information with the **density plot** in Fig. 4.52, which we recall to be the total number of misses over the total number of events. Comparing them with the latter one, we can see that the players in the middle of the field are almost always tracked: therefore, the large values at the center of the count histograms follow from the fact that there we expect a larger density of players. Moreover, we note that *at boundaries* of the visibility cone there are the most misses. This may be due to some reasons. First, the parametrization of the boundaries of such a cone might not be very precise: some players might be considered *visible*, although they are actually outside the visibility field and cannot be detected. In second place, when a player runs into this cone, it may happen that he is not detected at the very first frames, but might take a while. Another important result is that it seems that the further we go with respect to the camera, the more difficult it is for the algorithm to detect/track the player. In fact, many misses occur in the corners of the pitch, which are opposite to the camera. It is also visible in Fig. 4.51, where we see a significant tail on the left side, and the part of the graph that lies on the left side is generally higher than its correspondent on the left. Besides the aforementioned difficulty in the detection, this can be explained by the fact that this quantity has not been normalized: the event counting depends on the surface of the pitch visible, which clearly tends to be larger when we go further from the camera.

Conclusions and Future Work

In this work, we presented the Multi-Object-Tracking problem, providing some formalism of it and the metrics used to compare different algorithms. Then, we discussed ByteTrack and FairMOT algorithms, in particular their architecture, how they work and are trained, also presenting the dataset we used for training: MOT17. Afterwards, we proceeded to describe our dataset, which consists of a video of a football match we use as input of our tracker, so to reconstruct players' trajectories thanks to this visual information. The dataset, besides such a video, also makes available the data of a XYZ sensor throughout the duration of the entire match. Then, we state the goal of this work, that is, implementing an algorithm to perform the assignation between the information obtained by the bounding boxes and the trajectories provided by the sensor, only after we have proceeded to do some preprocessing. In addition, to increase the quality of the ByteTrack tracker, we attached an additional re-identification head that also exploits visual cues.

We have found that this hybrid detector, under the same training conditions and with two different loss options, performs generally worse than the original ByteTrack. However, for a certain choice of regularization term $\lambda_{ReID} = 1.0$ the results are comparable. There might be two possible reasons for this: the first is that the reidentification head whose task is to create an embedding vector might increase the need for training data for the network. This can be solved by using the *whole* $MOT17$ dataset for training or by mixing different datasets, as ByteTrack does for competition networks. The second is simply an unfortunate choice of the hyperparameters (i.e., w_1, w_2) present in the *uncertainty* loss (2.33), whose role is to balance the contribution of the losses related to two different tasks, or in the definition of the assignation cost C_{tot} in (3.52), that is, try different values for the parameter α. This issue could be solved by having more time to perform training with different sets of hyperparameters.

We have been able to assign a detection box for $\sim 60 \pm 10\%$ of players' trajectories inside the visibility cone. This means that we are generally able to *partially* track players. Clearly, the upper bound for this metric is fixed by the goodness of the detector, which we experimentally found to be $\sim 78\%$, when using a resolution higher than the native of the camera. Adding one more parameter to perform the assignation and fusing this information with the position one could increase the algorithm performance. In fact, one might also want to take advantage of some of the remaining information provided by the sensor. For example, a immediate quantity that can be estimated using tracklets is the velocity. Speaking of the detector, we are pretty confident that the network would perform even better if provided with additional training data, specifically related to the domain of soccer and sports. However, this would need a non-negligible effort that is typical of producing labeled data.

Note that since we only have data from the sensor worn by the home team, which wears a jersey of a very distinct color with respect to the opponent team, it would be possible to perform the classification of detections according to the team to which players belong. This would result in an algorithm better tuned to track specifically and create statistics of only a certain team, while not considering the other one.

