UNIVERSITÀ DEGLI STUDI DI MILANO

FACOLTÀ DI SCIENZE POLITICHE, ECONOMICHE E SOCIALI

Master's Degree Course in Data Science for Economics

EXTRACTING SOCCER BALL COORDINATES IN NON-BROADCAST IMAGES THROUGH LOCALISATION AND HOMOGRAPHY TECHNIQUES

Supervisor: Prof Alfio FERRARA

Co-supervisors: Dott. Sergio PICASCIA, Dott. Alberto RICCARDI,

Thesis by:
Leonardo ACQUAROLI
Student ID: 13965A

Academic Year 2023-2024

Acknowledgments

ACKNOWLEDGMENTS

Contents

Acknowledgments **ii**

1 Introduction **2**
1.1 Why ball localisation? . . . 3
1.2 The architecture . . . 6
1.3 Thesis structure . . . 7

2 State of the Art **8**
2.1 Object Detection . . . 8
2.1.1 Grounding DINO . . . 8
2.1.2 YOLO-World . . . 9
2.1.3 YOLOv9 . . . 9
2.1.4 RT-DETR (Real-Time DEtection TRansformer) . . . 9
2.1.5 YOLOv10 . . . 10
2.2 Keypoints detection . . . 11
2.2.1 TVCalib . . . 11
2.2.2 YOLOv8-pose . . . 12

3 Dataset **13**
3.1 Ball detection . . . 13
3.1.1 Roboflow - Football ball detection . . . 13
3.1.2 Soccerment's dataset . . . 13
3.1.3 Test dataset . . . 14
3.2 Pitch detection . . . 15
3.2.1 Dataset WC14 - TVCalib . . . 15
3.2.2 Roboflow - Football pitch detection . . . 15
3.2.3 Roboflow - Campo futebol keypoint classes v15 . . . 15
3.3 Ball localisation . . . 16
3.3.1 Soccerment's tagging data . . . 16
3.3.2 Refined Soccerment's tagging data . . . 16

4 Ball detection **17**
4.1 Ball detection for automatic labeling . . . 17
4.1.1 Model selection . . . 18
4.1.2 Automatic labeling test results . . . 19
4.1.3 The labeling process . . . 20
4.2 Ball detector . . . 21

4.2.1 Model selection . . . 21
4.2.2 Training . . . 22

5 Pitch detection **27**
5.1 Model selection . . . 27
5.2 Dataset augmentation . . . 29
5.3 Training . . . 29

6 Homography **32**
6.1 How does homography work? . . . 32
6.2 How is the homography matrix obtained? . . . 33
6.3 Application of homography in the football context . . . 33

7 The pipeline **35**
7.1 Implementation . . . 35

8 Results **37**
8.1 Correct pipeline execution . . . 37
8.2 Ball position accuracy . . . 38
8.3 Inference time . . . 41

9 Conclusions and future work **42**

Abstract

This thesis presents the development of an automated pipeline for extracting soccer ball coordinates from non-broadcast images, aimed at speeding up the collection of data from soccer games videos - i.e. tagging.
The pipeline consists of three modules: ball detection using a fine-tuned YOLOv10 model, pitch keypoints detection employing a YOLOv8-pose model, and homography transformation to map ball coordinates to a predefined 2D pitch representation.
The ball detector and the pitch detector were trained using proprietary footage with a majority of non-broadcast frames.

The evaluation of the pipeline showed that in 14.5% of the frames in the dataset the ball is successfully localised.
Additionally, a further analysis on ball positioning accuracy conducted using a 100×100 pitch model reveals that, for detections with error distance smaller than 15 pitch units, the system achieved a mean error of 8.74 units.
Accuracy must be conciliated with low inference time for the pipeline to empower tagging operators, thus using GPUs reduces runtime to under 10 milliseconds per frame.

Future work includes enhancing the pitch detector's robustness, exploring microservices architecture deployment, and investigating spatiotemporal methods.

Chapter 1

Introduction

The collection of granular data in soccer is, still in 2024, mostly a labor intensive and time consuming process which needs a deploy of operators watching a video and each annotating a piece of information about every event in the game in an assembly line fashion. When, instead, the collection is made by a single operator, the annotation of all the events of a single match usually takes 3 to 5 hours. [1]

Figure 1: Tagging center in live operation. Several persons involved in a cumbersome, and tedious manual process. Image taken from [2]).

The data collection process is called, in the jargon, ***tagging***. In particular, tagging means collecting the details about each action occurring in a football[1] game that either is performed by a player on the ball - in which case is called on-ball event - or comes after a whistle by the referee. Examples of this last type of events are a card, the start and the end of the game, a substitution or VAR assistance but in this work the focus is on on-ball events which are passes, receptions, tackles, shots, etc. which are the most frequent ones and represent the data allowing technical and tactical analysis. [3, 4, 5, 6, 7]
For every on-ball event several attributes are tagged. For example, if a player takes a shot the operator, in addition to the player identity, has to annotate the body part which the shot is taken with, if it is a volley, if the player is under pressure, and possibly other qualifiers. Among this attributes, a fundamental one, common to every on-ball event, is the ball position in the pitch when

[1] In this paper soccer and football will be used as synonyms following the UK English meaning.

the event happened as bi-dimensional coordinates in a 2D map of the pitch that we will call the ***radar*** from now on.

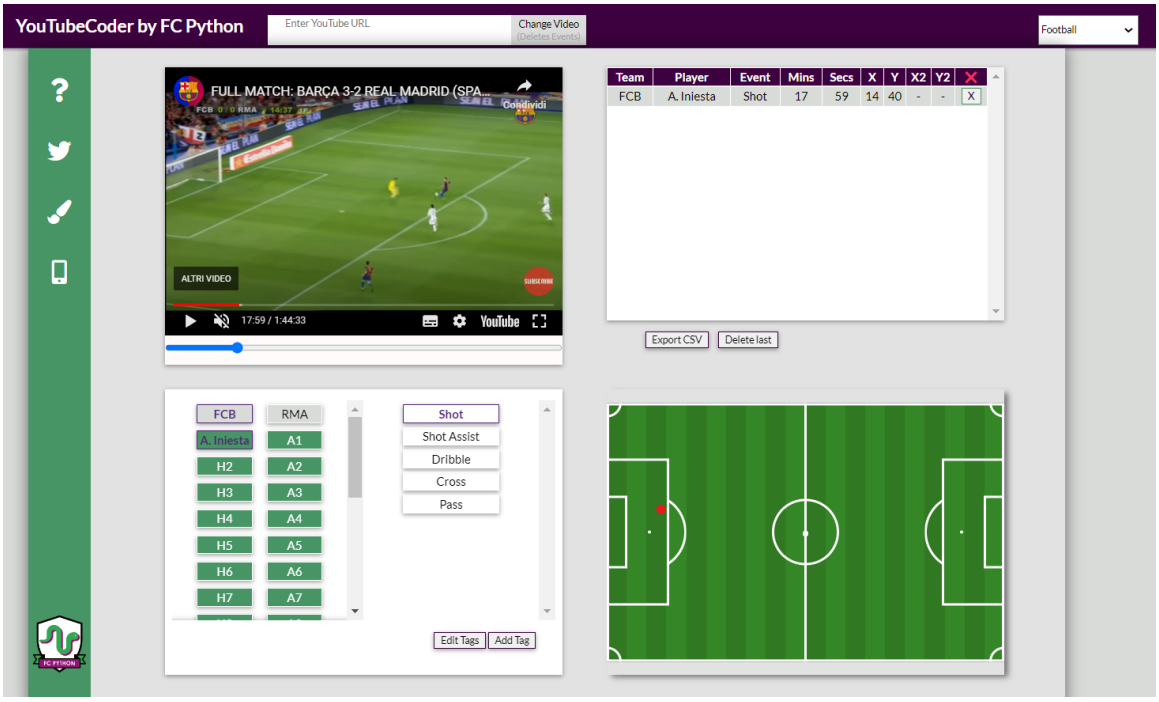

Figure 2: The ball position annotation in the radar of the open-source tagging tool by FC Python.

