CHALMERS
UNIVERSITY OF TECHNOLOGY

Tracking players and ball in football videos

Master's thesis in Data Science and AI

Hugo Ganelius
Jhanzaib Humayun

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2024

Master's thesis 2024

Tracking players and ball in football videos

Hugo Ganelius
Jhanzaib Humayun

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden 2024

Tracking players and ball in football videos
Hugo Ganelius Jhanzaib Humayun

Supervisor: Lennart Svensson, Department of Electrical Engineering
Advisor: Anders Sjöberg, Fraunhofer Chalmers Centre
Examiner: Lennart Svensson, Department of Electrical Engineering

Master's Thesis 2024
Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Players and ball detected using YOLO.

Typeset in LaTeX
Gothenburg, Sweden 2024

Tracking players and ball in football videos
Hugo Ganelius, Jhanzaib Humayun
Department of Electrical Engineering
Chalmers University of Technology

Abstract

This master's thesis explores challenges and methodologies of accurately tracking players and the ball within football video sequences, employing advanced object detection and tracking techniques. The research is integrated into a larger project aimed at reconstructing football sequences in a virtual reality (VR) environment, thus enhancing the training and strategic analysis capabilities in sports environments.

The thesis primarily utilizes models based on convolutional neural networks, such as the YOLOv9 model for object detection to identify the positions of the players and the ball within different frames. Advanced methods such as the online tracking algorithm BoT-SORT and the minimum cost flow algorithm are employed for detection-based tracking, optimizing the accuracy of continuous player and ball trajectories in the complex, fast-moving setting of a football game. We also employ the TrackNet V3 model for predicting ball trajectories, as an alternative to detection-based tracking. TrackNet utilizes multiple sequential frames to predict the trajectory of the ball, enhancing detection accuracy even during fast play or when the ball is momentarily occluded.

Results from these experiments indicate promising player tracking, though challenges persist when it comes to, e.g., id-switching. Ball tracking is shown to be more difficult due to its small size and high movement speed, which often leads to reduced detection reliability. TrackNet successfully predicts the ball's trajectory when it is moving quickly and not occluded, but instead struggles when the ball is still or frequently occluded (such as during dribbling). The outcomes contribute towards the understanding of object tracking in sports and the development of applications for enhanced game analysis.

Keywords: computer vision, multiple object tracking, convolutional neural networks, deep neural networks, engineering, project, thesis.

Acknowledgements

We would like to extend our thanks to our academic supervisor and examiner Lennart Svensson, who first had the idea for this project and graciously chose us to be the ones to see it through.

We would also like to thank Fraunhofer Chalmers Centre for providing us with a space to work and resources. In particular we would like to thank Anders Sjöberg at FCC, for acting as our advisor and providing us with invaluable ideas and input.

Thanks also go out to the football club IFK Göteborg for providing video data, as well as meeting us at several points in time to discuss what a professional club would want out of a project like this one.

Lastly, we wish to thank the other participants of the Football in VR project, Filip Anjou, Albin Ekström, Joakim Osterman, and Olof Sjögren. Despite being deeply engaged in their own theses, they have generously shared ideas that have enriched our work.

Hugo Ganelius, Jhanzaib Humayun, Gothenburg, 2024-06-13

Contents

List of Figures

List of Tables

1

Introduction

Football, also known as soccer in some regions of the world, is not just a sport but a global phenomenon that captures the hearts of millions. Its universal appeal and the dramatic moments it produces make it a significant cultural and social fixture. With its popularity there naturally arises a demand for high-quality analytical tools that can be used for analyzing play. The excitement and unpredictability of football make it an excellent candidate for advanced analysis.

This masters thesis is one component of a broader initiative consisting of three distinct thesis projects, each targeting different aspects of the same overarching goal: the reconstruction of football sequences in a virtual reality environment. The initiative aims to leverage the latest in VR and deep learning technology to create a dynamic and interactive model of football games.

1.1 Purpose of the Thesis

The primary focus of this thesis is to explore the challenges of performing accurate MOT (Multiple Object Tracking) of football games. The goal is the development of robust detection and tracking systems for both the football and the players involved in the game, for the purpose of game sequence recreation and play analysis. We explore different solutions to find out their strengths and shortcomings, in an effort to find the best approach to solving the task.

1.2 The football in VR project

The project of which this thesis is a component seeks to offer a novel way to experience football, providing perspectives and insights that are not possible with traditional camera perspectives. The completed project would allow the ability to stand and move around on a virtual field as an accurate replay of a certain game sequence plays out. This would allow coaches and players to achieve a more precise and complete understanding of the situation, such as distances between players and the ball. Furthermore, one could render the sequence from the point of view of a player, making it possible to understand exactly what that player was able to see based on their field of view.

The project's three main parts are as follows:

1. Detection and Tracking (subject of this thesis): Developing the methods and technologies necessary to detect and continuously track the position and movement of the players and the ball.

2. Pose Estimation (subject of [1]): Once tracking is reliably handled, the next step is estimating the poses of the players. This involves understanding the positions and movements of individual player limbs and their orientation during the game.

3. Virtual Reality Rendering (subject of [2]): The final part of the pipeline will take the tracking data and pose information to render these elements in a virtual reality environment, creating an immersive experience that can be used for analytical purposes.

1.3 Specific Challenges in Tracking for Football

Tracking football presents unique challenges due to the dynamic nature of the game and the environment in which it is played. Here are some aspects of football footage that make tracking difficult:

- **Fast-Paced Action:** Football is a fast-paced sport where players often cluster together, further complicating tracking efforts. Rapid changes in direction, speed, and group formations can confuse algorithms designed to follow individual movements.

- **Occlusions:** Players frequently block each other or the ball. Both are occasionally obscured by other elements on the field, such as the referee or goal posts. These occlusions can cause tracking algorithms to temporarily lose track of a player or the ball.

- **Distance from Camera:** The wide shots used to capture football matches mean that objects can appear very small in the frame. This distance reduces the resolution and detail available to detection and tracking models, making it more difficult to identify specific features such as numbers on jerseys.

Further complicating matters, players of the same team wear identical jerseys, making it difficult to distinguish one player from another, especially from distant camera angles. In professional football, it is also common for players on the same team to be of the same gender, ethnicity, and physical build. This uniformity can lead to frequent ID switches in tracking systems.

When it comes to the ball, we do not need to concern ourselves with object ID. In this thesis we only concern ourselves with tracking a single ball. Even if the methods were to be expanded such that they track multiple balls simultaneously, ID-switching is not a big issue for sports analytics as it seldom matters which ball is which.

Instead, the difficulty comes from actually detecting the ball in the first place. The small resolution and lack of distinguishable features mean there is little for a detection model to latch on to, unlike players where it can identify features such as

(a) Small resolution

(b) Compression artifacts

(c) High background noise, the ball is above the S at the bottom.

Figure 1.1: Examples demonstrating why the ball is difficult to detect. Note in all of these the lack of distinct recognisable features.

arms and legs. Compression artifacts and motion blur further complicates the issue, as the ball does not have a consistent recognisable shape. Beyond this, there is often a lot of noise in the image making it difficult even for humans to discern what is the ball and what is something else. Figure 1.1 contains some example images that demonstrate why a computer model might struggle to identify the ball in still frames.

1.4 Scope

As this thesis is a part of a project with a specific goal, some demarcations are made in the types of solutions that are explored to better fit this goal.

For the purpose of this thesis, we will only apply methods on footage using a fixed camera perspective. The end goal of this thesis is a program that can create three dimensional recreations of football games. Dynamic camera calibration for translating between pixel coordinates to real world coordinates is not in scope of the project at large, and thus dynamic perspective such as tilt, pan, and zoom will not be handled in this thesis.

The scope also does not include the development of a real-time tracking system. Instead, the focus is on post-processed analysis, aiming to maximize accuracy and reliability in identifying and tracking players and the ball across pre-recorded footage.

2

Background

In the upcoming sections, we will delve into prior research and developments in the field of MOT. We will explore various tracking methodologies, explaining and distinguishing between them to provide a comprehensive understanding of the subject. Furthermore, we will detail the specific models employed in this thesis, shedding light on their unique characteristics and the fundamental principles they leverage. This will include an examination of the underlying concepts and techniques that these models utilize to achieve effective tracking performance.

2.1 Object Detection

Object detection is the task of classifying and estimating the location of objects within an image. This involves not only recognizing the types of objects presented in an image but also pinpointing their specific positions and sizes, typically by drawing bounding boxes around each object. Object detection serves as a fundamental component in numerous applications including autonomous driving, video surveillance, and interactive robotic systems. Tracking systems based on the results of a detection model are known as detection-based trackers.

This section will detail some fundamental concepts in object detection and describe one of the state of the art series of detection models, the YOLO series of general-purpose object detectors.

2.1.1 Convolutional Neural Networks

CNNs (Convolutional Neural Networks) are a class of deep neural networks highly effective in areas such as image recognition and classification. CNNs have become fundamental to the development of object detection models due to their ability to automatically detect and learn optimal features from visual data, without the need for manual feature extraction.

The architecture of a CNN typically involves several layers that transform the input image to produce a desired output such as a class label or a set of bounding box coordinates. The key layers used in a CNN include:

- **Convolutional Layers:** These layers apply a number of convolutional filters to the input to create feature maps that summarize the presence of detected features in the input. Each filter detects different aspects of the image, such as edges, textures, or more complex patterns in deeper layers.

- **Activation Functions:** Non-linear activation functions, such as the ReLU function [3], are applied after each convolution operation to introduce non-linear properties into the network, allowing it to learn more complex patterns.
- **Pooling Layers:** Pooling (usually max pooling) reduces the dimensionality of each feature map but retains the most important information. Pooling layers help to make the detection of features invariant to changes in scale and translation.
- **Fully Connected Layers:** Towards the end of the network, fully connected layers use the features extracted by convolutional and pooling layers to perform various tasks. These could include classification, regression, or other types of predictions based on the training dataset
- **Normalization Layers:** Batch normalization [4] is often applied to the inputs of each layer to stabilize learning. This technique often reduces the number of training epochs required to effectively train deep networks.

2.1.2 Methods in Object Detection

Object detection methods can be broadly categorized into two types: one-stage detectors and two-stage detectors. Two-stage detectors, such as R-CNN [5] and its variants (Fast R-CNN [6], Faster R-CNN [7]), first generate potential regions of interest and then classifies and locates objects within those regions. In contrast, one-stage detectors like YOLO (You Only Look Once) [8] and SSD (Single Shot MultiDetector) [9] bypass the region proposal stage and predict object classes and locations directly from the image features in a single pass, achieving faster processing times at the cost of some accuracy.

2.1.3 YOLO Series

Among the one-stage methods, the YOLO series has been particularly influential due to its speed and efficiency [8]. YOLO frames object detection as a single regression problem, straight from image pixels to bounding box coordinates and class probabilities. Over several iterations, YOLO has been refined and improved. The latest in this series, YOLOv9, continues this evolution.

The backbone of YOLO is typically a CNN that is responsible for extracting features from the image. Initially, YOLO used a custom lightweight CNN, which was composed of convolutional layers with batch normalization and Leaky ReLU activations. As the versions progressed, the architecture incorporated more advanced backbones for improved feature extraction.

From YOLOv3 [10] onward, the architecture started using feature pyramids to improve detection at multiple scales. This approach uses pathways to bring high-resolution features from earlier layers together with lower-resolution features from deeper layers, enhancing the network's ability to detect objects at different sizes.

At the time of writing, YOLOv9 [11] is the latest in the YOLO series. It introduces several novel concepts in the domain of deep learning and object detection, emphasizing the use of PGI (Programmable Gradient Information) and a new network architecture called GELAN (Generalized Efficient Layer Aggregation Network). The innovations presented in YOLOv9 aim to address the critical issues of information loss during deep network computations and to enhance the learning efficacy of object detection systems.

The concept of PGI is central to YOLOv9. It provides a mechanism to control gradient propagation through the network, ensuring that gradients are more representative of the underlying data features needed for accurate predictions. In traditional models, as data passes through multiple layers, some information is invariably lost. This loss can degrade the quality of the gradients, leading to poor training outcomes. PGI addresses this by ensuring that the gradient flow maintains high fidelity to the original data features necessary for accurate predictions.

PGI is designed to work across various model sizes, from lightweight to larger models. It does this by adjusting the gradient flow based on the model's architecture, making it a versatile approach in different deployment scenarios.

GELAN is a new lightweight network architecture that leverages gradient path planning to optimize the network's learning capacity and efficiency. This architecture is designed to minimize parameter count while maximizing inference speed and accuracy, making it suitable for real-time applications.