Finally, one might also want to use the other camera views of the same event. In other words, we may want to stitch the images from different views and perform the same preprocessing we have done; which consists of correcting the radial distortion and then applying homography. With the output, we could either use the tracking algorithm on the stitched panorama, or as an alternative, on the three different videos separately and then re-identify players in the different views. In this way, the entire field will be visible in its entirety, although we would expect some difficulties in detecting players that are far from the camera, as we have found. We could also use another dataset, made available by the same authors [5] of the one used for this work. It consists of a video recorded with a fish eye lens that, despite being really distorted, frames the whole pitch during a different match. Ideally, this would also allow us to neglect the sensor information, although the radial distortion should be treated with much more care.

In this work, we have presented an algorithm through which it is possible to track players using a football match video as input, and then draw their trajectories on the pitch. In this way, simply mounting a pole with a camera on top of it, if in a good position, would also allow a team with a low budget to obtain the statistics of its players by tracking them, their position, thus creating more elaborate analysis like sprint analysis, position heatmap, distance covered throughout the game, etc. We hope that this project will be a starting point for these types of applications.

References

[1] L. Leal-Taixé, A. Milan, I. Reid, S. Roth, and K. Schindler, "MOTChallenge 2015: Towards a benchmark for multi-target tracking," *arXiv:1504.01942 [cs]*, Apr. 2015, arXiv: 1504.01942. [Online]. Available: http://arxiv.org/abs/1504.01942

[2] Y. Zhang, P. Sun, Y. Jiang, D. Yu, Z. Yuan, P. Luo, W. Liu, and X. Wang, "Byte-track: Multi-object tracking by associating every detection box," *arXiv preprint arXiv:2110.06864*, 2021.

[3] Y. Zhang, C. Wang, X. Wang, W. Zeng, and W. Liu, "Fairmot: On the fairness of detection and re-identification in multiple object tracking," *International Journal of Computer Vision*, vol. 129, pp. 3069–3087, 2021.

[4] P. Dendorfer, H. Rezatofighi, A. Milan, J. Shi, D. Cremers, I. Reid, S. Roth, K. Schindler, and L. Leal-Taixé, "Mot20: A benchmark for multi object tracking in crowded scenes," *arXiv:2003.09003[cs]*, Mar. 2020, arXiv: 2003.09003. [Online]. Available: http://arxiv.org/abs/1906.04567

[5] S. A. Pettersen, P. Halvorsen, D. Johansen, H. Johansen, V. Berg-Johansen, V. R. Gaddam, A. Mortensen, R. Langseth, C. Griwodz, and H. K. Stensland, "Soccer video and player position dataset," in *Proceedings of the 5th ACM Multimedia Systems Conference on - MMSys '14*. ACM Press, 2014. [Online]. Available: https://doi.org/10.1145/2557642.2563677

[6] Y. Deng, P. Coen, M. Sun, and J. W. Shaevitz, "Efficient multiple object tracking using mutually repulsive active membranes," *PLoS ONE*, vol. 8, no. 6, p. e65769, Jun. 2013. [Online]. Available: https://doi.org/10.1371/journal.pone.0065769

[7] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *2012 IEEE Conference on Computer Vision and Pattern Recognition*, 2012, pp. 3354–3361.

[8] [Online]. Available: https://www.sportperformanceanalysis.com/article/artificial-intelligence-ai-in-sports

[9] W. Luo, J. Xing, A. Milan, X. Zhang, W. Liu, and T.-K. Kim, “Multiple object tracking: A literature review,” *Artificial Intelligence*, vol. 293, p. 103448, Apr. 2021. [Online]. Available: https://doi.org/10.1016/j.artint.2020.103448

[10] J. Luiten, A. Osep, P. Dendorfer, P. Torr, A. Geiger, L. Leal-Taixé, and B. Leibe, “Hota: A higher order metric for evaluating multi-object tracking,” *International Journal of Computer Vision*, vol. 129, no. 2, p. 548–578, Oct 2020. [Online]. Available: http://dx.doi.org/10.1007/s11263-020-01375-2

[11] P. Jaccard, “THE DISTRIBUTION OF THE FLORA IN THE ALPINE ZONE.1,” *New Phytologist*, vol. 11, no. 2, pp. 37–50, Feb. 1912. [Online]. Available: https://doi.org/10.1111/j.1469-8137.1912.tb05611.x