As reported in [1], the taggers usually make use of special keyboards, remotes or shortcuts to interact with the tagging platforms without taking their hands off the keyboard in order to be as time efficient as they can in each event annotation. Yet the ball position annotation needs the operator to move the hand to a mouse or a pad, locate the cursor over the correct position and click. This action can take some seconds per event and, since we can expect to have between 1'500 and 2'000 events per game (1'675 in [1]), it can highly impact the duration of the process.

1.1 Why ball localisation?

The aim of this project is, then, to create a pipeline that speeds up the tagging process by automatically extracting the ball position from the frames of the game.
This paper is the foundation stone of a broader project started up by, Soccerment[2], a Football Data Intelligence company based in Milan, that aims to integrate automatic annotation systems within the company's existing video-tagging software.
The choice to begin with the extraction of the ball position was made after examining two aspects:

- The potential impact on time reduction

[2] https://soccerment.com/ - Accessed on August 9^{th}, 2024.

- The likelihood of successfully accomplishing the task

For what concerns the potential time cut of the tagging process, in addition to what was said above, managing to automatically extract the ball location, a common element of all the events, means speeding up the annotation of every game independently of the type of events occurring. Indeed, Soccerment's internal estimations[3] report that, avoiding to tag the position, the length of a tagging process decreases from approximately 230 to 130 minutes (from almost 4 hours to a bit more than 2 hours) with a potential reduction of 45%.

The second aspect of the decision, instead, needs a deeper contextualization because working on non-broadcast images brings an additional layer of complexity with itself.
Many trials have been attempted during the last 20 years to automate the extraction of information from football broadcast videos. We can divide them into three types of experiments:

- Specific events recognition
- Clips extraction
- Object detection

Since this work focuses solely on on-ball events, players and ball tracking is left out even if is a major task of the automated video analysis landscape.

Specific events recognition: Different research teams tried to solve the very complex task of action recognition. The difficulty here stems from the dynamic nature of a player's action, which is generally hard to encode in a single frame and from the variety and the ambiguity of the situations in which the same type of event occurs. D'Orazio et al. [8] developed a ball detection and tracking system which is the ancestor of the modern Goal-line technology[4] exploiting two cameras on each side of the pitch and using Circle Hough Transform (CHT), region segmentation and a neural classifier on the restricted area. Fassmeyer et al. [9] introduced the use of Variational Autoencoders to learn a meaningful feature representation of the frames and a Support Vector Machine classifier acting in this feature space but utilizing only a few of the features encoded by the VAE. In this way they were able to identify cornerkicks, crosses and counterattacks with great precision. Another attempt to recognize specific actions was made by Cintia and Pappalardo [10] with the development of PassNet, a framework that combines a set of artificial neural networks that perform feature extraction from video streams, object detection to identify the positions of the ball and the players, and classification of frame sequences to predict if they show a pass. In [11], instead, several algorithms are used to capture all the slow-motion segments in a game, the goals and the slow-motion segments classified according to object-based features. Finally Theagarajan and Bhanu [12] managed to extract tactical statistics for a player. In particular duration of ball possession, number of successful passes and number of successful steals. This is done by training Convolutional Neural Networks (CNNs) to localize and track the players on the field, classify the team of a detected player, identify the player controlling the ball and finally pooling all the information extracted. Also a variant of Deep Convolutional Generative Adversarial Networks (DCGAN) was subsequently used to improve generalization ability. All these works open a door on the multiple possibilities at disposal to process images and extract useful information. Yet, for the sake of event tagging only the automatic identification of passes with PassNet is practically useful in the annotation of broadcast videos. No method was trained or tested on non-broadcast images.

[3] Conducted by the Football Data Analysts Federico Soldano and Alessandro Mondini de Focatiis
[4] https://en.wikipedia.org/wiki/Goal-line_technology - Accessed on August 9^{th}, 2024.

Clips extraction: The task of highlights retrieval is mainly thought for two types of subjects: match analysts who use them to show specific footage to players and coaches; TV shows for post-game analysis and in general for the audience's entertainment.
One of the ground breaking papers in this field is [13] by Assfalg et al. published in 2003. In this work different techniques were used to extract highlight clips and their correspondent classification. The authors exploited image segmentation, homography application with segmented lines instead of points and a rule-based system to check the several models that classify each clip into the correspondent type.
Three more recent works that promisingly explore this task are instead [14], [15] and [2].
By integrating object detection and tracking, Optical Character Recognition (OCR), and color analysis, PlayerTV [14] facilitates the generation of player-specific highlight clips from extensive game footage and serves them through an interactive Graphical User Interface to allow football professionals for a streamlined use.
In [15] the authors tackle three tasks: play-back detection, soccer event recognition, and commentator excitement classification, the first and second being of matter ofr our scope. For play-back detection, they frame it as a view classification problem, using a MobileNetV2 convolutional neural network to categorize video frames into global view, closeup view, and play-back view. The soccer event recognition task, instead, employs state-of-the-art action recognition algorithms, including I3D, I3D with non-local blocks (I3D-NL), ECO, and SlowFast networks, to classify video segments into events like goals, celebrations, card incidents, and passes.
Finally Valand et al. [2] explored deep learning approaches for automating soccer highlight clipping. Their system combines two computer vision tasks: logo detection using convolutional neural networks (CNNs) like ResNet and VGG, and scene boundary detection using the TransNet V2 model. The CNNs were trained on logo transition frames, while TransNet V2 applies convolutions and RGB histogram analysis to detect various scene changes.

Object detection: The roots of object detection in football lies in two papers by D'Orazio et al. In [16] a combination of the Circle Hough Transform (CHT) and a neural network classifier are employed to solve the task of ball detection. The CHT is utilized to identify circular patterns in the image and a neural classifier is applied to refine the detection.
In [17] a background subtraction algorithm and Maximum a Posteriori Probability were used for identifying and tracking players.
Yet, the methods presented in these papers have been largely overcome and as of August 2024, the most advanced attempt in combining object detection accuracy and low latency predictions is the project carried out by Piotr Skalski[5] from Roboflow and stored in the public GitHub repository "Sports"[6]. Skalski solves four challenges: ball tracking, reading jersey numbers, player tracking, player re-identification and camera calibration using several different models and Computer vision techniques. For what concerns object detection in "Sports" three different fine-tuned versions of YOLOv8 are used: one for detecting the players and the referees, one for the ball and a tuned version of a YOLOv8-pose that detects the pitch as the unique class and then identifies the pitch keypoints as a pose estimation task.

Finally, a lot of improvements in the general field of information extraction from football videos come from the yearly challenge organized by SoccerNet[7] in which they tackles three main themes and seven vision-based tasks distributed as follows ([18]):

[5] https://www.linkedin.com/in/skalskip92/
[6] https://github.com/roboflow/sports
[7] https://www.soccer-net.org/

- Broadcast Video Understanding:
  - Action Spotting
  - Ball Action Spotting
  - Dense Video Captioning
- Field Understanding:
  - Camera Calibration
- Player Understanding:
  - Re-identification
  - Multiple Object Tracking
  - Jersey Number Recognition

Coming back to the reason of this review, all of the works presented above and almost the totality of the research made on the extraction of information from soccer videos utilize high-quality broadcast images datasets (one of the very few non-broadcast examples, is [19]). In addition, in order for an automatic video tagging system to be deployed in production, not only it must be accurate but also pretty fast in detecting the elements needed to annotate an event which are the player, the event type, possibly its qualifiers and the position of the ball.
So what comes out from this analysis is that the task of ball localisation was perfectly aligned with the company's scope, unlike Clips extraction, and that it was allegedly manageable with success but needed further research in order to build a system able to generalize to lower-quality non-broadcast images. In fact, with the rise of faster Computer vision models like the ones of the YOLO (You Only Look Once) family, ball localisation is promisingly becoming an almost real-time task which can be performed rapidly and with very small computation power unlike the methods used for Specific events recognition.

1.2 The architecture

Following the approach described in the "Sports" repository, the task was split into three main blocks of a pipeline:

1. **Ball detection:** An object detection task specialized on one class.
2. **Pitch keypoints detection:** An object and keypoints detection task.
3. **Homography application:** A geometric transformation that exploits linear systems to map the points of an image to a predefined bidimensional plane (the standard football pitch).