YOLOv9 represents a significant advancement in the field of object detection, addressing several limitations of previous YOLO versions and other contemporary detection systems. Its innovative approach to managing gradient information and network architecture allows for highly efficient and accurate object detection, suitable for various applications including autonomous driving, surveillance, and real-time video analysis.

2.2 Online vs offline tracking

Algorithms for MOT can be categorized into two main types; online and offline algorithms. These two types differ in the type of data they use and the situations they can be applied [12].

Online tracking refers to trackers that only rely on current and previous information when processing a frame in a video. In other words, online trackers do not make use of information from future frames and thus can run in real time. This real-time capability makes online tracking algorithms highly valuable in applications where immediate feedback is crucial, such as in live surveillance, autonomous driving, and interactive augmented reality systems. Since they run in real time, these trackers are often designed to be lightweight and efficient, balancing the need for speed with the requirement for precision.

Offline tracking algorithms, in stark contrast to online algorithms, utilize information from future frames when estimating object trajectories. This approach allows for more accurate and coherent tracking because it takes into account the entire sequence of frames rather than making decisions based solely on the current and past frames. This foresight helps in resolving ambiguities that might arise in real-time tracking scenarios and can lead to significantly improved tracking performance, especially in complex and dynamic environments. The trade off for this is that these algorithms are often more computationally expensive, and less viable to run live.

2.3 Kalman Filter

The Kalman Filter [13] is a powerful predictive and corrective mathematical algorithm used for estimating the state of linear dynamic systems from a series of incomplete and noisy measurements. It is extensively used in the field of automated control systems, robotics, aerospace, and, what is relevant for our purpose, in MOT algorithms.

The Kalman Filter operates in a two-step process, the *predict* step and the *update* step:

- **Predict**: The filter predicts the current state and covariance estimates to advance the state ahead temporally. This prediction is based on the system's previous state and the known control inputs affecting the system.
- **Update**: When new measurements are available, the filter updates its estimates to correct the state vector and covariance estimates. This correction is based on the discrepancy (residual) between the predicted estimates and the actual measurements obtained at that step.

2.3.1 Kalman Filter Equations

This section outlines the key equations of the Kalman Filter along with a detailed description of the matrices and variables involved:

1. **State Prediction**:
$$\hat{x}_{k|k-1} = F_k \hat{x}_{k-1|k-1} + B_k u_k,$$
   where:
   - $\hat{x}_{k|k-1}$ is the predicted state estimate,
   - F_k is the state transition model applied to the previous state $\hat{x}_{k-1|k-1}$,
   - B_k is the control-input model that scales the control vector u_k,
   - u_k is the control vector, representing external inputs to the system.
2. **Covariance Prediction**:
$$P_{k|k-1} = F_k P_{k-1|k-1} F_k^T + Q_k,$$
   where:

- $P_{k|k-1}$ is the predicted covariance estimate of the state,
- $P_{k-1|k-1}$ is the covariance of the previous state estimate,
- F_k^T is the transpose of the state transition model,
- Q_k is the process noise covariance matrix.

3. **Kalman Gain Calculation**:

$$K_k = P_{k|k-1}H_k^T(H_kP_{k|k-1}H_k^T + R_k)^{-1},$$

where:

- K_k is the Kalman Gain, a factor that reflects the importance of the incoming measurement compared to the current estimate,
- H_k is the measurement model that relates the state estimate to the measured data,
- R_k is the measurement noise covariance, representing the expected variability in the measurements.

4. **State Update**:

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k(z_k - H_k\hat{x}_{k|k-1}),$$

where:

- $\hat{x}_{k|k}$ is the updated estimate of the state after incorporating the measurement,
- z_k is the actual measurement observed at time k.

5. **Covariance Update**:

$$P_{k|k} = (I - K_kH_k)P_{k|k-1},$$

where:

- $P_{k|k}$ is the updated state covariance after the measurement has been incorporated,
- I is the identity matrix, used in updating the covariance matrix.

2.3.2 Application in Tracking

In many tracking applications, the Kalman Filter estimates the state of moving objects considering their position and velocity. In online tracking algorithms such as BoT-SORT, a kalman filter is used to predict the next bounding box of a previously tracked object [14]. This track is then associated with a detected bounding box through metrics such as IoU (Intersection over Union, see section 2.6.1). More detail about BoT-SORT is found in section 2.7.4.

2.4 Appearance feature embedding

MOT involves detecting multiple objects within a video sequence and maintaining their identities over time. One of the critical components in MOT is the effective representation and matching of an object's visual features, which improves accurate data association between different frames. This section discusses the concepts of appearance feature embedding and matching, which are essential for robust MOT systems.

Feature embedding refers to the process of transforming raw data, such as images or object detections, into a lower-dimensional space where the characteristics of the data are preserved [15]. This transformation is typically achieved using deep neural networks, which learn to encode relevant information into a compact vector representation. The embedded features should ideally capture the unique attributes of objects, such as shape, color, and texture, while being invariant to changes in scale, illumination, and viewpoint.

In the context of MOT, appearance embeddings are often extracted from detected bounding boxes using convolutional neural networks. This is then used as a metric for associating the same object in different frames based on appearance. The use of appearance embeddings for MOT can be attributed to Deep-SORT [16]. More information about Deep-SORT and other tracking algorithms can be found in section 2.7.

2.5 Foreground Extraction

Foreground extraction is a key process in computer vision that isolates dynamic elements, such as players and the ball, from static backgrounds in video sequences. A prevalent approach to achieving accurate foreground extraction is through the use of a median frame as the background model.

The median frame is derived by computing the median value of each pixel across a set of frames. This technique is effective because it minimizes the impact of transient elements that might appear in only a few frames, thus creating a stable representation of the static background.

The subsequent extraction of the foreground involves comparing the current frame of the video to this median background, highlighting areas with significant pixel intensity changes. These differences typically indicate motion, and thus, the presence of foreground elements.

This foreground extraction method is effective primarily under specific conditions. It assumes a stationary camera and a relatively static background, as the median frame relies on consistent background content. However, this method has limitations and may not perform well in dynamic environments with moving shadows, fluctuating lighting, or changing background elements. Camera movement or shake can also disrupt the alignment between the median background frame and current frames, resulting in inaccurate foreground masks and detections.

Prerequisites:

- Stationary camera
- Relatively static background
- Minimal background changes (e.g., lighting, shadows)

2.6 Important metrics in multiple object tracking

This section outlines a comprehensive set of metrics used in data analysis, computer vision, and specifically in machine learning model evaluation for object detection. These metrics assess various aspects of model performance, from general accuracy to specific losses associated with object detection tasks.

- **Accuracy:** Measures the overall correctness of the model, calculated as the ratio of correctly predicted observations to the total observations
$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}}.$$
- **Precision:** Reflects the ratio of correctly predicted positive observations to the total predicted positives
$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}.$$
- **Recall (Sensitivity):** Measures the ratio of correctly predicted positive observations to all actual positives
$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}.$$
- **F1 Score:** The harmonic mean of precision and recall
$$F1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}.$$
- **Specificity:** Measures the ratio of correctly predicted negative observations to all actual negatives
$$\text{Specificity} = \frac{\text{True Negatives}}{\text{True Negatives} + \text{False Positives}}.$$
- **MAE (Mean Absolute Error) and MSE (Mean Squared Error):** Measures the accuracy of predictions, with MAE averaging absolute differences, and MSE squaring differences before averaging.
$$MAE = \frac{1}{n}\sum_{i=1}^{n} |y_i - \hat{y}_i|, \tag{2.1}$$
$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2, \tag{2.2}$$

where n is the number of observations, y_i represents the actual values of the observations and $\hat{y}_i$ represents the predicted values.

- **Box Loss:** Reflects the model's accuracy in predicting bounding boxes around detected objects. They measure the discrepancy between the predicted and actual bounding box coordinates. Lower values indicate higher precision in locating objects.
- **Classification Loss:** Gauges the models capability to correctly classify objects within the detected bounding boxes. These metrics evaluate the error in classification predictions, with lower scores denoting better classification performance.
- **DFL (Distribution Focal Loss):** Is a loss function used in object detection that focuses on improving the accuracy of bounding box predictions by modeling the distribution of box offsets instead of direct value prediction. This method enhances the model's precision in localizing objects.
- **WBCE (Weighted Binary Cross Entropy) Loss:** Is a loss function used to handle imbalanced classes in binary classification tasks. It assigns different weights to the positive and negative classes, which helps in emphasizing the minority class and improving the overall model performance. The WBCE can be calculated as:

$$WBCE = -\frac{1}{N}\sum_{i=1}^{N}\left[w_+ y_i \log(\hat{y}_i) + w_-(1-y_i)\log(1-\hat{y}_i)\right],$$

  where N is the total number of observations, y_i is the actual class label, $\hat{y}_i$ is the predicted probability, w_+ is the weight for the positive class, and w_- is the weight for the negative class.
- **Mean Average Precision (mAP):** Is a metric used to evaluate the accuracy of object detection models in computer vision. It calculates the average precision across all classes, considering the precision and recall of the models predictions, providing a comprehensive measure of the model's overall effectiveness in detecting various objects.
  - mAP50 calculates the mean average precision at 50% Intersection over Union (IoU), providing a benchmark for model accuracy in object localization at a moderate threshold.
  - mAP50-95 extends this evaluation across a range of IoU thresholds from 50% to 95%, averaging the precision values at these levels to offer a comprehensive view of the models performance across varying degrees of localization strictness.
- **MOTA (Multiple Object Tracking Accuracy):** Measures the accuracy of a tracking system by accounting for false positives, false negatives, and identity switches. It is defined as:

$$\text{MOTA} = 1 - \frac{\sum_t(\text{FP}_t + \text{FN}_t + \text{IDSW}_t)}{\sum_t \text{GT}_t},$$

where FP_t is the number of false positives, FN_t is the number of false negatives, IDSW_t is the number of identity switches, and GT_t is the number of ground truth objects at time t. [17]

- **MOTP (Multiple Object Tracking Precision):** Measures the precision of the tracking system by evaluating the average dissimilarity between the predicted and the ground truth object positions. It is defined as:

$$\mathrm{MOTP} = \frac{\sum_i^N \sum_t^T d_{it}}{\sum_t c_t},$$

where d_{it} is the distance between the detected and ground truth position of object i at time t, N is the number of objects, T is the total number of frames, and c_t is the number of matches found in frame t [17].

These metrics collectively form a robust framework for evaluating and monitoring model performance, crucial for the effective deployment and optimization of computer vision technologies in various applications.

Another metric worth mentioning, while not used by us in this thesis, is the one proposed by [18]. It is a metric of similarity between sets of trajectories, designed for the evaluation of MOT algorithms. The proposed metric is designed to be a true metric space function — it is non-negative, satisfies the identity of indiscernibles, is symmetric, and obeys the triangle inequality. The metric is formulated through solving a multi-dimensional assignment problem, which is non-polynomial and not feasible when the sets of trajectories are large (contain a large number of trajectories). The paper details a lower bound for the metric that can be computed in polynomial time using linear programming.

2.6.1 Intersection over Union

IoU (Intersection over Union) is a crucial metric in the field of computer vision, particularly for tasks such as object detection and segmentation. IoU measures the overlap between two bounding boxes, Such as one predicted by a tracking algorithm and one detected by a detection model. Mathematically, IoU is defined as the area of overlap between the two bounding boxes divided by the area of their union:

$$\mathrm{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}.$$

The IoU value ranges from 0 to 1, where 0 indicates no overlap and 1 signifies perfect alignment. A higher IoU score generally indicates a more accurate prediction.

In the context of MOT, IoU plays a pivotal role as a metric for data association as well as in assessing the performance of tracking algorithms. MOT involves detecting objects in video frames and maintaining their identities across frames, which requires robust and accurate tracking mechanisms. IoU is used in several key aspects of MOT:

- **Tracking through Association:** In each frame of a video, objects are detected, and IoU is used to associate these detections and decide if they belong to the same track. For online tracking the IoU is usually calculated between a prediction based on an existing track, and a new detection that may or may not belong to this track. A high overlap means the detection is more likely to belong to the tracked object. For offline tracking, IoU is usually computed between detections in different frames.

- **Performance Evaluation:** IoU serves as a critical metric for evaluating the performance of MOT systems. Metrics such as MOTA and MOTP [17] often incorporate IoU to measure how well the tracking algorithm maintains accurate and consistent trajectories of objects over time. High IoU scores indicate better tracking performance, as they reflect more accurate localization of objects.