[12] D. Helbing and P. Molnár, “Social force model for pedestrian dynamics,” *Physical Review E*, vol. 51, no. 5, pp. 4282–4286, May 1995. [Online]. Available: https://doi.org/10.1103/physreve.51.4282

[13] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” *CoRR*, vol. abs/1506.02640, 2015. [Online]. Available: http://arxiv.org/abs/1506.02640

[14] Z. Ge, S. Liu, F. Wang, Z. Li, and J. Sun, “Yolox: Exceeding yolo series in 2021,” *arXiv preprint arXiv:2107.08430*, 2021.

[15] A. Milan, L. Leal-Taixé, I. Reid, S. Roth, and K. Schindler, “MOT16: A benchmark for multi-object tracking,” *arXiv:1603.00831 [cs]*, Mar. 2016, arXiv: 1603.00831. [Online]. Available: http://arxiv.org/abs/1603.00831

[16] M. G. Kendall, “A new measure of rank correlation,” *Biometrika*, vol. 30, no. 1/2, p. 81, Jun. 1938. [Online]. Available: https://doi.org/10.2307/2332226

[17] E. Itskovits, A. Levine, E. Cohen, and A. Zaslaver, “A multi-animal tracker for studying complex behaviors,” *BMC Biology*, vol. 15, no. 1, Apr. 2017. [Online]. Available: https://doi.org/10.1186/s12915-017-0363-9

[18] W.-L. Lu, J.-A. Ting, J. J. Little, and K. P. Murphy, “Learning to track and identify players from broadcast sports videos,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 7, pp. 1704–1716, Jul. 2013. [Online]. Available: https://doi.org/10.1109/tpami.2012.242

[19] M. Manafifard, H. Ebadi, and H. A. Moghaddam, “A survey on player tracking in soccer videos,” *Computer Vision and Image Understanding*, vol. 159, pp. 19–46, Jun. 2017. [Online]. Available: https://doi.org/10.1016/j.cviu.2017.02.002

[20] K. Bernardin and R. Stiefelhagen, “Evaluating multiple object tracking performance: The CLEAR MOT metrics,” *EURASIP Journal on Image and*

Video Processing, vol. 2008, pp. 1–10, 2008. [Online]. Available: https://doi.org/10.1155/2008/246309

[21] E. Ristani, F. Solera, R. Zou, R. Cucchiara, and C. Tomasi, "Performance measures and a data set for multi-target, multi-camera tracking," in *Lecture Notes in Computer Science*. Springer International Publishing, 2016, pp. 17–35. [Online]. Available: https://doi.org/10.1007/978-3-319-48881-3_2

[22] F. Farina, D. Fontanelli, A. Garulli, A. Giannitrapani, and D. Prattichizzo, "Walking ahead: The headed social force model," *PLOS ONE*, vol. 12, no. 1, p. e0169734, Jan. 2017. [Online]. Available: https://doi.org/10.1371/journal.pone.0169734

[23] S. K. Pal, A. Pramanik, J. Maiti, and P. Mitra, "Deep learning in multi-object detection and tracking: state of the art," *Applied Intelligence*, vol. 51, no. 9, pp. 6400–6429, Apr. 2021. [Online]. Available: https://doi.org/10.1007/s10489-021-02293-7

[24] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval Research Logistics Quarterly*, vol. 2, no. 1-2, pp. 83–97, Mar. 1955. [Online]. Available: https://doi.org/10.1002/nav.3800020109

[25] J. Munkres, "Algorithms for the assignment and transportation problems," *Journal of the Society for Industrial and Applied Mathematics*, vol. 5, no. 1, pp. 32–38, 1957. [Online]. Available: https://doi.org/10.1137/0105003

[26] M. Jaward, L. Mihaylova, N. Canagarajah, and D. Bull, "Multiple object tracking using particle filters," in *2006 IEEE Aerospace Conference*. IEEE, 2006. [Online]. Available: https://doi.org/10.1109/aero.2006.1655926