Pitch keypoints are the intersections of the white lines of the pitch and they are detected in order to calculate the homography matrix, a 3×3 matrix computed by solving a linear system of at least four equations. This means that in order to obtain the homography matrix, four or more pitch keypoints need to be detected.
Once we have the matrix, the geometric transformation can be applied to map the pixels coordinates of the detected ball onto the 2D pitch representation that we call the radar, whose dimensions are predefined, and thus extract the ball coordinates.

1.3 Thesis structure

The thesis starts with a review of the SOTA (State of the Art) models for the object detection and the keypoints detection tasks (2); then describes in detail all the datasets that were used for all the experiments brought about, again dividing between datasets for ball detection and for pitch keypoints detection (3); in chapters 4 and 5 the Ball detector and the Pitch detector models are presented; chapter 6 shows how the homography is used in this context; the seventh chapter is about the pipeline implementation (7) and finally chapters 8 and 9 respectively show the results obtained by testing the pipeline in terms of speed and accuracy and suggest possible further improvements.

Chapter 2

State of the Art

2.1 Object Detection

The object detection models employed in the project are used to solve two different but interconnected tasks: image labeling and ball detection.
In order to develop a ball detector able to identify the sphere in non-broadcast images, a large number of annotated images is needed and, since the manual annotation is a highly time consuming task, an automatic annotator is a useful support for speeding up the labeling process, which, however, eventually needs to be checked manually.
A key technology for automatic labeling is *zero-shot* object detection (or *open-set* object detection). The SOTA zero-shot detectors are Grounding DINO and YOLO-World, the first for accuracy, the second for inference speed. Models of this kind are extremely useful because they are "promptable" and so they are able, without any further fine-tuning, to detect objects not present in their training set.

The ball detection task, instead, needs a model which produces almost real-time predictions and is accurate enough to find the ball in the several different scenes that can occur. The three best solutions available are then the two most advanced models of the YOLO family (YOLOv9 and YOLOv10) and RT-DETR which exploits a different architecture to obtain its promising results.

2.1.1 Grounding DINO

Grounding DINO [20] represents a significant advancement in open-set object detection, combining the DINO architecture with grounded pre-training. This model excels at detecting arbitrary objects specified by human language inputs, making it highly adaptable for various detection tasks, including soccer ball detection.
The key innovation of Grounding DINO lies in its tight fusion approach, which incorporates three feature fusion phases: a feature enhancer, language-guided query selection, and a cross-modality decoder. This deep fusion strategy effectively improves open-set object detection performance. The model also introduces a novel sub-sentence level text representation, which eliminates unwanted dependencies between category names while preserving fine-grained information.
Grounding DINO's flexibility in handling various types of language inputs, from simple category names to complex referring expressions, makes it particularly suitable for diverse detection scenarios. This adaptability could be especially valuable in football analysis, where the ball might need to be detected under varying conditions and descriptions.

The model's architecture consists of dual encoders for image and text, utilizing a Swin Transformer for image processing and BERT for text encoding. The feature enhancer and cross-modality decoder enable effective fusion of visual and textual information, contributing to its strong performance.

In terms of performance, Grounding DINO has demonstrated remarkable results in zero-shot transfer tasks. It achieved 52.5 AP on COCO minival without any COCO training data, and 63.0 AP on COCO test-dev after fine-tuning. Furthermore, it sets a new state-of-the-art on the ODinW zero-shot benchmark with 26.1 mean AP.

A unique aspect of Grounding DINO is its capability in Referring Expression Comprehension (REC) tasks. Unlike many open-set detectors, Grounding DINO extends its evaluation to REC tasks, showing strong performance in detecting objects with specific attributes. This capability could be particularly useful in a football scene where the ball needs to be identified in relation to players or field positions.

2.1.2 YOLO-World

YOLO-World [21] is an innovative approach that enhances the YOLO architecture with open-vocabulary detection capabilities through vision-language modeling and pre-training on large-scale datasets. It aims to address the limitation of traditional YOLO detectors, which are confined to predefined and trained object categories.
The key innovation of YOLO-World lies in its Re-parameterizable Vision-Language Path Aggregation Network (RepVL-PAN) and region-text contrastive loss. These components facilitate the interaction between visual and linguistic information, enabling the model to detect a wide range of objects in a zero-shot fashion with efficiency.
YOLO-World demonstrates promising zero-shot transfer capabilities, performing well on novel categories without specific fine-tuning.
The authors also reports remarkable performance on challenging datasets. On the LVIS dataset, it achieves 35.4 AP with 52.0 FPS gaining an edge in efficiency which is crucial for real-time applications like ball detection.

2.1.3 YOLOv9

YOLOv9 continues the evolution of the YOLO family, known for its real-time object detection capabilities. It introduces two key innovations: Programmable Gradient Information (PGI) and Generalized Efficient Layer Aggregation Network (GELAN). PGI allows the model to learn what information to learn, improving its ability to capture relevant features for detection. GELAN, a new lightweight network architecture designed based on gradient path planning, offers improved efficiency and accuracy trade-offs.
These advancements enable YOLOv9 to achieve significant improvements over its predecessors while maintaining the real-time performance that YOLO models are known for. This balance of speed and accuracy makes YOLOv9 a strong candidate for applications like ball tracking in football videos, where real-time processing is crucial.

2.1.4 RT-DETR (Real-Time DEtection TRansformer)

RT-DETR [22] stands out as the first real-time end-to-end object detector based on the DETR architecture. It addresses the computational bottlenecks of previous DETR models while preserving

their benefits. The model introduces an efficient hybrid encoder designed to process multi-scale features quickly by decoupling intra-scale interaction and cross-scale fusion. Additionally, it employs uncertainty-minimal query selection to provide high-quality initial queries to the decoder, thereby improving accuracy.

A notable feature of RT-DETR is its support for flexible speed tuning, allowing adjustment of the number of decoder layers to adapt to various real-time scenarios without retraining.

RT-DETR has demonstrated impressive results, outperforming many YOLO variants (but not YOLOv9 and YOLOv10) in both speed and accuracy on the COCO dataset. Its end-to-end design eliminates the need for post-processing steps like Non-Maximum Suppression (NMS), potentially offering more consistent performance in varying detection scenarios, such as different camera angles or lighting conditions in football matches.

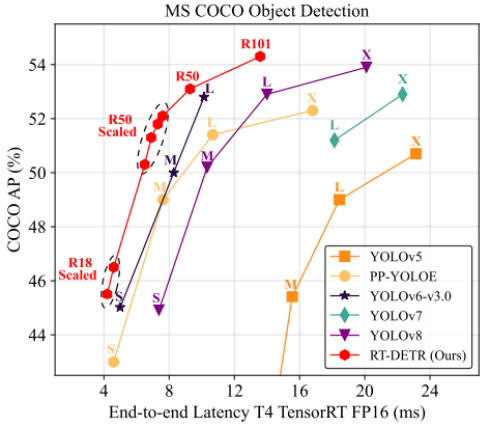

Figure 3: Image take from: [22]. RT-DETR model comparison about accuracy and speed (YOLOv9 and YOLOv10 were published after this work).

2.1.5 YOLOv10

YOLOv10 represents the latest advancement in the YOLO family. It introduces several innovative approaches to enhance both speed and accuracy, making it a powerful tool for tasks like soccer ball detection.

A key feature of YOLOv10 is its consistent dual assignments for NMS-free training. This novel approach eliminates the need for Non-Maximum Suppression in post-processing by combining one-to-many and one-to-one assignments during training. The one-to-many branch provides rich supervision, while the one-to-one branch enables efficient, NMS-free inference. This design not only improves inference speed but also enhances the model's adaptability to different detection scenarios, which is crucial in the dynamic environment of a football match.