Overall, IoU is a fundamental metric in multiple object tracking, enabling precise object localization, effective identity association, and comprehensive performance evaluation. Its application ensures that tracking algorithms can reliably follow multiple objects across complex and dynamic video scenes, thereby enhancing the robustness and accuracy of MOT systems.

2.7 Some Online Trackers

Online trackers process each frame sequentially and make immediate decisions about object identities without access to future frames. This section introduces and discusses several relevant prominent online tracking algorithms that have made significant contributions the field.

2.7.1 SORT

SORT (Simple Online and Realtime Tracking) [19] is a minimalist approach to online tracking which primarily uses the Hungarian algorithm [20] for frame-to-frame association. This method employs a simple linear constant velocity model that predicts the motion of each tracked object between frames and matches these predictions with incoming detections. Detections are assigned to existing tracks based on the IoU metric, which measures the overlap between predicted bounding boxes and new detections. SORT is designed to be lightweight and efficient, but it struggles with long-term occlusions and identity switches, primarily because it does not incorporate appearance information to aid in re-identification of objects.

2.7.2 Deep-SORT

Deep-SORT [16] enhances the SORT algorithm by integrating appearance information, significantly improving tracking performance, especially in scenarios where objects are occluded or interact closely. It extends the standard SORT by employing a deep learning-based feature extractor, which generates a unique appearance descriptor for each detected object. These descriptors are then used to compute a Mahalanobis distance metric for matching detections to existing tracks, in combination with the IoU strategy used in SORT.

The deep-SORT algorithm also incorporates a more sophisticated motion model using a Kalman filter, which helps in predicting object positions more accurately from one frame to the next. The addition of an age parameter and a hit counter for each track allows deep-SORT to handle temporary occlusions by retaining identity even when objects disappear for short periods. These enhancements make deep-SORT more robust against the challenges of tracking objects in crowded scenes where frequent interactions and occlusions occur.

2.7.3 ByteTrack

ByteTrack [21] stands out as a significant advancement in online tracking algorithms. It improves on previous tracking algorithms primarily by utilizing low-confidence detections in its tracking, unlike previous trackers that simply discarded them. It at first tries to find a suitable detection for each tracklet among detections that reach a certain confidence threshold. If no match is found, it will then try to find a suitable detection for that tracklet among the detections that did not reach the threshold.

2.7.4 BoT-SORT

BoT-SORT [14] further builds upon the foundation of ByteTrack, enhancing its capabilities through several significant innovations. These improvements are designed to tackle some of the inherent limitations in the previous "SORT-like" algorithms and to integrate the advantages of both motion and appearance information effectively. Here are the primary ways in which BoT-SORT advances over ByteTrack:

- **Camera Motion Compensation (CMC)**: Integrates camera motion estimation and correction to align predicted bounding boxes with actual detections accurately. This reduces identity switches and false negatives caused by camera movement.

- **Enhanced Kalman Filter State Vector**: Modifies the Kalman filter's state vector to estimate the width and height of bounding boxes directly, rather than the aspect ratio. This leads to more accurate bounding box predictions, especially in scenes with varying perspectives.

- **Robust Association with IoU and Re-ID Fusion**: Employs a fusion of IoU and Re-Identification (Re-ID) features to associate detections to tracks robustly. This method uses deep appearance features and a dual-thresholding strategy to ensure only reliable associations are maintained.

- **Superior Performance**: Demonstrates significant improvements over existing methods in key MOT metrics (MOTA [17], IDFL [22], and HOTA [23]) on the MOT17 [24] and MOT20 [25] datasets, indicating better tracking accuracy and identity maintenance.

These enhancements collectively contribute to BoT-SORT's high performance, making it suitable for real-time tracking in dynamic and crowded environments.

2.8 Minimum cost flow

One effective method for offline tracking is to formulate the problem as an optimal path graph problem. One such method of formulating object tracking as a graph problem is known as MOT based on MCF (Minimum Cost Flow) [26].

The core of this method lies in establishing a comprehensive graph where each node represents a detection and each edge signifies a possible transition between detections across consecutive frames. The task of the algorithm is to find the optimal paths through this graph, where each path corresponds to the trajectory of an object over time. This allows the algorithm to leverage global information and make more informed decisions about the continuity and identity of each object.

The cost assigned to each edge reflects the likelihood or confidence that the transition between these detections is valid, encompassing criteria such as spatial proximity, motion consistency, and appearance similarity. To construct trajectories, the algorithm traces the path of the flow from its start to the end across the network. By following the minimum-cost flow, we can link detections from frame to frame, thereby mapping out the trajectory of each object throughout the sequence.

2.8.1 Graph Construction and Edge Definition

In MOT using the MCF algorithm, a directed graph is typically constructed where each detection from the video frames is represented as a node. Special nodes, namely the *source* and *sink*, are introduced to manage the initiation and termination of object tracks:

- **Source Node:** Acts as the entry point for potential tracks.
- **Sink Node:** Serves as the exit point where tracks are terminated.

Figure 2.1 shows a graph structure with five detections across three frames. It illustrates connections that allow track initiation or termination at any time step as well as "skip" connections that allow tracks to travel across multiple frames at once, in case of missing detections.

2.8.2 Edge Cost and Capacity

Edges within the graph are specifically defined by their costs and capacities to optimize tracking:

- **Edge Cost:** The cost of travelling between two nodes is calculated as a weighted sum of costs for certain characteristics. These are usually characteristics such as spatial proximity, appearance similarity, number of frames being skipped, and other relevant metrics, influencing the path selection by making certain transitions more favorable. There are also separate costs for creating and terminating tracks.
- **Edge Capacity:** When minimum cost flow is used for MOT, capacities are typically set to 1 between detection nodes to ensure that each detection is

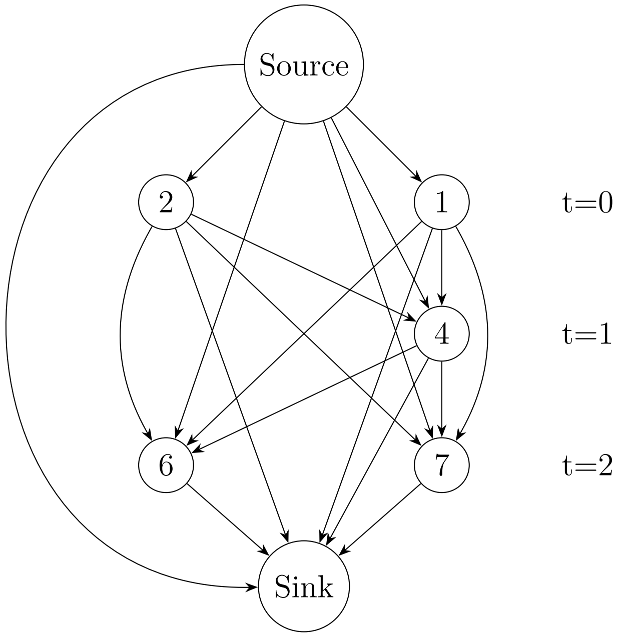


Figure 2.1: Graph simplification where each node represents a detection. Nodes that are aligned horizontally are detections from the same frame. Edges represent possible transitions between detections, i.e. possible movement of objects. This graph demonstrates the flexibility in track management, including frame skipping.

used exactly once per track. The capacity on the edge connecting the source to the sink reflects the expected maximum number of concurrent tracks.

2.8.3 Pathfinding via Flow Optimization

MCF is an optimization problem, this section lists the objective function and the constraints for the problem. For this section we use the following definitions.

- V denotes the set of all nodes in the graph, where in our case each node represents an object detection.
- s and t denote the source and sink nodes, respectively.
- E denotes the set of all edges in the graph, where each edge connects a pair of nodes, indicating potential transitions from one detection to another or from/to special nodes like the source and sink.
- f_{ij} represents the flow on the edge from node i to node j, indicating how many tracks (or parts of tracks) pass through this edge.
- c_{ij} represents the cost associated with the edge from node i to node j.

Objective Function to minimize

We wish to minimize the sum of the costs of all flows through the graph, expressed by

$$\sum_{(i,j)\in E} f_{ij} c_{ij}. \tag{2.3}$$

Flow Conservation

The total flow entering any node i (except for the source s and sink t) must equal the total flow exiting node i. This is crucial for maintaining consistent object tracks without discontinuities. The equation for this constraint is

$$\sum_{j\in V} f_{ij} - \sum_{j\in V} f_{ji} = 0 \quad \text{for all } i \in V \setminus \{s,t\}. \tag{2.4}$$

Non-Negative Flow

The flow on any edge cannot be negative, reflecting the physical interpretation of flow representing the number of tracks. This is expressed as

$$f_{ij} \geq 0 \quad \text{for all } (i,j) \in E. \tag{2.5}$$

Flow Equality at Source and Sink

The total flow entering the network from the source must equal the total flow exiting the network to the sink. This guarantees that all initiated tracks are properly terminated. The equation for this constraint is

$$\sum_{i\in V\setminus\{s\}} f_{si} = \sum_{i\in V\setminus\{t\}} f_{it} = \text{Total number of tracks.} \tag{2.6}$$

In summary, this optimization formulation encapsulates the logic needed to connect detections across frames in a way that respects the flow of objects through space and time, aiming to construct the most probable tracks at the minimal possible cost. This setup is fundamental for algorithms designed to handle complex object tracking scenarios, such as in surveillance or sports analytics, where accuracy and computational efficiency are crucial.

2.9 TrackNet

TrackNet [27] is a tracking model designed for tracking the ball in sports video, specifically tennis and badminton. Rather than relying on object detection to predict bounding boxes which are then connected in tracks, the TrackNet architecture is designed to directly predict the trajectory of the ball from the video data. The

TrackNet model uses a neural network to which the input is a sequence of subsequent RGB frames, and the output is a sequence of heatmaps of the same resolution as the input, representing probable ball locations.

The model follows a U-net architecture [28], with the input first passing through a fully convolutional network extracting features from the sequence, followed by a deconvolutional network constructing the heatmaps. The U-net model is characterized by its encoder-decoder structure, with skip connections that help preserve spatial hierarchies between input and output. This design enables TrackNet to capture both high-level features and fine details from the video frames, which is essential for accurate localization of fast-moving objects across frames.

The integration of convolutional and deconvolutional layers allows the model to process and reconstruct image data effectively. The convolutional layers act as feature extractors, analyzing the frames to detect patterns and features that signify the presence and movement of the ball. Conversely, the deconvolutional layers work to project these lower-dimensional feature representations back into the original image space, creating a detailed and accurate heatmap.

The fact that the model considers a sequence of images allows it to intrinsically identify movement. This ability to identify movement is crucial in dynamic sports like tennis and badminton, as well as football, where the ball is often difficult to make out in single frames due to its small size, lack of distinguishable features, and motion blur.

TrackNet V3 [29], designed solely for badminton, builds upon TrackNet by concatenating the match background to the input of the tracknet model, allowing it to more easily identify non-static objects. The background is calculated as the mean RGB values for each pixel across the match/video sequence. This introduces the requirement that the camera must be fixed, as changing the perspective makes it impossible to calculate and use a background frame. The architecture of the TrackNet V3 model is visualised in figure 2.2.

They also improve the performance by adding a separate rectification model that learns to correct the trajectory of the tracknet model in cases where the original trajectory estimation fails, such as when the shuttlecock is occluded. This separate model is named InpaintNet, and is inspired by inpainting techniques used by generative image models. The architecture of the InpaintNet model follows a U-Net architecture, similarly to the main tracking model. The input of the model is the trajectory produced by the main model, while the output is the rectified trajectory.

In the training phase, the primary tracking module, TrackNet, employs weighted binary cross entropy loss to optimize performance. Meanwhile, the rectification module, InPaintNet, utilizes mean squared error loss to refine its accuracy.