[27] L. Jiao, F. Zhang, F. Liu, S. Yang, L. Li, Z. Feng, and R. Qu, "A survey of deep learning-based object detection," *IEEE Access*, vol. 7, pp. 128 837–128 868, 2019. [Online]. Available: https://doi.org/10.1109/access.2019.2939201

[28] "Visual tracking: An experimental survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, no. 7, pp. 1442–1468, Jul. 2014. [Online]. Available: https://doi.org/10.1109/tpami.2013.230

[29] R. Szeliski, *Computer Vision*. Springer International Publishing, 2022. [Online]. Available: https://doi.org/10.1007/978-3-030-34372-9

[30] A. Bhattacharyya, "On a measure of divergence between two statistical populations defined by their probability distributions," 1943.

[31] J. Han, M. Kamber, and J. Pei, "Getting to know your data," in *Data Mining*. Elsevier, 2012, pp. 39–82. [Online]. Available: https://doi.org/10.1016/b978-0-12-381479-1.00002-2

[32] R. E. Kalman, "A new approach to linear filtering and prediction problems," 1960.

[33] R. R. L. Jr, "Kalman and bayesian filters in python," https://github.com/rlabbe/Kalman-and-Bayesian-Filters-in-Python, 2020.

[34] S. J. Julier and J. K. Uhlmann, "New extension of the kalman filter to nonlinear systems," in *SPIE Proceedings*, I. Kadar, Ed. SPIE, Jul. 1997. [Online]. Available: https://doi.org/10.1117/12.280797

[35] P. S. Maybeck, *Stochastic models: V. 1*, ser. Mathematics in Science and Engineering. San Diego, CA: Academic Press, Sep. 1979.

[36] A. Rosebrock, "Intersection over union as a similarity measure for object detection on images - an important task in computer vision." cC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=57718561. [Online]. Available: https://pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/

[37] C. Heindl, "py-motmetrics," https://github.com/cheind/py-motmetrics, 2022.

[38] L. Henderson, "On the fluid mechanics of human crowd motion," *Transportation Research*, vol. 8, no. 6, pp. 509–515, Dec. 1974. [Online]. Available: https://doi.org/10.1016/0041-1647(74)90027-6

[39] S. Okazaki, "A study of simulation model for pedestrian movement with evacuation and queuing," 1993, original article in Japanese, 1979.

[40] O. Biham, A. A. Middleton, and D. Levine, "Self-organization and a dynamical transition in traffic-flow models," *Physical Review A*, vol. 46, no. 10, pp. R6124–R6127, Nov. 1992. [Online]. Available: https://doi.org/10.1103/physreva.46.r6124

[41] D. Helbing, "Boltzmann-like and boltzmann-fokker-planck equations as a foundation of behavioral models," *Physica A: Statistical Mechanics and its Applications*, vol. 196, no. 4, pp. 546–573, Jul. 1993. [Online]. Available: https://doi.org/10.1016/0378-4371(93)90034-2

[42] J. Redmon and A. Farhadi, "YOLO9000: better, faster, stronger," *CoRR*, vol. abs/1612.08242, 2016. [Online]. Available: http://arxiv.org/abs/1612.08242

[43] ——, "Yolov3: An incremental improvement," *CoRR*, vol. abs/1804.02767, 2018. [Online]. Available: http://arxiv.org/abs/1804.02767

[44] X. Long, K. Deng, G. Wang, Y. Zhang, Q. Dang, Y. Gao, H. Shen, J. Ren, S. Han, E. Ding, and S. Wen, "PP-YOLO: an effective and efficient implementation of object detector," *CoRR*, vol. abs/2007.12099, 2020. [Online]. Available: https://arxiv.org/abs/2007.12099

[45] P. Sun, J. Cao, Y. Jiang, R. Zhang, E. Xie, Z. Yuan, C. Wang, and P. Luo, "Transtrack: Multiple object tracking with transformer," 2020. [Online]. Available: https://arxiv.org/abs/2012.15460

[46] T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: common objects in context," *CoRR*, vol. abs/1405.0312, 2014. [Online]. Available: http://arxiv.org/abs/1405.0312

[47] A. Krizhevsky, "One weird trick for parallelizing convolutional neural networks," 2014. [Online]. Available: https://arxiv.org/abs/1404.5997

[48] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015. [Online]. Available: https://arxiv.org/abs/1512.03385

[49] G. Jocher, "Yolov5," https://github.com/ultralytics/yolov5, 2020.