YOLOv10 takes a holistic approach to model design, optimizing various components for both efficiency and accuracy. It introduces a lightweight classification head to reduce computational overhead without significantly impacting performance. The spatial-channel decoupled downsampling improves efficiency in feature processing, while the rank-guided block design adaptively integrates compact

block designs for improved efficiency. To enhance model capability with minimal computational cost increase, YOLOv10 incorporates large-kernel convolution and partial self-attention mechanisms. The NMS-free design could lead to more consistent detections, especially in situations where the ball might be partially occluded or in close proximity to players.

YOLOv10 supports various model scales, from YOLOv10 - N to YOLOv10 - X, allowing users to choose the best trade-off between speed and accuracy for their specific application.

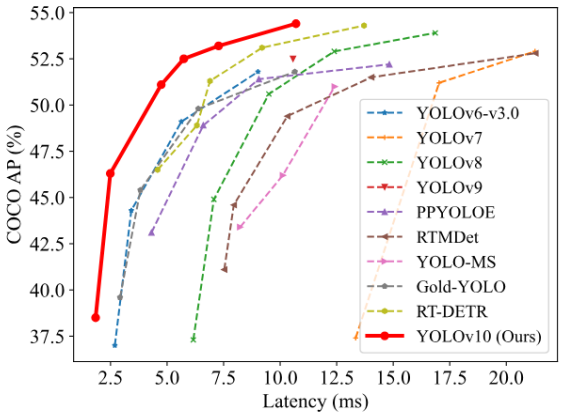

Figure 4: Comparisons between YOLOv10 and other models in terms of latency-accuracy

The model's efficiency and scalability allow for deployment on a range of hardware, from high-performance GPUs to more constrained compute environments. This versatility could be valuable in different football analysis scenarios, from broadcast enhancement to post-match tagging on CPU.

2.2 Keypoints detection

For what concerns the detection of the pitch keypoints for the calculation of the homography matrix, two very different approaches have been evaluated: camera calibration, that is the process of determining the internal parameters of a camera, such as focal length, principal point, and distortion coefficients, as well as its position and orientation relative to a world coordinate system [23] with TVCalib and a Pose estimation solution with YOLOv8-pose.

2.2.1 TVCalib

TVCalib, while not primarily a keypoint detection model, is a camera calibration framework designed specifically for sports field registration in broadcast videos. It addresses the challenge of mapping 2D broadcast images to a 3D sports field model, which is crucial for accurate ball position extraction in football analysis.

The key innovation of TVCalib is its keypoint-less calibration approach. Unlike traditional methods that rely on specific keypoints, TVCalib uses line segments and other field markings for calibration. This makes it more robust to partial field views, a common scenario in broadcast footage.

TVCalib employs a novel differentiable objective function that measures the reprojection error of field segments, allowing for end-to-end optimization. The model uses gradient-based iterative optimization to refine camera parameters, leading to accurate field registration.

While TVCalib does not directly detect keypoints, its ability to accurately map the broadcast view to a predefined field model by providing a stable reference frame, helps in translating pixel coordinates of detected points to real-world field positions.

2.2.2 YOLOv8-pose

YOLOv8-pose [24] extends the YOLOv8 object detection framework [25] to pose estimation and hence keypoint detection tasks. While focused on human body keypoints detection for pose estimation this model can be fine-tuned to detect any object with its fundamental points such as the keypoints of a football pitch.

The model integrates object detection and keypoint localisation into a single network, allowing for efficient and accurate pose estimation. Leveraging the speed optimizations of the YOLO architecture, YOLOv8-pose can perform keypoint detection with little latency.
Architecturally, YOLOv8-pose builds upon the YOLOv8 design, using a CSPDarknet backbone, spatial pyramid pooling, path aggregation neck, and multi-scale prediction heads. For pose estimation, it adds branches to predict keypoint locations and their associated confidences. The training process combines object detection loss for bounding box prediction with a keypoint localisation specific loss function.

In terms of performance, YOLOv8-pose has shown strong results on standard pose estimation benchmarks. On the COCO keypoints dataset, it achieves competitive mean Average Precision (AP) scores while maintaining real-time inference speeds. The model demonstrates good performance across various scales, from the smallest (n) to the largest (x) versions. The n version achieves a mAP 50-95 of 50.4 with a CPU inference time of 131.8 ms, while the YOLOv8x-pose reaches a higher mAP 50-95 of 69.2 but with a significantly longer CPU inference time of more than one and half second.

Chapter 3

Dataset

In this section all the different datasets used for training and testing purposes in the project development are described and their sources presented starting from the datasets used for the ball detection task, following with pitch detection and ending with the data used for evaluating the precision of the system in locating the ball.

3.1 Ball detection

The two datasets for ball detection presented below come from different sources and contain different information but share the annotation type. In fact since this is an object detection task we need the coordinates of bounding boxes around the object of interest, i.e. the ball.

3.1.1 Roboflow - Football ball detection

The first dataset used for experimenting with ball detection is one of the open-source image sets available in the Roboflow Universe platform[1]. Its title is "*football-ball-detection Computer Vision Project*" [26] and the version 1 is made of 903 high-quality, broadcast frames of football matches annotated with a bounding box around the ball. In the context of this work, it was employed as a fast and ready-to-use dataset at the beginning of the project to test the ability of a YOLOv9 model fine-tuned on those images to generalize to non-broadcast pictures.

3.1.2 Soccerment's dataset

The most extensive dataset used is Soccerment's proprietary images collection extracted from 21 videos of internally tagged games.
The footage is of four quality types dubbed with the letters A, B, C, D from the best to the worst. The frames come from recordings operated with high-quality broadcast cameras, tactical cameras that almost cover the full pitch and a portable camera positioned close to the benches or in the stands of non-professional leagues stadiums (Pixellot Air[2]). Quality was assessed with the assistance of two football analysts experienced in tagging operations that appraised the resolution, the camera position and possibly any disturb to the footage (for example colored smoke from the stands altering the sight of the portable camera).

[1] https://universe.roboflow.com/
[2] https://www.pixellot.tv/products/pixellot-air/

Since the videos are also tagged, it is possible to match the information about tagged event in the corresponding timestamp with the exact frame recording the action.

Frames extraction In order to train an efficient system to extract the ball location we need to work with single frames and not to process entire video sequences. Hence, from the 21 videos, 45'880 frames have been extracted by taking an image anytime an event is tagged.
For every event, in addition to the image, the quality of the frame and, most importantly, the type of event recorded are extracted. Indeed, the event type detail is of fundamental importance for the automatic labeling of the ball as we will see in section 4.1.
After the above-mentioned automatic labeling, the images are attached with the coordinates of the bounding box around the ball when present.

Finally, after removing the images without ball annotations, the remaining 34'043 frames are split in training (30'660), validation (1'732) and test (1'651) set balancing each subset across the four quality types.

Figure 5: Four examples of Soccerment's ball detection dataset.

3.1.3 Test dataset

From the complete Soccerment's ball detection dataset a small set of 100 images, uniformly distributed across the four quality types, has been extracted in order to perform light and fast tests.

In particular this dataset is used for comparing the performance of the automatic labeling models explored in section 4.1.1.

3.2 Pitch detection

While the datasets used for ball detection share the annotation type, describing datasets used for pitch detection we can observe three different types of annotations.

3.2.1 Dataset WC14 - TVCalib

One of the datasets utilized in the paper presenting TVCalib ([27]), it is a collection of 395 annotated images pulled out of 20 games of the World Cup 2014 published in [28]. The annotations in this dataset are ground truth fields and also grass segmentations.

3.2.2 Roboflow - Football pitch detection

Inspired by the work of Piotr Skalski cited in the Introduction (footnote 5 in 3.3), the starting point for pitch detection as a pose estimation task is represented by this open-source dataset [29]. Version 11, which is the one used in this project, contains 276 high-quality, broadcast frames. The annotations in this dataset are the pitch, bounded by a box, as the only object in the image and its keypoints which are considered the visible pitch line intersections, plus the penalty spots and the vertices of the midfield circle.

3.2.3 Roboflow - Campo futebol keypoint classes v15

This dataset [30] is another of the precious open-source projects of Roboflow Universe and in its version 15 contains 808 broadcast images of heterogeneous quality divided into 808 train, 63 validation and 33 test frames.