2.10 Datasets

SoccerNet [30] has multiple datasets publicly available for different prediction tasks related to football. Their data for tracking is made from broadcast footage (non-

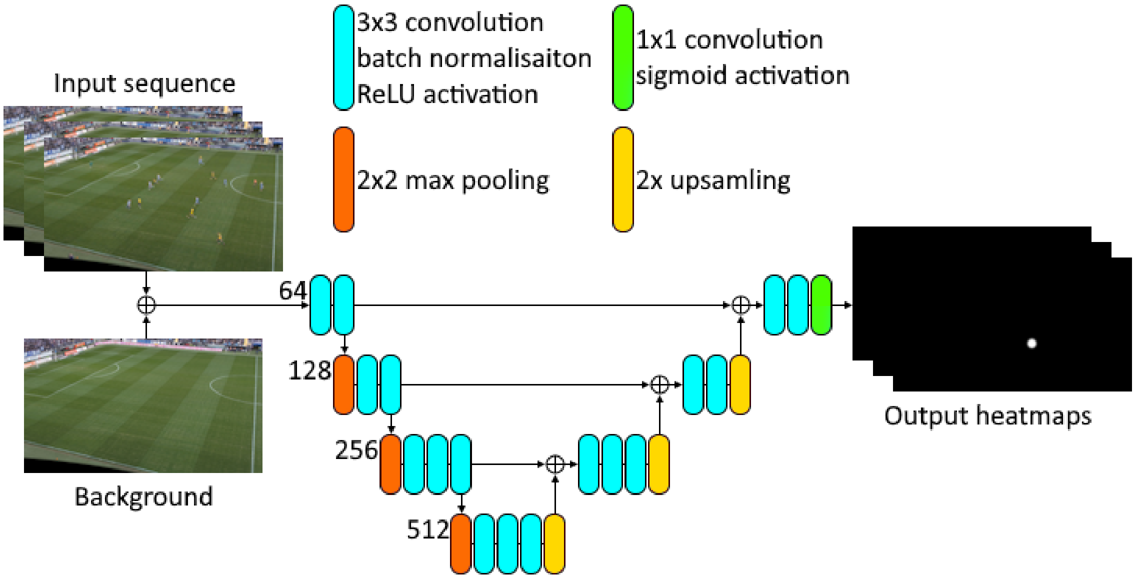


Figure 2.2: Model architecture of TrackNet V3. The numbers at each level correspond to the number of channels for each layer on that level.

static camera), and contains bounding box annotations with object id. The annotations are of eight classes, these being 'player team left', 'player team right', 'goalkeeper team left', 'goalkeeper team right', 'main referee', 'side referee', 'staff', and 'ball'. The dataset comprises one hundred 30-second clips, each shot at 25 frames per second, resulting in a total of 75,000 frames across various matches. Some images from this dataset are shown in figure 2.3.

We also annotated our own dataset for the ball, specifically. The video data for this set was provided by the football club IFK Göteborg. Unlike the SoccerNet data, our dataset uses a fixed camera perspective for each sequence. Rather than bounding boxes, the annotations in our set are simple point annotations for the center of the ball in each frame. There is also a binary indicator showing whether or not the ball is visible or hidden due to e.g. occlusion or being out of frame.

Our training set consists of video sequences from 7 separate professional matches at 1080p, 25fps. In total the training set includes 19,483 frames, roughly 13 minutes. We also have a test set consisting of eight shorter sequences, totaling 4,050 frames or 2.7 minutes. Examples of frames from our dataset can be seen in figure 2.4

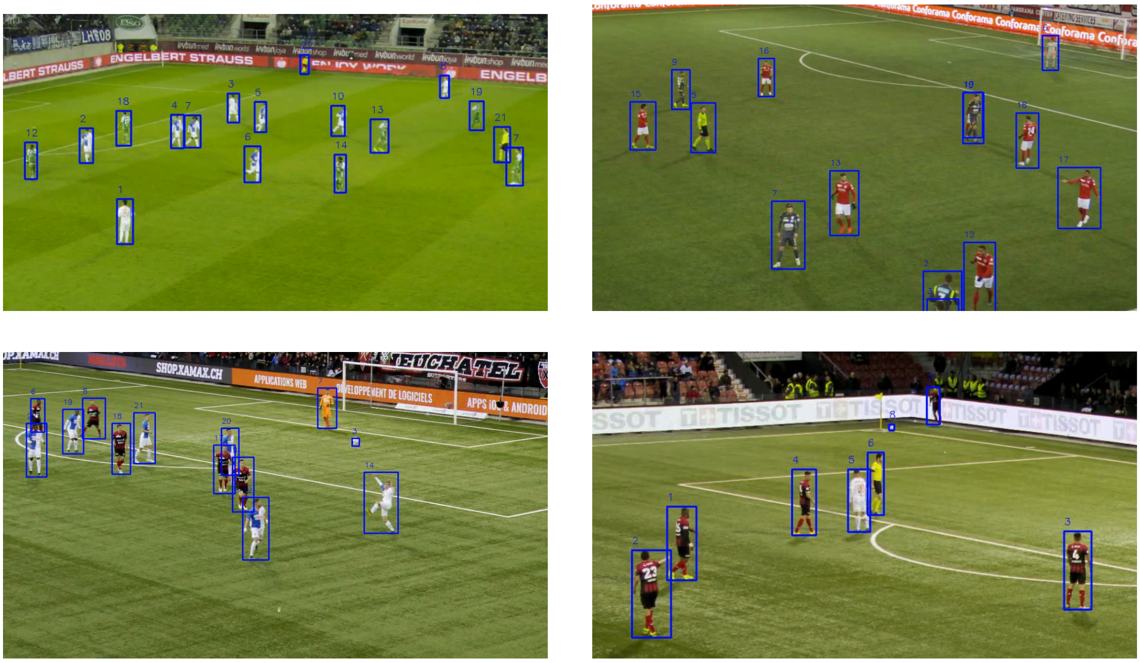


Figure 2.3: Example frames from the SoccerNet dataset with visible bounding boxes and object ID.

Figure 2.4: Example frames from our own annotated dataset. The point annotations are indicated with a small red circle.

3

Methodology

This chapter delineates the various methodologies explored to facilitate effective player and ball tracking in a football context. Different approaches have been tested when it comes to player and ball tracking, to facilitate the unique difficulties of the two tasks.

3.1 Detection

We fine-tuned YOLOv9c, a variant of YOLOv9 with 25.5 million parameters, on the SoccerNet [30] dataset using the Ultralytics python package [31]. Of the dataset's 100 matches, 10 were reserved for the validation dataset and another 10 were used for the test dataset.

Two separate models were trained: one for detecting the ball and another for detecting the players.

The models were trained for 30 epochs using default parameters provided by the Ultralytics library [31]. Multiple data augmentations were applied to the dataset, such as change in rotation, scale, translation, and color values. Figure 3.1a shows a batch used for training the ball detection model after data augmentation, and figure 3.1b shows the same for training the player detection model.

3.2 Foreground Extraction

Implementing the foreground extraction method involved several specific steps aligned with the theoretical principles discussed earlier:

- **Median Frame Calculation:** We selected multiple frames at random intervals from the video sequence to form a diverse sample. The median of these frames was computed pixel-by-pixel to establish a robust background model.
- **Grayscale Conversion:** Both the median background frame and each current frame were converted to grayscale to simplify the process of detecting changes due to motion.
- **Frame Differencing:** The absolute difference between the grayscale median background and the current frames was calculated, identifying significant changes in pixel intensity.

(a) A batch for ball detection (b) A batch for player detection

Figure 3.1: Batches used for training the detection model, after data augmentation is applied.

- **Threshold Application:** A predefined threshold of 15 was applied to the difference image. Pixels exceeding this threshold were turned white, while those below it were turned black, with 0 representing black and 255 representing white.

- **Morphological Refinement:** To improve the quality of the foreground mask, morphological operations such as dilation and erosion [32] were applied, which helped eliminate noise and close gaps in detected foreground objects.

This practical application of the median frame approach enables the effective isolation and tracking of foreground objects, crucial for tasks such as ball tracking in sports videos.

3.2.1 Removing Stationary Detections

To further refine the detections, stationary objects are removed using a calculated foreground percentage within each bounding box. The process involves the following steps:

Algorithm 1: Video Detection Filtering Algorithm

```
Result: Filtered detection results updated in CSV file
Initialization:
Read video and corresponding detections from a CSV file;
foreach frame in video do
    Generate Foreground Mask:
    Generate a foreground mask using the median background frame and the
      current frame;
    Foreground Analysis and Detection Filtering:
    foreach detection in frame do
        Extract and Analyze:
        Extract bounding box coordinates and dimensions;
        Crop the foreground mask to the area defined by the bounding box;
        Count the number of white pixels (foreground) in the cropped area;
        Calculate the percentage of foreground pixels over total pixels in the
          bounding box;
        Apply Threshold Filter:
        if foreground percentage < threshold then
            Remove the detection from the CSV file;
        end
    end
end
```

By applying this method, stationary objects that do not contribute to the dynamic aspects of the game are effectively filtered out, improving the accuracy and relevance of the detection results.

3.3 Player tracking pipeline

When it comes to player tracking we have tried two detection-based algorithms, BoT-SORT and MCF, both using detections from our finetuned YOLOv9 model. The pipelines for player tracking are depicted in figure 3.2.

BoT-SORT, which is incorporated from the BoxMOT library [33], uses several parameters that we have configured. These include:

- **New Track Threshold:** Set to 0.7,
- **High Threshold:** Set to 0.4,
- **Low Threshold:** Set to 0.1,
- **Track Life:** Defined as 4 seconds.

Detections with a confidence level below 0.1 are discarded, while those above 0.4 are considered reliable. Detections with confidence levels between these thresholds are classified as uncertain, and are only used to refine the tracks.

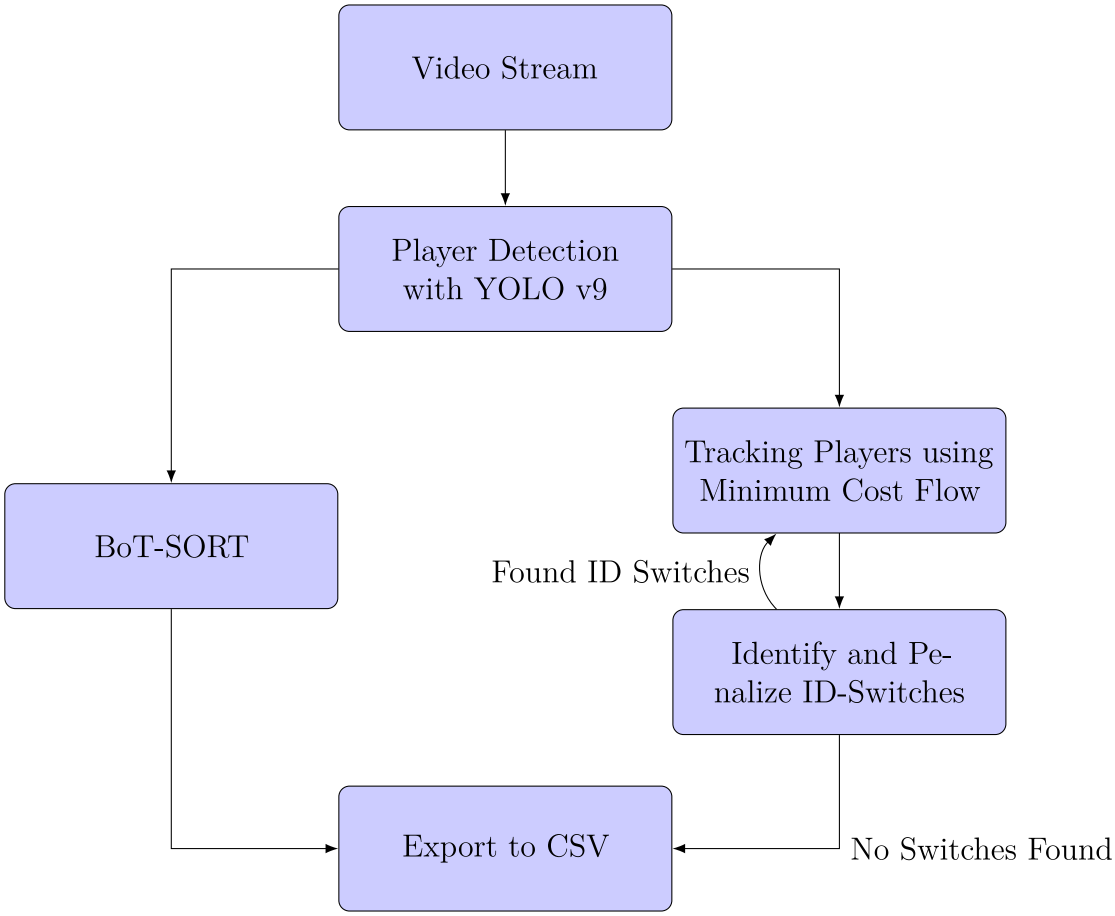


Figure 3.2: Player Tracking Pipeline: This diagram illustrates the sequential processing steps involved in tracking players in a sports video. The pipeline includes player detection, tracking using either BoT-SORT or the minimum cost flow algorithm.