[50] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," 2015. [Online]. Available: https://arxiv.org/abs/1506.01497

[51] F. Altenberger and C. Lenz, "A non-technical survey on deep convolutional neural network architectures," 03 2018.

[52] A. Mounsif, "Football and computer vision can computer vision improve football?" https://web.unibas.it/bloisi/corsi/progettivep/soccer-player-detection.html.

[53] A. Bochkovskiy, C. Wang, and H. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," *CoRR*, vol. abs/2004.10934, 2020. [Online]. Available: https://arxiv.org/abs/2004.10934

[54] Z. Ge, S. Liu, F. Wang, Z. Li, and J. Sun, "Yolox," https://github.com/Megvii-BaseDetection/YOLOX, 2021.

[55] A. Bochkovskiy, C. Wang, and H. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," *CoRR*, vol. abs/2004.10934, 2020. [Online]. Available: https://arxiv.org/abs/2004.10934

[56] H. S. Lee, "Bag of tricks for image classification with convolutional neural networks review," https://hoya012.github.io/blog/Bag-of-Tricks-for-Image-Classification-with-Convolutional-Neural-Networks-Review/, 2019.

[57] A. Anka, “Yolo v4: Optimal speed & accuracy for object detection,” https://towardsdatascience.com/yolo-v4-optimal-speed-accuracy-for-object-detection-79896ed47b50, 2020.

[58] J. Ferryman and A. Ellis, “PETS2010: Dataset and challenge,” in *2010 7th IEEE International Conference on Advanced Video and Signal Based Surveillance.* IEEE, Aug. 2010. [Online]. Available: https://doi.org/10.1109/avss.2010.90

[59] A. Kendall, Y. Gal, and R. Cipolla, “Multi-task learning using uncertainty to weigh losses for scene geometry and semantics,” 2017. [Online]. Available: https://arxiv.org/abs/1705.07115

[60] F. Devernay and O. Faugeras, “Straight lines have to be straight,” *Machine Vision and Applications*, vol. 13, no. 1, pp. 14–24, Aug. 2001. [Online]. Available: https://doi.org/10.1007/pl00013269

[61] G. Bradski, “The OpenCV Library,” *Dr. Dobb’s Journal of Software Tools*, 2000.

[62] A. R. Channel. Roma v lazio, 2018-19 — full match. Youtube. [Online]. Available: https://www.youtube.com/watch?v=QdDrugBVuEA&ab_channel=ASRoma

[63] correcting fisheye distortion programmatically. Stack Overflow. [Online]. Available: https://stackoverflow.com/questions/2477774/correcting-fisheye-distortion-programmatically

[64] H. GuangXin and A. Nicolai, “Bytetrack_reid,” https://github.com/andrybicio/ByteTrack_ReID, 2022.

[65] G. Van Rossum and F. L. Drake Jr, *Python tutorial.* Centrum voor Wiskunde en Informatica Amsterdam, The Netherlands, 1995.

[66] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. Hamrick, J. Grout, S. Corlay, P. Ivanov, D. Avila, S. Abdalla, and C. Willing, “Jupyter notebooks – a publishing format for reproducible computational workflows,” in *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, F. Loizides and B. Schmidt, Eds. IOS Press, 2016, pp. 87 – 90.

[67] T. pandas development team, “pandas-dev/pandas: Pandas,” Feb. 2020. [Online]. Available: https://doi.org/10.5281/zenodo.3509134

[68] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, “Array programming

with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2

[69] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[70] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[71] P. T. Inc. (2015) Collaborative data science. Montreal, QC. [Online]. Available: https://plot.ly