At the date of its use it was the most extensive open-source dataset for pitch keypoints detection while, today, following versions have added new annotated images.

Here labels still refer to bounding boxes and keypoints. However, each intersection of the white lines represents an object. This object is enclosed by a rectangular bounding box that encompasses some area around the intersection point and the exact spot where the lines cross is the object's unique keypoint. Thus, while with the annotation approach of 3.2.2 there is always one big bounding box (the field) and as many keypoints as the pitch keypoints in the image, now we have several objects each with a single keypoint. The challenge is then shifted from detecting the pitch and then identifying the keypoints to performing a good object detection by identifying the correct bounding boxes in the correct position, each keypoint will then fall inside its box.

The keypoints annotated in this dataset follow the scheme represented in figure 6 dividing the pitch in seven vertical lines and going from up to down, from left to right. The keypoints of the penalty box the annotator sees in the left part of the screen are defined as defensive keypoints (D1, ..., D12), the ones in the central line are midfield keypoints (M1, ..., M4) and finally the ones at the right are offensive keypoints (O1, ..., O12).

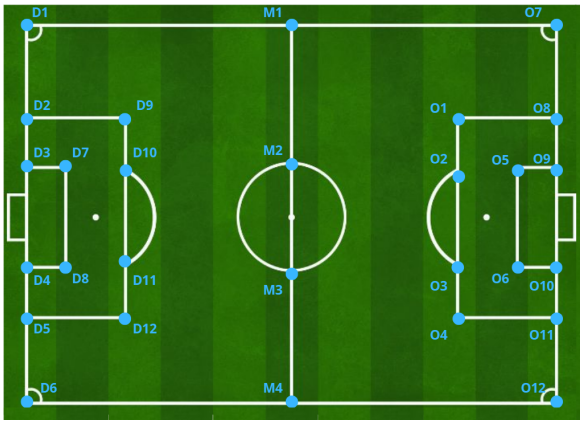

Figure 6: Pitch keypoints map in *"Campo futebol keypoints classes v15"*

Figure 7: An example of annotated image from *"Campo futebol keypoints classes v15"*

Augmentation For the sake of generalization to non-broadcast images this dataset was augmented by adding 350 manually annotated frames from Soccerment's dataset (3.1.3). The process is described in 5.2.

3.3 Ball localisation

The last section of the chapter presenting the datasets is dedicated to the images used to measure the precision of the pipeline in localising the ball in the pitch. Hence, these datasets are used for testing purposes only.

3.3.1 Soccerment's tagging data

By exploiting again the match between frames and event record as for building Soccerment's ball detection dataset in 3.1.3 the tagged position of the ball during that event can be extracted and used to measure how far is the ball with respect to location output of the pipeline.
Yet, an impeding characteristic of these images is that they can be tagged some fractions of seconds or even seconds after the event. Of course this implies that the tagged ball location represents where the ball was some frames before and not where it is in the current image.
For this reason a new, smaller but accurate dataset was manually tagged.

3.3.2 Refined Soccerment's tagging data

This is a collection of 1'000 images in which only the ball position was tagged by paying attention that the ball is located in the most accurate spot with respect to the frame visualized.
Images are sampled from different games to have heterogeneity of the light conditions, pitch color, jerseys color, etc. but they are evenly divided among quality A, B, C and D.

Chapter 4

Ball detection

In this chapter two different applications of object detection, both tailored to ball detection, are presented. The first is ball detection for performing automatic image labeling while the second is the actual ball detection that empowers the ball localisation pipeline.

4.1 Ball detection for automatic labeling

Object detectors who have been trained using the COCO dataset contain the class `sports ball` and so they could be able to identify a soccer ball.
Hence, a low-cost experiment to start building the ball detector consists of using a pre-trained version of the state-of-the-art object detection models trained on the COCO dataset such as YOLOv9 and YOLOv10. Yet, those model did not yield appreciable results when applied to Soccerment's dataset of non-broadcast images because images of the COCO dataset that contain a sports ball are frames in which the ball represents a large fraction of the display contrary to what happens in football games recordings where the ball is usually very small and seen by a higher perspective. The bad performance of pre-trained models highlighted the necessity for a fine-tuned model that could be able to generalize to very different recordings situation.

In order to train effectively an object detector for a new class not present in the training set, a large number of images is required - typically thousands. This is especially true when each image contains only one instance of the class, as is the case with the single ball in play during a game. Ultralytics[1], a reputable source in the field, provides guidance on the dataset size needed to train a model (at least a YOLO model) for substantial results. They recommend:

- At least 1'500 images per class (we have only one class: the soccer ball).
- A minimum of 10'000 labeled instances of the object per class.

As described in 3.1.3, Soccerment's dataset comprises 45'880 frames that, once labeled, can be more than enough to respect previously mentioned suggestions. Of course labeling this amount of images can take several days or even weeks of work and thus an automatic labeling method was used to speed up the activity.

[1] https://docs.ultralytics.com/yolov5/tutorials/tips_for_best_training_results/

Since real-time is not necessary during the labeling process and, as said, light pre-trained models do not offer appreciable results, in order to annotate this amount of images, a heavier, zero-shot detection model was employed to find the ball in the highest possible number of non-broadcast football game frames in the dataset.

In the next two subsections the alternative models explored are compared by testing their accuracy on the test set described in 3.1.3.

4.1.1 Model selection

Three different models have been tested in order to choose the most promising one based on the performance achieved on the test dataset of 100 images.
These are:

- Grounding DINO (see 2.1.1)
- YOLO-World (see 2.1.2)
- YOLOv9 (see 2.1.3)

The first architecture, **Grounding DINO**, is a zero-shot object detector that combines visual information from an image's pixels with a textual description of the target object that can go beyond a mere naming and be a detailed expression featuring explicit visual characteristics of the target such as color, dimension and relative position with respect to other objects in the picture. In this context Grounding DINO was evaluated using a base prompt which then was enriched with more details about the ball first specifying that it must be found inside the pitch lines and then adding the on-ball event occurring in the frame (dynamic prompt) as it follows.

Base prompt: *soccer ball that can be white or yellow, sliding on the grass or close to the feet, the head or the hands of a player.*

Prompt with "inside lines": *soccer ball that can be white or yellow, sliding on the grass or close to the feet, the head or the hands of a player inside the pitch lines.*

Dynamic prompt: *soccer ball that can be white or yellow, sliding on the grass or close to the feet, the head or the hands of a player in the action of a* {*event_type*} (e.g. *shot*).

Dynamic prompt with "inside lines": *soccer ball that can be white or yellow, sliding on the grass or close to the feet, the head or the hands of a player inside the pitch lines in the action of a* {*event_type*}.

The model then outputs 900 bounding boxes each with similarity scores across all input words. It selects the boxes whose highest similarities are higher than a `box_threshold` fixed to 0.15 and extracts the words whose similarities are higher than a `text_threshold` equal to 0.25 as predicted labels.

The second model is **YOLO-World** which is another zero-shot object detector elaborated on the YOLO architecture. As Grounding DINO, it takes as input an image and a textual prompt and outputs bounding boxes for the described objects.
It is very lightweight compared to Grounding DINO and thus is way faster during inference. Yet, even though it trades off latency and accuracy with surprising results, we will see in the next section

that it is still far from yielding satisfying results.
YOLO-World was tested using two prompts: the base prompt and the extended version that includes the phrase "inside the pitch lines".

The last model experimented is **YOLOv9** which was the state-of-the-art object detector as the date of the test (March 2024) for its latency-accuracy tradeoff. It does not take any prompt and only takes the image as input.
YOLOv9 was tested in three versions: the pre-trained version released in the official implementation[2] and two quickly fine-tuned versions, one trained on the dataset published in Roboflow (see 3.1.1) and the other trained on the 100 frames of the Test dataset described in 3.1.3.

4.1.2 Automatic labeling test results

Here the results of nine tests - four with Grounding DINO, three with YOLOv9 and two with YOLO-World - are presented using vertical bar charts that show the times a model found the correct ball, found multiple balls including the correct target, did not find the ball in play. All the trials were made on the Test dataset and then assessed by human eye.