The MCF algorithm used is implemented from scratch by us, the details of this implementation can be found in section 3.4.

3.4 Minimum Cost Flow implementation details

The code for this implementation is written in Python. The graph is constucted using the library Networkx [34] and the MCF problem is solved using OR-Tools [35]. This algorithm optimizes the flow through the graph to find the most feasible set of trajectories for the tracked objects. This section explains the implementation details of this method.

3.4.1 Graph Construction

A directed graph is constructed using the player and ball detections from the input data. The nodes and edges in the graph represent the states and transitions of the tracked objects across frames.

- **Nodes:**
  - Each detection is represented by two nodes: an 'in' node and an 'out' node.
  - A 'source' node and a 'sink' node are added to the graph to handle the start and end of trajectories.
- **Edges:**
  - An edge from 'source' to each detection's 'in' node with a birth cost.
  - An edge from each detection's 'out' node to 'sink' with a death cost.
  - An edge connecting each detection's 'in' node to its corresponding 'out' node with a confidence cost.
  - Edges connecting 'out' nodes of detections in one frame to 'in' nodes of detections in subsequent frames, representing possible transitions. The weight of these edges is calculated based on a custom cost function considering factors like IoU, center distance, and frame differences, the details of this cost function is presented in section 3.4.2.
  - An edge from source to sink, representing not creating a new track.

Edges are limited to only being used by one object track. By representing each detection using two nodes and a connecting edge, the nodes also become limited to a single track due to the edge. This setup ensures that each detection contributes to only one track. This dual-node representation is not the only possible implementation with these characteristics but is chosen specifically because it provides a straightforward method to enforce this exclusivity by restricting the capacity of the edges. Figure 3.3 shows a graph example involving two frames, each with three detections.

3.4.2 Cost Calculation

To reflect the likelihood and feasibility of transitions between detections, we assign the following costs to edges. Note that negative cost are advantageous.

- **Node Confidence Cost:** This cost is based on the confidence level of each detection and is applied to edges connecting the 'in' and 'out' nodes of the same detection. The cost ranges from -1 to 1; it is -1 when the confidence is 100% and 1 when the confidence is 0%. This corresponds to the cost of using a certain detection regardless of the path taken to and from it.
- **Transition Costs:** These costs are calculated for edges representing transitions between detections across different frames. The custom cost function

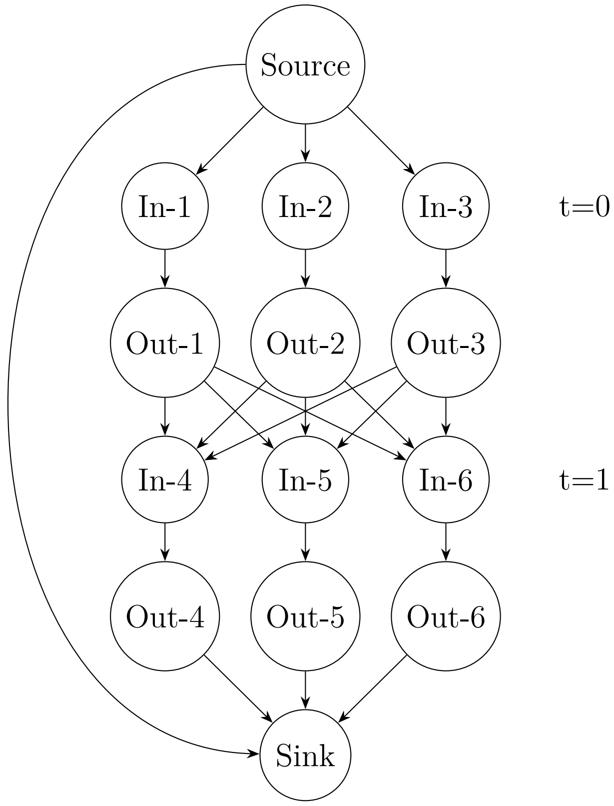


Figure 3.3: Simple illustration of a minimum cost flow graph from two frames, each with three detections numbered 1 to 6. Each detection is split into “In” and “Out” nodes.

includes:

- **IoU:** Measures the overlap between bounding boxes of detections. The cost is -1 when the boxes perfectly overlap and 0 when there is no overlap.
- **Center Distance:** Calculates the distance between the centers of bounding boxes. Edges are not connected if the distance exceeds a certain threshold, expressed in pixels. This threshold is adjusted depending on the resolution.
- **Frame Difference:** Penalizes transitions with a cost of +1 for each frame skipped, accounting for the temporal gap between detections.

- **Birth and Death Costs:** These are predefined costs set for initiating and terminating a trajectory. Applied to edges connecting the ’source’ to ’in’ nodes and ’out’ nodes to ’sink’, respectively. Both costs are set to 10. Given the frame rate of 25 fps, a good track lasting 25 frames would typically accumulate a positive score of around -30. Thus, setting the combined birth and death costs to 20 implies that a track should be followed if it can be reliably tracked for approximately one second.

- **No track cost:** The cost of going straight from source node to sink node is zero. This ensures the algorithm will only create tracks with a net negative total cost.

These cost assignments are strategically designed to initiate tracking when a track can be reliably followed for about one second, and to remove unlikely edges, thereby optimizing the solver's efficiency.

3.4.3 Improbable Connections for the Players

Post-processing involves identifying and penalizing improbable connections to improve player tracking accuracy. This includes detecting ID switches using color changes and acceleration patterns.

- **Color Consistency:** Detect possible ID switches if a tracked player's color changes between frames.
- **Acceleration Analysis:** Identify ID switches if a player exhibits rapid acceleration or deceleration, suggesting unrealistic speed changes.

A new graph is constructed with added penalties for these improbable connections:

- **Graph Update:** Identify potential ID switches and add penalties to the corresponding edges.
- **Recalculate MCF:** Solve the updated graph to find the optimized set of trajectories with penalized improbable connections.

This additional step ensures that only realistic transitions are considered, minimizing ID-switches.

3.4.4 Ball Tracking Using Minimum Cost Flow

Tracking the ball in a football game involves specific strategies to address its small size, rapid movement, and potential occlusions. Effective ball tracking comprises several crucial steps:

- **Background Removal:** Implement a foreground mask to eliminate background detections, reducing false positives by focusing on moving objects.
- **Bounding Box Adjustment:** Enlarge the bounding box of the ball to maintain effective IoU across frames for consistent detection.
- **Single Ball Tracking:** Focus the tracking mechanism to track only one ball at a time to prevent tracking errors.
- **Interpolate ball position in missing frames:** When skip connections are utilized, use interpolation to fill in the gaps. This process is explained in detail in section 3.4.5.

The complete pipeline for tracking the ball using minimum cost flow is shown in figure 3.4.

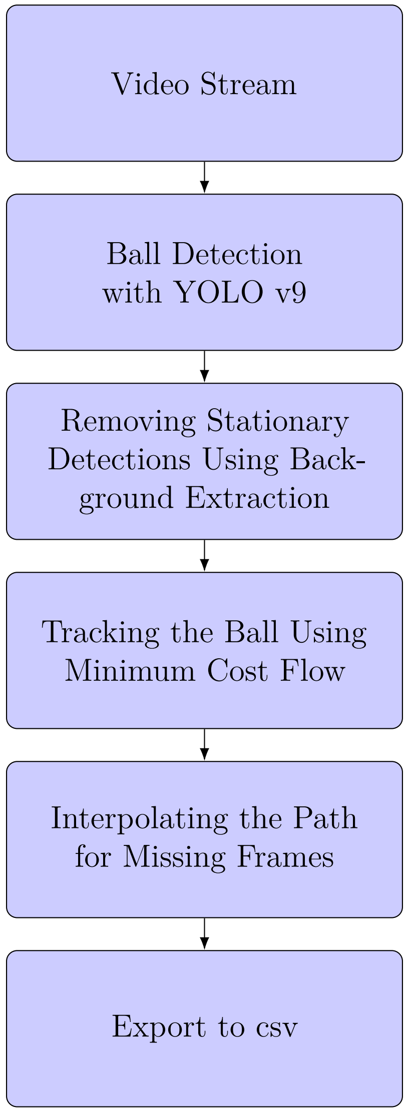


Figure 3.4: Ball Tracking Pipeline: This figure illustrates the sequence of processing steps involved in the ball tracking system, starting from the video stream and progressing through detection, filtering stationary detections, tracking, and interpolating the path for missing frames.

3.4.5 Quadratic Interpolation for Ball Path Tracking

When the ball becomes occluded or is blurred in video sequences, quadratic interpolation is employed to estimate its trajectory during these obscured periods. This method is particularly effective for capturing the parabolic motion typical of projectiles, such as a ball in flight.

Quadratic interpolation uses the ball's positions at three specific points in time, typically before and after the occlusion, denoted as t_1, t_2, and t_3, with corresponding positions $(x_1, y_1), (x_2, y_2)$, and (x_3, y_3). The objective is to fit a parabolic equation of the form:

$$y = ax^2 + bx + c$$

, where a, b, and c are coefficients determined by the interpolation.

To find these coefficients, we solve the following system of equations:

$$\begin{aligned} ax_1^2 + bx_1 + c &= y_1, \\ ax_2^2 + bx_2 + c &= y_2, \\ ax_3^2 + bx_3 + c &= y_3. \end{aligned}$$

Solving this system yields the coefficients a, b, and c, defining the quadratic curve that best fits the trajectory of the ball among the known points. This curve then allows us to estimate the balls position at any intermediate time t, enabling continuous tracking even when direct visual observation is compromised.

This method not only smooths the trajectory in the temporal domain but also significantly enhances the accuracy of the tracking system by bridging gaps where direct data are unavailable. The application of quadratic interpolation is justified by the natural parabolic paths followed by objects under the influence of gravity and initial velocities, making it a robust choice for predicting the motion of footballs in mid-air.

3.5 Tracking the ball using TrackNet

We used the TrackNet V3 [29] implementation found on GitHub [36] and trained it from scratch using the football video data annotated by us.

3.5.1 Adaption to football

As TrackNet was originally made with badminton in mind, we found it appropriate to make some adjustments to better suit our task.

The original TrackNet V3 paper uses a sequence length of eight frames for heatmap prediction and 16 frames for trajectory rectification [29]. This means they look at eight subsequent frames when making a prediction, and 16 subsequent predictions when rectifying one prediction. When it comes to football, the ball is often occluded for longer sequences and follow more complex movement patterns compared to badminton. To adhere to this, we decided to use a greater sequence length, granting the model more context for each prediction. More details about the chosen sequence length can be found in section 3.5.2 about training the model.

Ahead of training the rectification model, a mask must be generated specifying which frames rectification should be performed on. This is because the model should only rectify trajectories inside the frame, not outside. Since TrackNet V3 is based on badminton, where the shuttlecock often reaches great height, there is a check to make sure segments where the shuttlecock is above the frame do not get rectified. For our purpose, in football, the ball can often exit the frame on either side or the bottom. We therefore thought it appropriate to extend this check to cover all edges of the frame.

3.5.2 Training the main tracking model

We tried various combinations of hyperparameters when training the main TrackNet model. Out of the many parameters available, we thought the ones most appropriate to change were sequence length and learning rate.

We trained the model with sequence length 10 and 20, as longer sequences led to issues with computational resources. The learning rate varied between 1e-3 and 1e-4.

3.5.3 Training the rectification model

The InpaintNet model was trained for 300 epochs, and a learning rate scheduler was utilized which reduced the learning rate by a factor of 10 every 100 epochs. As the rectification model is significantly more lightweight than the tracking module we were not as limited by computational resources. Thus, we were able to use a longer sequence length, train for more epochs, and perform more total training runs.

Various combinations of sequence length and initial learning rate were used in different training runs. For sequence length we tried using 20, 30, and 40. We tried a multitude of initial learning rates, between 1e-2 and 1e-5.

4

Results and discussion

This chapter presents the results obtained and discusses their implications. It includes the outcomes from training YOLOv9 on both the ball and player datasets. Additionally, it features comparisons between the BoT-SORT and MCF algorithms for player tracking, and between TrackNetV3 and MCF for ball tracking. Each comparison aims to highlight the distinct advantages and limitations of the different methods.

4.1 Detection

The result of training YOLOv9c on the soccernet player dataset are shown in figure 4.1. For our purposes, the classification loss is not relevant since the model only trained on one class. Comparing the box loss, DFL, and different metrics of training and validation, it becomes apparent that the model started overfitting after epoch 10. which is why the model at epoch 10 was chosen as the best model.