[2] https://github.com/WongKinYiu/yolov9

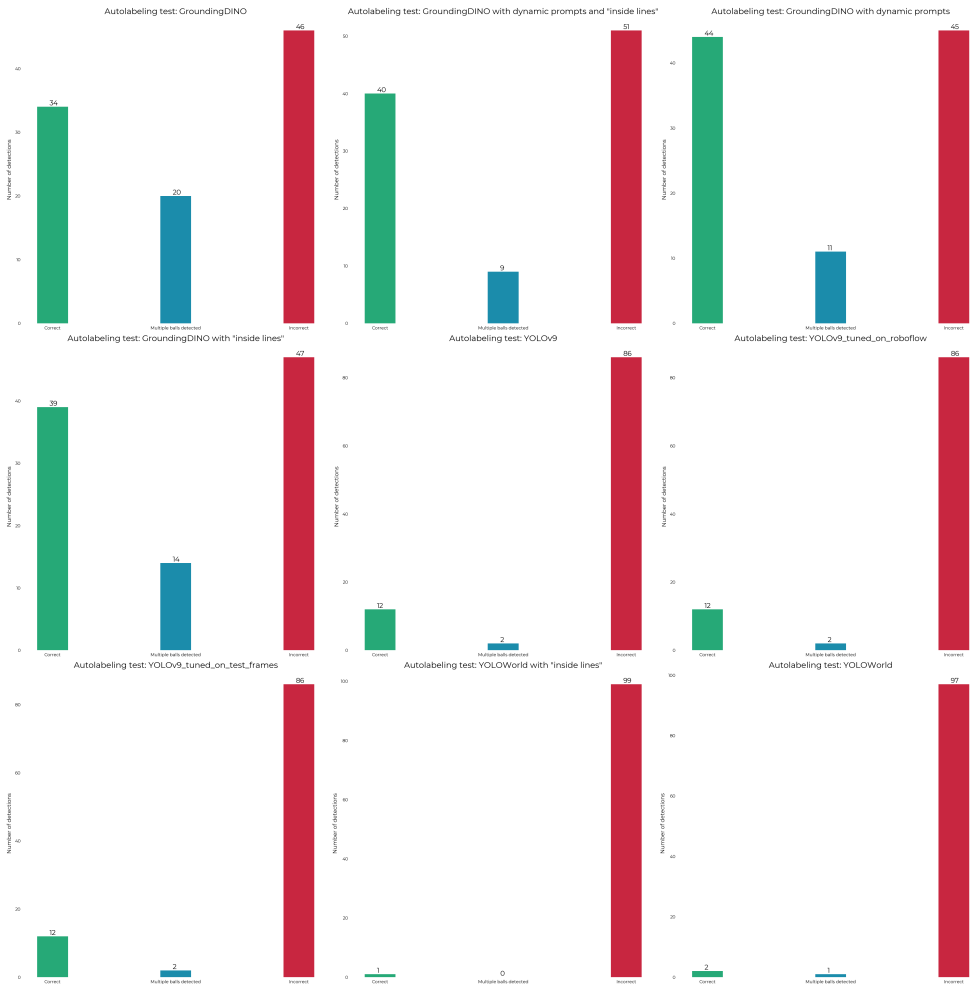

Figure 8: Test results for automatic labeling model selection. Correct detections in green, multiple ball detections in blue and incorrect detections in red.

Figure 8 shows that the most promising model is Grounding DINO with the use of the dynamic prompt which includes the event type occurring in the frame. Indeed, with this configuration the model was able to detect the correct ball 55 times out of 100 even though 11 times of those 55 other targets were identified.
Neither YOLOv9 nor YOLO-World managed to achieve similar performance. Indeed, the four attempts with Grounding DINO are also the best four.

4.1.3 The labeling process

Once identified the most promising model and the best prompting strategy, the 45'880 images of the Soccerment's dataset were passed as input to Grounding DINO, together with the prompt each time updated with the event name captured in the frame. In particular, in order to change dynamically the event in the prompt, each frame was saved using a format that includes `game_id`, `quality`, `client_team_name`, `frame_id` and finally the `event_type` (e.g. `99997_A_italy_frame_26_Pass.png`).

For each frame the zero-shot detector returned the predicted coordinates of the bounding boxes that identify the ball and these coordinates were saved in a format compatible with an annotator software - that will be introduced in the next paragraph - in order to perform manual refinement of those automatically annotated images.
The entire labeling job, ran on four Tesla T4 GPUs and took around 13 hours with an average inference time of about 1s/frame.

Even if Grounding DINO was particularly successful in identifying the ball we could not expect a much higher accuracy than the figures reported in 4.1.27. So a manual check and possible refinement was needed.
The open-source software VGG Image Annotator [31, 32] was used to perform a visual check on the labeled images and in order to fix the incorrect predictions. Indeed, by using this light software accessible offline through a simple HTML file, all the frames containing a misdetection or the ones in which Grounding DINO did not identify the ball can be manually corrected. In particular, when the automatic labeler identified wrong objects as the target ball, these unnecessary boxes are selected and removed by pressing the backspace key; when the ball in play is not detected a bounding box is drawn around it by dragging its vertices with a mouse and when, instead, the target ball is the only object identified the frame is already correctly labeled and thus no manual correction is needed.
A useful observation is that there is no necessity to explicitly specify the label name (`soccer_ball`) in each frame as it is the only label in the dataset and thus every annotation is automatically matched to it.
Finally, given the different situations a frame can be found in after the automatic labeling - i.e. with multiple wrong objects identified, with the correct ball or with no detections at all - the time needed to perform this manual check is usually between 1s and 10s per frame.

Among all the 45'880 checked and corrected images, 11'837 were background images i.e. frames without any target object that translated in this context means without the ball in play. The other 34'043 pictures were split in the training, validation and test folders as described in 3.1.3.

4.2 Ball detector

The ball detector is the first block of the pipeline architecture. The duty of this component is to identify the ball in a frame and to return its pixel coordinates.
This is done by using an object detection model that outputs a bounding box around the ball and then taking the center of the rectangle as the ball coordinates.
This section covers the building of the ball detector from model selection to training.

4.2.1 Model selection

The reported performance of the SOTA object detectors (see Chapter 2) have been used to narrow down the decision on the best object detector for this use case to two different models: RT-DETR (see 2.1.4), which is the best real-time detection transformer and YOLOv10 (see 2.1.5) which is the fastest and most accurate model based on the YOLO architecture.
Additionally, we must keep in mind that, in addition to accuracy, the final application of the pipeline should be capable of running on a CPU with little inference time. This is to ensure that a tagging operator can use it on a standard laptop without loosing time.

Regarding performance, the evaluations in the respective papers demonstrate that YOLOv10 beats RT-DETR both for accuracy and latency placing the first on a more efficient frontier as shown in Figure 4.
For what concerns CPU computing, the computation complexity was estimated comparing the number of FLOPs (FLOating Points operations) needed for a single forward pass of the two models and they confirm the superiority of YOLOv10. The biggest version, YOLOv10 - X, needs 160.4G FLOPs to perform inference on one image while the heaviest version of the second model, RT-DETR-R101, needs 259G FLOPs. And more FLOPs means higher latency in inference.
Hence, YOLOv10 was chosen as the most promising model to be fine-tuned on Soccerment's dataset for ball detection.
In the next subsection the training strategy and its results are discussed.

4.2.2 Training

A practical way to train a YOLO model is to exploit the Ultralytics Python library which comes with a SDK to train and run inference using different models of the YOLO family, including YOLOv10. Its implementation requires a specific data structure consisting of three main folders: `train`, `valid`, and `test` and a `.yaml` configuration file that specifies the paths to these folders and the class names. Each main folder must contain two subfolders: `images` (containing frames in `.png` or `.jpg` format) and `labels` (containing corresponding `.txt` files). The label files follow a specific format, with one object per line. Each line contains the `class_id` and bounding box coordinates all separated by a white space. The coordinates of the bounding boxes follow the rectangle representation obtained by taking the coordinates of its center and then expanding horizontally and vertically using the width and height values. Thus, each row looks like: `[class_id center_x center_y width height]`.
An example of the dataset structure can be then visualized as it follows:

```
YOLO_dataset/
|- train/
|  |- images/
|  |  |- image1.png
|  |  |- ...
|  |- labels/
|     |- image1.txt
|     |- ...
|- valid/
|  |- images/
|  |  |- image2.png
|  |  |- ...
|  |- labels/
|     |- image2.txt
|     |- ...
|- test/
|  |- images/
|  |  |- image3.png
|  |  |- ...
|  |- labels/
|     |- image3.txt
|     |- ...
|- config.yaml
```

After having organized the dataset as requested, two training jobs were needed in order to store a model instance that could work within the pipeline environment. This because the Ultralytics version which the first ball detector was trained with (8.1.34) was different than the version of the pitch detector (8.2.71) and this caused a dependency conflict. Indeed, if the pipeline needs to operate entirely in the same cpu-powered environment (i.e. without exploiting a microservices architecture) it cannot have dependencies issues.
Yet, this created the opportunity to fine-tune a smaller and faster ball detector. The first model, in fact, was a YOLOv10 - X instance, the heaviest version, while in the second job a YOLOv10 - M instance was trained and it achieved similar results being faster during inference. Below the training metrics of both the jobs are analyzed.

Both the models were trained for 100 epochs using four Tesla T4 GPUs and with a batch size of 48 images. In Figure 9 the training metrics of the YOLOv10 - X instance are reported. The plots offer a visual of the behavior over the training epochs of six loss functions and four metrics.
In particular, these losses are:

- Objectness Matching Box Loss (box_om) measures the error in the predicted bounding box coordinates for object matches. It typically uses a combination of IoU and GIoU (Generalized IoU) losses to ensure better alignment with the ground truth.

$$\text{box\_om} = \lambda_{\text{box}} \sum_{i=0}^{N} \mathbb{I}_i^{\text{obj}} \left[\text{IoU}(B_i, \hat{B}_i) + \text{GIoU}(B_i, \hat{B}_i) \right]$$

  Here, λ_{box} is a scaling factor, N is the number of bounding boxes, and $\mathbb{I}_i^{\text{obj}}$ indicates if an object is present.

- Objectness Overlap Box Loss (box_oo) focuses on the overlap between predicted and ground truth boxes. It uses IoU and DIoU (Distance IoU) for better spatial alignment.

$$\text{box\_oo} = \lambda_{\text{box}} \sum_{i=0}^{N} \mathbb{I}_i^{\text{obj}} \left[\text{IoU}(B_i, \hat{B}_i) + \text{DIoU}(B_i, \hat{B}_i) \right]$$

- Objectness Matching Class Loss (cls_om) measures the error in the predicted class probabilities for object matches. It uses Focal Loss to handle class imbalance.

$$\text{cls\_om} = \sum_{i=0}^{N} \mathbb{I}_i^{\text{obj}} \sum_{c \in \text{classes}} [\text{FL}(p_i(c), \hat{p}_i(c))]$$

  In this formula, FL denotes Focal Loss.

- Objectness Overlap Class Loss (cls_oo) is similar to cls_om but focuses on overlapping objects. It ensures better classification accuracy in dense scenes.

$$\text{cls\_oo} = \sum_{i=0}^{N} \mathbb{I}_i^{\text{obj}} \sum_{c \in \text{classes}} [\text{FL}(p_i(c), \hat{p}_i(c))]$$

- Objectness Matching Distribution Focal Loss (dfl_om) is used for fine-grained classification. It

focuses on hard-to-classify examples.

$$\text{dfl\_om} = -\alpha_t(1 - p_t)^\gamma \log(p_t)$$

- Objectness Overlap Distribution Focal Loss (dfl_oo) is similar to dfl_om but is applied to overlapping objects.

$$\text{dfl\_oo} = -\alpha_t(1 - p_t)^\gamma \log(p_t)$$

The four reported metrics are:

- Recall is the ratio of correctly detected objects to the total number of actual objects in the image.

- mAP50 represents the Mean Average Precision calculated at an Intersection over Union (IoU) threshold of 50%.

- mAP50-95 is the Mean Average Precision calculated across multiple IoU thresholds, ranging from 50% to 95% in 5% increments.

- Precision is the ratio of correctly detected objects to the total number of objects the model predicted.

The consistent decrease of all the loss functions and the slightly increasing stabilization of the metrics indicate that the training converges and that, even though with a smaller impact, the model becomes better at ball detection epoch after epoch.
Loss functions values are calculated at every epoch on the training set while the metrics are calculated by performing a batch of predictions on the images of the validation set in order to validate the model during training on unseen frames.

The standard metrics used for evaluating the performance of the fine-tuned model are mAP50 and mAP50-95 calculated on the test set in order to assess the model's ability of detecting the ball on completely unseen frames. Figure 10 shows that the heaviest model achieved respectively a mAP50 of 0.863 and a mAP50-95 of 0.588.

Finally, other than detection performance we are interested also in the expected inference time in CPU which for YOLOv10 - X is around 750ms per detection.

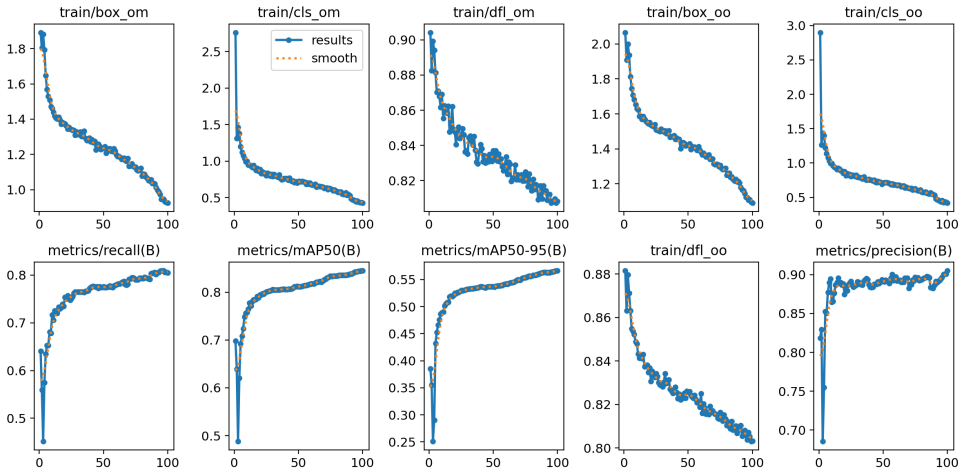

Figure 9: Training metrics of YOLOv10 - X

metrics/precision(B)	metrics/recall(B)	**metrics/mAP50(B)**	metrics/mAP50-95(B)	fitness
0.912	0.818	0.863	0.588	0.615

Figure 10: Test results of YOLOv10 - X

The lighter model YOLOv10 - M, instead, achieved quite impressive training and testing metrics with respect to the previous one. Indeed, training losses decrease sharply after some epochs and then stabilize their curve decreasing with lower pace showing training convergence. Version 8.2.71 of Ultralytics returns also the validation loss functions, as it can be spotted in the left-bottom subplots of Figure 12, which are also sharper than the training losses.
Validation metrics are similar to the ones observed above but what emerges is that the test mAP50 and mAP50-95 are slightly higher than the ones obtained with the X version of the model as they stand respectively to 0.868 and 0.599 thus confirming that the performance of the two models are comparable.
Finally, since the performance of the lighter M version of YOLOv10 obtained similar metrics with respect to the YOLOv10 - X instance, the focus moves to expected inference time in order to find a computing advantage. On a 11th Gen Intel® Core™ i7-1165G7 CPU, this model employs $365\,\text{ms} \pm 17.1\,\text{ms}$ for a detection, almost half of the latency of the heaviest model, which sets this lighter version out as the most efficient without losing in precision of the predictions.