The result of training YOLOv9c on the soccernet ball dataset are shown in figures 4.1. Once again, comparing the box loss and DFL of training and validation, the loss stops reducing after epoch 25. The metrics mAP50, precision, and recall also start showing a decrease, which is the reason that model is chosen to be the optimal one.

4.1.1 Player detection evaluation

The performance of the model is measured using F1-score, precision, recall and precision-recall. The graphs for them are shown in figure 4.3.

The performance curves presented in figure 4.3 illustrate a robust model capable of maintaining high levels of precision, recall, and F1-score across varying thresholds of confidence. Notably, the F1 curve (figure 4.3a) demonstrates exceptional stability at high scores until a sharp decline near the maximum confidence, indicating that the model performs with high accuracy and balance between precision and recall for most confidence levels.

The precision curve (figure 4.3b) highlights the model's effectiveness in ensuring that positive predictions are accurate, particularly as the confidence threshold increases, which is crucial for applications where false positives are particularly undesirable.

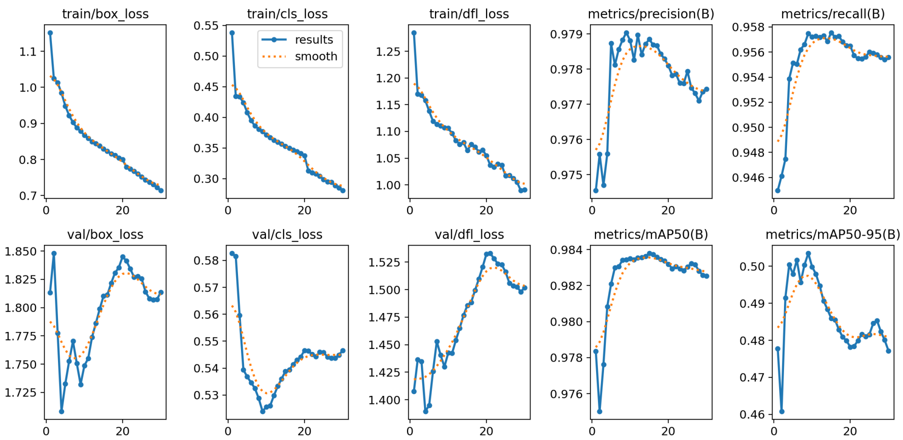


Figure 4.1: Training and validation loss per epoch for the player detection model

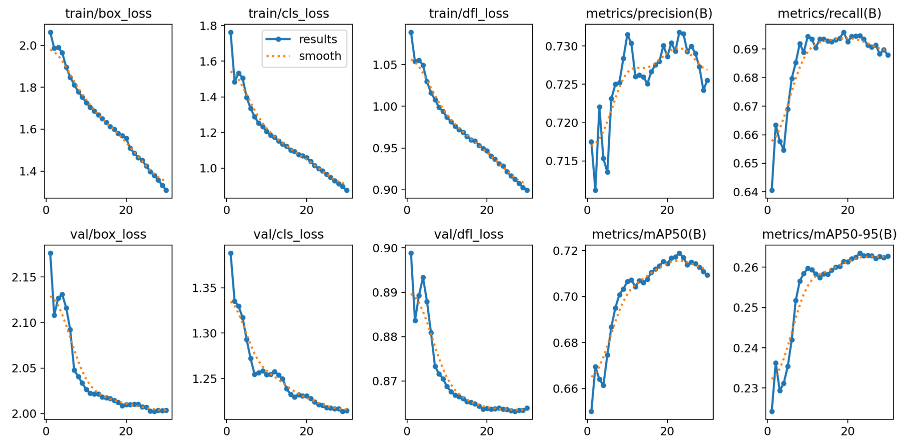
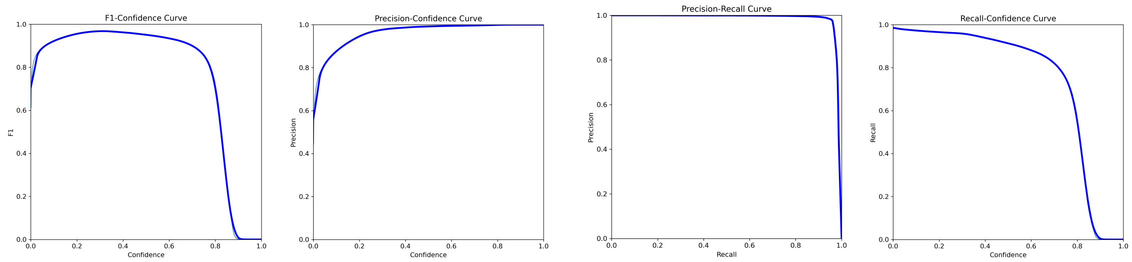


Figure 4.2: Training and validation loss per epoch for the ball detection model

The precision-recall curve (figure 4.3c) showcases an almost ideal scenario where the model achieves high precision without a significant sacrifice in recall, up until very high recall levels. This behavior is critical for scenarios where it is essential to capture as many positive instances as possible without introducing many errors.

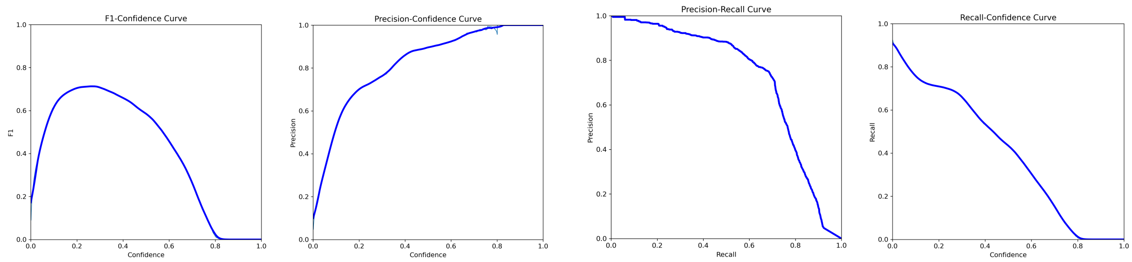
Lastly, the recall curve (figure 4.3d) indicates that the model can identify almost all positives at lower confidence thresholds, which is beneficial in applications where missing a positive detection has serious consequences.

Overall, these performance metrics suggest that the model not only learns well but also generalizes effectively to new data, balancing the trade-offs between detecting positives and minimizing false detections efficiently. This capability makes it highly suitable for deployment in critical scenarios where reliable precision and comprehensive recall are required.



(a) F1 curve (b) Precision curve (c) Precision-Recall curve (d) Recall curve

Figure 4.3: Various metrics for the player model training and evaluation.



(a) F1 curve (b) Precision curve (c) Precision-Recall curve (d) Recall curve

Figure 4.4: Various metrics for the ball model training and evaluation

4.1.2 Ball detection evaluation

The performance of the model is measured similarly as the player model performace and is shown in figure 4.4.

It immediately becomes clear that the model struggles quite a bit with detecting the ball. The f1 curve peaks at 0.7 at confidence 0.3, which is very low compared to the player model. The sharp decline after a certain confidence threshold suggests a deterioration in precision and recall, likely due to the model failing to correctly generalize the detection of the ball under varying conditions.

The precision-recall curve illustrates a struggle to balance recall and precision. The observed continuous decrease in precision as recall increases indicates that the model, in its effort to detect every possible ball, increasingly misidentifies other objects as the ball, leading to a rise in false positives. This aspect is particularly critical in our scenario because while missing a ball can be somewhat mitigated by interpolating between preceding and subsequent frames, this approach becomes ineffective when numerous false detections are present.

Overall, while the model exhibits some capability in detecting the ball, its performance in minimizing false positives is suboptimal. This is particularly concerning in environments where high precision is critical to avoid false alarms, such as in automated sports tracking systems where each detection needs to be highly reliable. The challenge lies in enhancing the model's precision without substantially sacri-

Figure 4.5: YOLOv9 detection boxes with confidence scores, identifying players' white shorts, and text in the background, as balls.

ficing recall, ensuring that it can reliably differentiate the ball from other similar objects under diverse conditions.

Visually inspecting the YOLO detections in video it is clear that it suffers from many false positive detections, often incorrectly detecting e.g. players' white shorts such as in figure 4.5. While the confidences in these detections are low, this is often true for the actual ball as well. In fact, it also struggles to detect the actual ball at all, especially when far from the camera.

Foreground Extraction

The foreground extraction process effectively isolated dynamic elements from the static background, as demonstrated in Figure 4.6. This algorithm was particularly adept at identifying and removing stationary objects, such as balls in the background, thereby ensuring that only the moving ball remained visible in the foreground. This selective masking underscores the efficiency of our foreground extraction technique.

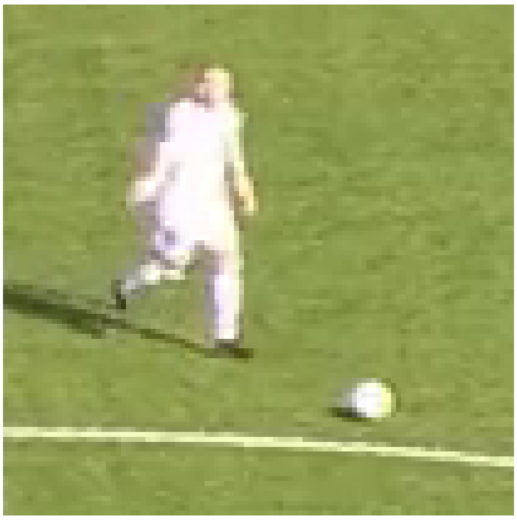
Figure 4.7b presents a zoomed-in view of the extraction, focusing on the area where the ball's background is masked. This detailed view clearly depicts the ball isolated against the background, showcasing the precision of our method.

Additionally, Figure 4.8 illustrates the detections outputted by the YOLO model. Although detections include background elements, these can be effectively eliminated using our foreground extraction technique, as demonstrated in the earlier figures. This capability highlights the practical utility of our approach in real-world scenarios where background noise must be minimized to focus on dynamic elements.

4.2 Results of tracking the players

To assess the performance of BoT-SORT and our MCF algorithm in player tracking, we employed two key metrics: MOTA and MOTP [17]. Additionally, we recorded the number of false positives and the number of identity switches during tracking.

Figure 4.6: Foreground extraction. The leftmost image displays the original frame, the middle image illustrates the background after extraction, and the rightmost image shows the foreground with the moving objects clearly isolated.



(a) Original frame with the ball

(b) Foreground with masked background

Figure 4.7: Zoomed-in view of the ball with background masked.

This evaluation utilized the SoccerNet dataset as the ground truth, applying both the BoT-SORT and the MCF algorithm to the player tracking data. For these evaluations, we used the 'motmetrics' library in Python [37].

The central box in each plot represents the interquartile range, covering the middle 50% of scores from the 25th to the 75th percentile. The median within this box marks the typical performance value, effectively dividing the dataset in half. In our case, there were no outliers, meaning the whiskers also cover 25% of the data each, extending from the quartiles.

The total number of unique IDs recorded was identical for both the MCF and BoT-SORT. Both methods only generated new IDs when necessary, such as when new players entered the scene.

Figure 4.9a illustrates the MOTA scores for both tracking algorithms. The results indicate very similar performance levels, with minor variations across different test scenarios. This metric reflects the overall tracking accuracy, where both algorithms excel by maintaining high accuracy in complex dynamic scenes. MCF holds a slight edge over BoT-SORT, especially when it comes to the worst performing samples.

Figure 4.8: Detections outputted by the YOLO model. Note the presence of background elements, which are candidates for removal via foreground extraction.

Similarly, as shown in Figure 4.9b, the MOTP scores, which assess the precision of object positioning by the trackers, are also comparable. This suggests that both algorithms are effective in precisely locating the objects across frames, though here BoT-SORT performs slightly better.

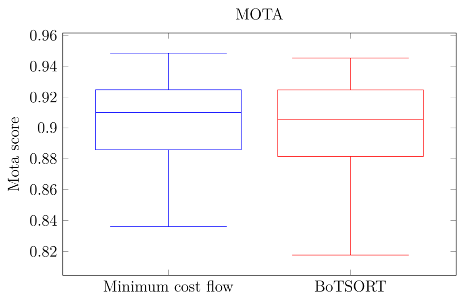
Notably, as depicted in Figure 4.9c, the MCF implementation tends to miss fewer detections compared to BoT-SORT. This means that the BoT-SORT algorithm tends to ignore certain detections while minimun cost flow makes sure to include them.

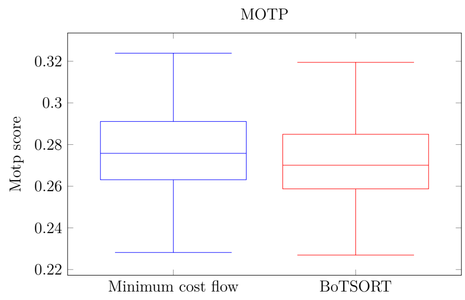
However, despite winning over BoT-SORT in other metrics, the MCF method exhibits a higher number of identity switches as shown in Figure 4.9d. This indicates a potential area for improvement in the MCF algorithm, particularly in its ability to maintain consistent identity tracking over extended sequences.

Overall, the analysis suggests that both the MCF and BoT-SORT algorithms perform comparably well in terms of tracking accuracy and precision. However, while the MCF method is more robust in maintaining detection continuity, it tends to have more identity switches compared to BoT-SORT. This disparity in identity preservation might be attributed to the use of the Kalman filter in the BoT-SORT algorithm, which effectively aids in maintaining object identity across frames. The absence of a similar mechanism in the MCF algorithm could be a contributing factor to its higher rate of identity switches.

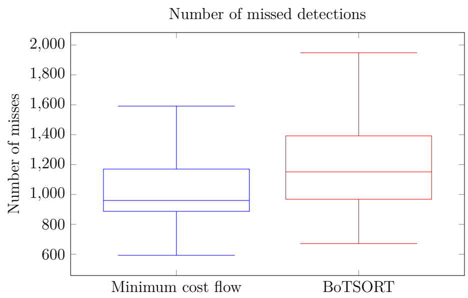
4.2.1 Challenges in Automated ID Switch Correction

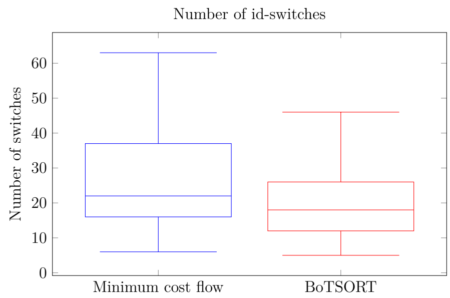
In our MCF algorithm, correcting an ID switch is feasible once the switch is detected. This assertion is supported by a manual test, during which an identified ID switch was successfully removed from the tracking data. To streamline this process, we explored automating the detection of ID switches by analyzing factors such as





(a) MOTA scores, higher is better. (b) MOTP scores, lower is better.





(c) Number of missed detections, lower is better. (d) Number of identity switches, lower is better.

Figure 4.9: Comparison of MCF and BoT-SORT algorithms in various matrics, showing similar performance.

team colors and abrupt changes in acceleration. However, due to time constraints, these solutions were not fully implemented and their effectiveness remains unproven. Further development and testing are required to enhance the robustness of these methods.

4.3 TrackNet training loss

Figure 4.10 shows the loss during training for the tracking model TrackNet V3 using different sets parameters. The graph shows that all of the models struggle with generalisation, as the validation loss does not come down along with the training loss. This suggests our own annotated data may not have been enough, and that more data is necessary to improve the model's ability to generalise. As we did not have time to collect a significant amount of additional data, we decided to move ahead with the model with sequence length 10 and learning rate 5e-4 (green curve) as it seemed to have the lowest validation loss overall.

a subset of the combinations of sequence length and learning rate tried are presented in figures 4.11 and 4.12. This subset was chosen to present as they were among the best in terms of validation loss.

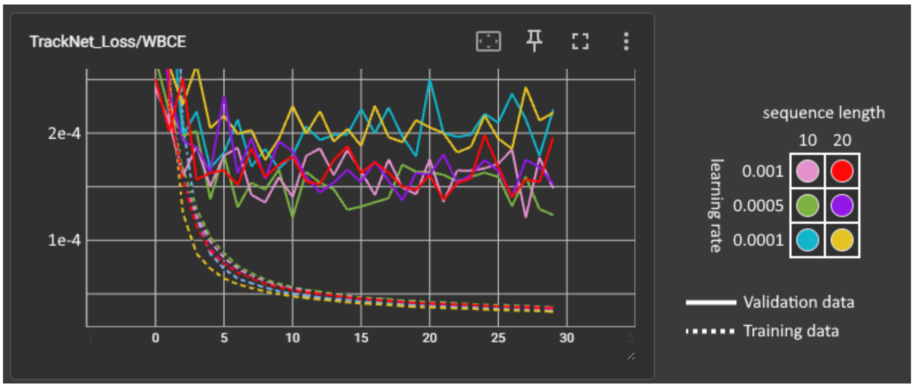
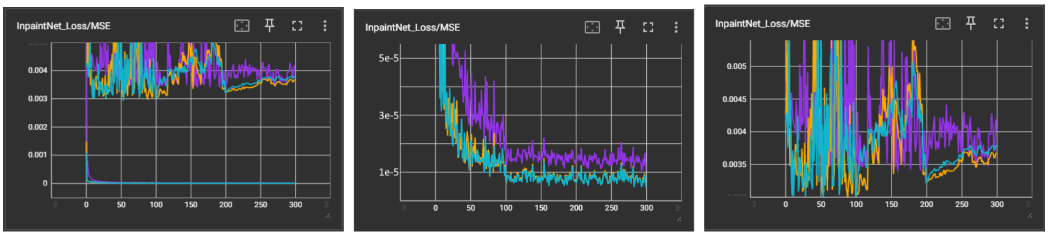


Figure 4.10: Loss for training data and validation data during six separate training runs using different sets of parameters. The x-axis is expressed in epochs.



(a) training and validation (b) only training loss (c) only validation loss

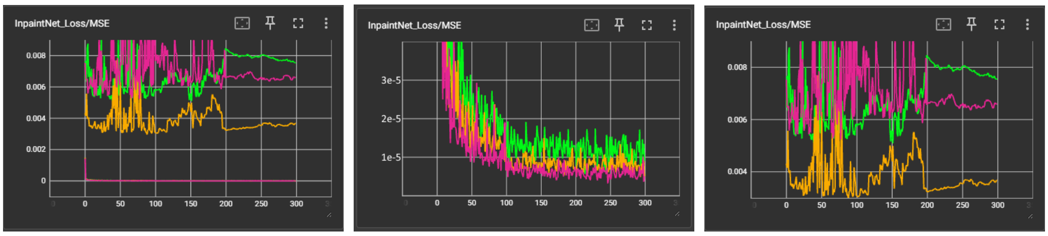
Figure 4.11: Loss for training data and validation data during three separate training runs, all using sequence length 30 but different initial learning rates. The initial learning rates are 5e-5 (purple), 5e-4 (orange), and 1e-3 (blue). The x-axis is expressed in epochs.

To no surprise, both of these figures show the same issue as the tracking module, namely a failure to generalise. A sequence length of 30 seems to have generalised the best, though it shows the worst loss on the training data, where sequence length 40 is the best. Due to its superior generalisation we decided to settle with the model of sequence length 30.

4.4 Evaluation of ball methods on test set

For ball tracking, it was essential to employ a common evaluation strategy that was suitable for both methods under comparison. Given that we have not treated the ball tracking as a MOT problem, since we are only predicting one trajectory, we treated this as a classification problem. If the predicted position is within a certain distance threshold of the ground truth label, we consider it a correct prediction. We opted not to use distance based metrics such as mean squared error as it does not matter where the model incorrectly predicts the ball, all incorrect predictions are equally incorrect. More comprehensively, this is how we decided to categorise each prediction:

- **True positive:** The model has predicted a location and it is within threshold.



(a) training and validation (b) only training loss (c) only validation loss

Figure 4.12: Loss for validation data during three separate training runs using initial learning rate of 5e-4. The sequence lengths are 20 (green), 30 (orange), and 40 (pink).

Method	True positives	Wrong location	False negatives
YOLO+MCF	2488	66	1086
TrackNet	1940	1126	568

Method	True negatives	False positives
YOLO+MCF	396	20
TrackNet	162	254

Table 4.1: Total number of predictions of each category across all eight test set sequences, using a distance threshold of 20 pixels.

- **Wrong location:** The model has predicted a location but it is not within threshold.
- **False negative:** The model has not predicted a location while there is a ball in the frame.
- **True negative:** The model has not predicted a location and there is no visible ball.
- **False positive:** The model has predicted a location while there is no ball.

Considering the average width of the ball is approximately 40 pixels in diameter, we established a threshold of 20 pixels. The evaluation was conducted using the test set we annotated ourselves, more detail about it can be found in section 2.10.

Table 4.1 shows the total number of predictions of each cateogory for both of our tracking methods. By these metrics YOLO+MCF outperforms TrackNet in all but false negatives. These numbers indicate that YOLO+MCF is better at finding the correct ball location, as TrackNet has a very high number of predictions in the wrong location.

To achieve a better overview of the performances of the complete MCF ball tracking pipeline versus TrackNet V3, we calculated accuracy, precision, recall, F1-score, and specificity. Since we have recorded wrong locations in addition to false negatives and false positives, we decided to slightly alter the formulas for precision and recall by incorporating the number of wrong location predictions in the denominators:

- **Precision:** Reflects the ratio of correctly predicted positive observations to

Method	Accuracy	Precision	Recall
YOLO+MCF	72.9% ± 13.3%	96.8% ± 4.9%	70.2% ± 15.4%
TrackNet	54.3% ± 20.3%	57.4% ± 15.7%	55.3% ± 21.1%

Method	F1	Specificity
YOLO+MCF	80.7% ± 11.1%	95.2% ± 6.3%
TrackNet	55.7% ± 19.0%	36.8% ± 41.0%

Table 4.2: Performance Metrics Comparison across eight annotated sequences using a distance threshold of 20 pixels. The shown values represent the mean, with the ± symbol indicating the standard deviation.

the total predicted positives.

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives} + \text{Wrong locations}}.$$

- **Recall (Sensitivity):** Measures the ratio of correctly predicted positive observations to all actual positives.

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives} + \text{Wrong locations}}.$$

These modifications were done as a wrong location prediction can be considered both an actual positive and a predicted positive, despite being an incorrect prediction. The rest of the metrics are calculated as usual, the way they are presented in section 2.6. Note that F1 is affected by these changes as it is based on precision and recall. Specificity was not changed as the metric is only meant to concern actual negative samples. The results are presented in Table 4.2.

From these quantitative results we can see that our MCF implementation generally performs better, outperforming TrackNet in every metric. Neither of these performances are satisfactory to create a convincing recreation of sequences.

While the performance of YOLO+MCF at this stage is more promising, it is important to note that the TrackNet model has a significantly higher standard deviation on values across the eight separate sequences. This is another sign pointing to poor generalisation, leading to different performance on different data. We hypothesise that the TrackNet architecture has a high potential to yield better results if it is trained on more data.

Since TrackNet has a high number of wrong location predictions, we wanted to see if these were close to the correct location. To this end, we ran the evaluation again but with increased distance thresholds of 100 and 150 pixels for correct predictions. The result of this change can be seen in table 4.3. We see here that a significant number of wrong location predictions are somewhat close to the correct position, though the majority are farther away still.

Method	20 pixels	100 pixels	150 pixels
YOLO+MCF	66	55	46
TrackNet	1126	868	777

Table 4.3: Total number of predictions with the wrong location across all eight test set sequences, using different distance thresholds of 20, 100, and 150 pixels.

4.4.1 Visual analysis of TrackNet result

By observing the predicted trajectories overlaid on top of the original video, we can see that the model accurately predicts the trajectory while the ball is traveling freely, but struggles if the ball is still, moving slowly, or being dribbled. Examples of these scenarios can be seen in figure 4.13. We take this to mean that the model did not struggle to learn to find the ball in movement with a grassy background. However, the model's movement-based design predictably struggles to detect still balls. Furthermore, situations where the ball is being dribbled are too visually distinct from one another for the model to successfully generalise across different players seen from different angles.

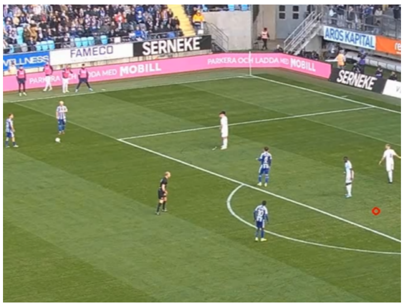
Figure 4.13c shows the model missing the ball when it gets very close to the camera and appears larger than usual. This may indicate that the model has not seen enough large balls during training, yet another sign of poor generalisation.

Figure 4.13d on the other hand shows the model defaulting its prediction to a spot close to the middle of the screen while the ball is not in motion. This is something that we have observed happening occasionally, but it is unclear to us why this happens. Presumably, it has learnt that this location is generally more likely to contain the ball compared to other parts of the frame. Other times, the model successfully does not predict a ball when the ball goes off-screen. This may be fixable with more training data, or it may require tuning parameters such as the weight in the weighted binary cross entropy loss, to make the model more likely to choose not to predict a ball.

(a) A correct prediction.

(b) Model missing dribbled ball.)

(c) Model missing large ball (close to camera).



(d) Model missing motionless ball.

Figure 4.13: Positions predicted by our TrackNet model overlaid on top of the original video. Videos are from the test set annotated by us.

5

Conclusion

In this thesis, we explored the complex task of tracking players and the ball in football videos, leveraging advanced computer vision techniques and deep learning models. Our primary goal was achieving accurate and reliable object detection and tracking in the dynamic and visually complex environment of football games. By doing so, we aimed to contribute to the broader project of reconstructing football sequences in a virtual reality environment, thus improving the tools available for sports analysis and training.

Throughout our research, we evaluated several methods and models, including YOLOv9 for object detection and the BoT-SORT and MCF algorithms for tracking. Additionally, we explored the potential of the TrackNet V3 model for ball trajectory prediction.

5.1 Summary of Findings

The MCF algorithm demonstrated robust performance in tracking players, maintaining high precision and effectively minimizing false positives. However, it exhibited a higher frequency of identity switches compared to the BoT-SORT algorithm, which indicates a need for improved mechanisms to maintain player identity across frames.

Tracking the ball proved more challenging due to its small size, rapid movement, and frequent occlusions. The combination of YOLOv9 for detection and MCF for tracking outperformed TrackNet V3 in terms of accuracy and overall stability. However, TrackNet V3 showed potential for improvement when it comes to generalisation, suggesting it could outperform detection-based tracking given more parameter tuning and training data.

5.2 Contributions to the Field

Our work contributes to the understanding and development of object tracking in sports, particularly in football. By identifying specific challenges such as ID-switching and detecting the ball, we have provided insights that could inform the development of more robust tracking systems. These systems are crucial for applications in sports analytics, coaching, and even in creating immersive virtual reality experiences.

6

Future work

This chapter details areas of potential improvement and additional ideas we would have liked to explore, given more time.

6.1 Correcting ID-switches for player Minimum Cost Flow

The MCF algorithm has shown promising results in player tracking but struggles with ID switches. Future work should focus on developing and integrating methods to more effectively identify and correct identity switches. Once an identity switch is detected, the MCF algorithm can be rerun with penalisations applied to the edges that led to the ID switch. These strategies are aimed at refining the identity tracking capabilities of the MCF method, thereby enhancing its applicability and reliability for real-world player tracking scenarios.

To further enhance its accuracy and robustness, we propose the integration of additional information sources into the algorithm. These enhancements could significantly improve the performance of the tracking system by providing more discriminative features and reducing the incidence of errors.

6.1.1 Jersey Colors and Deep Features Learned by Neural Networks

Incorporating jersey colors into the tracking algorithm could provide a straightforward and effective method for distinguishing between players, especially in sports with clearly defined team uniforms. Additionally, leveraging deep features learned by neural networks could allow the algorithm to capture more complex patterns and player-specific characteristics that are not immediately apparent to simpler models.

6.1.2 Jersey Numbers

Jersey numbers offer a direct method of identifying players. By training an optical character recognition (OCR) component within our system, we can continuously update player IDs based on the visibility of their jersey numbers. This would likely reduce the number of ID switches, particularly in high-quality footage where numbers are clearly visible.

6.1.3 Pose Estimation

Integrating pose estimation can provide contextual clues that enhance the tracking accuracy. By understanding the orientation and actions of players, the system can better predict their movements and interactions, reducing the likelihood of misidentifications, especially in crowded scenes. Pose estimation has been explored separately in the football in VR project [1].

6.1.4 Speed Analysis

Analyzing the speed and direction of movement can also be instrumental in improving tracking. Sudden changes in speed or atypical movement patterns can indicate potential errors in player identification or tracking, prompting a re-evaluation of the tracking decision at that instant. Kalman Filters have potential to be useful in this area.

6.2 Better tracking of ball

We have seen that YOLO+MCF is the better performer when tracking the ball, though TrackNet has shown more signs of potential for improvement. It is our opinion that both of these methods are worth exploring further.

Regarding the detection based MCF tracking, the bottleneck is in YOLO's struggle to detect the ball. Future work should go into finding ways to improve detection, whether that's by improving the performance of YOLO or by testing state-of-the-art two-stage detectors such as Detectron2 [38].

For TrackNet, the biggest issue is that the validation loss does not follow the training loss. Effort should be put into building a larger training set, as well as using data augmentation to further increase the variability of the training data. Beyond this, it is also worth exploring how the model itself can be tweaked to improve performance, starting with tuning hyperparameters such as the weight in the WBCE loss.

6.3 Predicting the ball's position in 3D space

During play, the ball often gains considerable height when kicked. To make a convincing recreation of football sequences, the ball position must be estimated in three dimensions. Once ball tracking is reliable in the two dimensional video plane, translation to three dimensional coordinates relative to the field is something that should be explored.

The thesis group responsible for pose estimation have explored the possibility of improving pose predictions using multiple cameras at different angles [1]. By using accurate camera calibration that estimates the cameras' positions and properties such as focal length, they can associate one person in two perspectives and use the two angles to achieve more accurate predictions. Similarly, one could use the camera calibration and the ball's pixel coordinates from two angles to triangulate the precise position of the ball.

Bibliography

[1] J. Osterman and O. Sjögren, "3d pose estimation of football players," To be published concurrently, Master's thesis, Chalmers University of Technology, 2024.

[2] F. Anjou and A. Ekström, "Football analysis in vr, Texture estimation with differentiable rendering and diffusion models," To be published concurrently, Master's thesis, Chalmers University of Technology, 2024.

[3] A. F. Agarap, *Deep learning using rectified linear units (relu)*, 2019. arXiv: `1803.08375 [cs.NE]`.

[4] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning*, pmlr, 2015, pp. 448–456.

[5] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2014.

[6] R. Girshick, "Fast r-cnn," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Dec. 2015.

[7] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *Advances in neural information processing systems*, vol. 28, 2015.

[8] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2016.

[9] W. Liu, D. Anguelov, D. Erhan, *et al.*, "Ssd: Single shot multibox detector," in *Computer Vision – ECCV 2016*, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds., Cham: Springer International Publishing, 2016, pp. 21–37, ISBN: 978-3-319-46448-0.

[10] J. Redmon and A. Farhadi, *Yolov3: An incremental improvement*, 2018. arXiv: `1804.02767 [cs.CV]`.

[11] C.-Y. Wang, I.-H. Yeh, and H.-Y. M. Liao, *Yolov9: Learning what you want to learn using programmable gradient information*, 2024. arXiv: `2402.13616 [cs.CV]`.

[12] 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. 103 448, 2021, ISSN: 0004-3702. DOI: `https://doi.org/10.1016/j.artint.2020.103448`.

[Online]. Available: https://www.sciencedirect.com/science/article/pii/S0004370220301958.
[13] G. Welch, G. Bishop, *et al.*, "An introduction to the kalman filter," 1995.
[14] N. Aharon, R. Orfaig, and B.-Z. Bobrovsky, *Bot-sort: Robust associations multi-pedestrian tracking*, 2022. arXiv: 2206.14651 [cs.CV].
[15] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2015.
[16] N. Wojke, A. Bewley, and D. Paulus, "Simple online and realtime tracking with a deep association metric," in *2017 IEEE international conference on image processing (ICIP)*, IEEE, 2017, pp. 3645–3649.
[17] 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.
[18] Á. F. García-Fernández, A. S. Rahmathullah, and L. Svensson, "A metric on the space of finite sets of trajectories for evaluation of multi-target tracking algorithms," *IEEE Transactions on Signal Processing*, vol. 68, pp. 3917–3928, 2020. DOI: 10.1109/TSP.2020.3005309.
[19] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft, "Simple online and realtime tracking," in *2016 IEEE international conference on image processing (ICIP)*, IEEE, 2016, pp. 3464–3468.
[20] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval Research Logistics Quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955. DOI: https://doi.org/10.1002/nav.3800020109. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/nav.3800020109. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/nav.3800020109.
[21] Y. Zhang, P. Sun, Y. Jiang, *et al.*, "Bytetrack: Multi-object tracking by associating every detection box," in *Computer Vision – ECCV 2022*, S. Avidan, G. Brostow, M. Cissé, G. M. Farinella, and T. Hassner, Eds., Cham: Springer Nature Switzerland, 2022, pp. 1–21, ISBN: 978-3-031-20047-2.
[22] E. Ristani, F. Solera, R. Zou, R. Cucchiara, and C. Tomasi, "Performance measures and a data set for multi-target, multi-camera tracking," in *European conference on computer vision*, Springer, 2016, pp. 17–35.
[23] J. Luiten, A. Osep, P. Dendorfer, *et al.*, "Hota: A higher order metric for evaluating multi-object tracking," *International Journal of Computer Vision*, vol. 129, no. 2, pp. 548–578, 2021.
[24] A. Milan, L. Leal-Taixe, I. Reid, S. Roth, and K. Schindler, *Mot16: A benchmark for multi-object tracking*, 2016. arXiv: 1603.00831 [cs.CV].
[25] P. Dendorfer, H. Rezatofighi, A. Milan, *et al.*, *Mot20: A benchmark for multi object tracking in crowded scenes*, 2020. arXiv: 2003.09003 [cs.CV].
[26] M. Wang, X. Li, P. Liu, K. Xu, and Z. Fu, "Multiple object tracking by multi-feature combination based on min-cost network flow," in *2016 IEEE 13th International Conference on Signal Processing (ICSP)*, 2016, pp. 714–718. DOI: 10.1109/ICSP.2016.7877925.
[27] Y.-C. Huang, I.-N. Liao, C.-H. Chen, T.-U. k, and W.-C. Peng, "Tracknet: A deep learning network for tracking high-speed and tiny objects in sports

applications," in *2019 16th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, 2019, pp. 1–8. DOI: 10.1109/AVSS.2019.8909871.

[28] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, N. Navab, J. Hornegger, W. M. Wells, and A. F. Frangi, Eds., Cham: Springer International Publishing, 2015, pp. 234–241, ISBN: 978-3-319-24574-4.

[29] Y.-J. Chen and Y.-S. Wang, "Tracknetv3: Enhancing shuttlecock tracking with augmentations and trajectory rectification," in *Proceedings of the 5th ACM International Conference on Multimedia in Asia*, ser. MMAsia '23, Tainan, Taiwan: Association for Computing Machinery, 2024, ISBN: 979-8-400-70205-1. DOI: 10.1145/3595916.3626370. [Online]. Available: https://doi.org/10.1145/3595916.3626370.

[30] A. Cioppa, S. Giancola, A. Deliege, *et al.*, "Soccernet-tracking: Multiple object tracking dataset and benchmark in soccer videos," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 3491–3502.

[31] Ultralytics, *Ultralytics documentation*, Web Page, Accessed: 2023-05-29, 2023. [Online]. Available: https://docs.ultralytics.com/.

[32] D. Soille, "Erosion and dilation," in Jan. 2004, pp. 63–103, ISBN: 978-3-642-07696-1. DOI: 10.1007/978-3-662-05088-0_3.

[33] M. Broström, *Yolo tracking*, https://github.com/mikel-brostrom/yolo_tracking, Accessed: 2024-02-01, 2023.

[34] A. Hagberg, P. Swart, and D. S Chult, "Exploring network structure, dynamics, and function using networkx," Los Alamos National Lab.(LANL), Los Alamos, NM (United States), Tech. Rep., 2008.

[35] L. Perron and V. Furnon, *Or-tools*, version v9.10, Google, May 7, 2024. [Online]. Available: https://developers.google.com/optimization/.

[36] "Tracknetv3." Github repository. (2024), [Online]. Available: https://github.com/qaz812345/TrackNetV3.

[37] C. Heindl, *Py-motmetrics: Python measures of tracking performance*, Accessed: 2024-04-27, 2024. [Online]. Available: https://github.com/cheind/py-motmetrics.

[38] "Detectron2." Github repository. (2024), [Online]. Available: https://github.com/facebookresearch/detectron2.