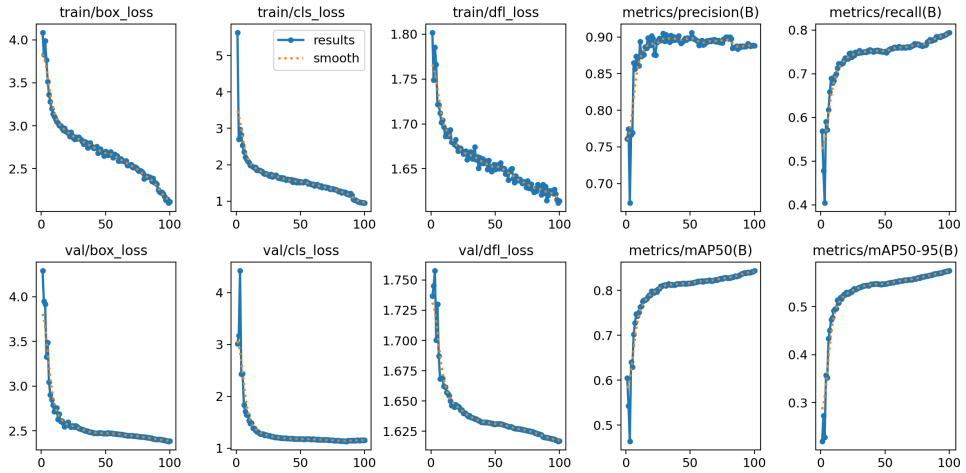

Figure 11: Training metrics of YOLOv10 - M

metrics/precision(B)	metrics/recall(B)	**metrics/mAP50(B)**	metrics/mAP50-95(B)	fitness
0.913	0.812	**0.868**	0.599	0.626

Figure 12: Test results of YOLOv10 - M

Chapter 5

Pitch detection

The pitch detection task represents the second block of the pipeline. Its target is to obtain the coordinates of some of the keypoints in the pitch in order to estimate its position and orientation in the image. The keypoints are then sent as input to the homography module for the matrix calculation.
This task was faced with two different approaches explained below.

5.1 Model selection

Two different solutions, and respectively two different models, have been explored for building the pitch detector module. The first is TVCalib which turns the pitch detection task into a camera calibration problem. Its novel iterative method to reduce the segment reprojection error after segment localisation (see 2.2.1) allow the model to have remarkable performances on partial field view images, which is the situation observed in almost every frame.
TVCalib is able to estimate the entire pitch by finding the correct camera parameters. It exploits the pitch lines segments instead of just the keypoints and its implementation also allows for the application of homography in one step.
Yet, this model has two heavy technical drawbacks: it is very slow on CPU and the annotations it employs are segments and not bounding boxes and keypoints.
Additionally, TVCalib was trained exclusively on broadcast images from the SoccerNet dataset that was published in [33].

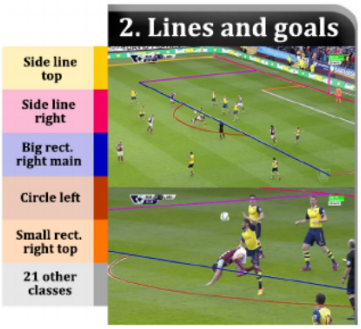

Figure 13: An example of the segments annotations taken from Fig. 1 of [33].

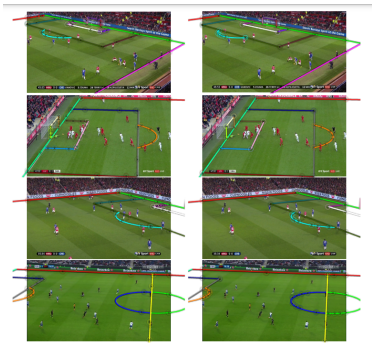

Figure 14: TVCalib predictions on test samples.

If we wanted to train it on non-broadcast images, we would need to label the frames in the same manner as in the aforementioned dataset. This would involve annotating all pitch segments, including the curved ones. Consequently, the labeling process would become more cumbersome and more time-consuming than simply annotating the pitch anchor points with bounding boxes and keypoints. Finally, TVCalib is expected to take around 23s for predicting the pitch in one image (tested on the same processor seen in 4.2.2) making it unsuitable for manual tagging in CPU computing.

Given the impairing limitations of TVCalib, the second model experimented was YOLOv8-pose which is the YOLOv8 branch devoted to pose-estimation and, as a model of the YOLO family, it is characterized by a well optimized accuracy-latency tradeoff.
The pre-trained version of YOLOv8-pose is conceived for human body detection and pose estimation. Yet, as we saw for the ball detector, this model can be fine-tuned to make it applicable to a completely different domain.
YOLOv8-pose processes a label as a bounding box and its keypoints, e.g. in a picture with three persons it would detect three bodies as objects and the respective body joints as the keypoints of each object. In our context, the objects are, instead, the intersections of the white lines of the pitch each of which contains one keypoint positioned exactly in the intersection spot.
This way manually annotating one image becomes a more manageable task even if still highly time-consuming.
Additionally, YOLOv8-pose in its largest version - `YOLOv8x-pose-p6` - is able to run inference on one image in a bit more than 4s and its second heaviest model - `YOLOv8x-pose` - in just 1.6s using an Amazon EC2 P4d[1] instance as reported by the authors[2].

For these reasons YOLOv8-pose was chosen as the best model candidate to face the pitch detection task. In particular, due to the restricted amount of images in the training set but still with the need to minimize inference time on CPU, the second heaviest version of the pose detection model, YOLOv8x-pose, was chosen in order to have acceptable performance and latency.

[1] Inference time on the same CPU used to test TVCalib are reported later.
[2] https://github.com/ultralytics/ultralytics/issues/1915

5.2 Dataset augmentation

Image labeling for the pitch detection task is, as already hinted at in the previous section, a time-intensive task and that is why an already annotated open-source dataset (see 3.2.3) was chosen for training the pitch detector. Yet, this collection does not contain non-broadcast images and thus the instance of YOLOv8-pose trained on those 808 images could not offer satisfying results on Soccerment's non-broadcast frames.
The dataset was then expanded including 350 images from Soccerment's footage manually annotated using the VGG Image Annotator software. The annotation took around two minutes per frame for a net total of almost twelve working hours of manual labeling. Indeed, manually labeling pitch keypoints is more complex than annotating the ball. The first reason for this increased complexity is that pitch keypoint labeling requires multiple bounding boxes, each of which must be named. Additionally, the process involves not only drawing these named boxes but also precisely marking a keypoint within each box to indicate the exact spot where a line-crossing occurs. These factors combined make the pitch keypoint labeling process significantly more intricate than the simpler task of ball annotation where, as seen in 4.1.3, for each frame we need to draw only one box and there is no need to name the bounding box since there is a unique class `soccer_ball`.
After the addition, the size of the dataset splits became 1'122 frames for the training set, 89 for the validation set and 43 images for the test set.

5.3 Training

The dataset structure for fine-tuning YOLOv8-pose is the same as what we saw in 4.2.2 but with a difference. Each `labels.txt` file in the augmented dataset now contains two more elements for every keypoint in the bounding box which are the keypoint's x and y coordinates. A row then becomes:
`[class_id center_x center_y width height keypoint_x keypoint_y]`
containing information about the bounding box around the line-crossing spot and its unique keypoint.

The pre-trained instance of YOLOv8x-pose was re-trained on the above-mentioned dataset using the Ultralytics SDK for 200 epochs on a Tesla T4 GPU for about 5 hours. The job's loss functions and metrics are reported in the subplots of Figure 15. They are the training and validation loss functions we saw for the training of the ball detector in 4.2.2 with two additions: the keypoints objectness loss function and the pose loss function which are, respectively, the Binary Cross-Entropy with logit loss which balances the confidence of keypoint predictions and the overall pose loss as a weighted sum of the Binary Cross-Entropy logit loss and the object area ensuring accurate placement of keypoints.
Those cost functions and metrics highlight training convergence with decreasing losses but also wide deviations from epoch to epoch when it comes to the loss function and the precision, recall and mean average precision metrics regarding the keypoints localisation. This is mainly due to the relatively small size of the training set which causes the YOLOv8 network to learn sub-optimal parameters because it cannot observe enough samples of each object class and which is a major issue to face in the near future.

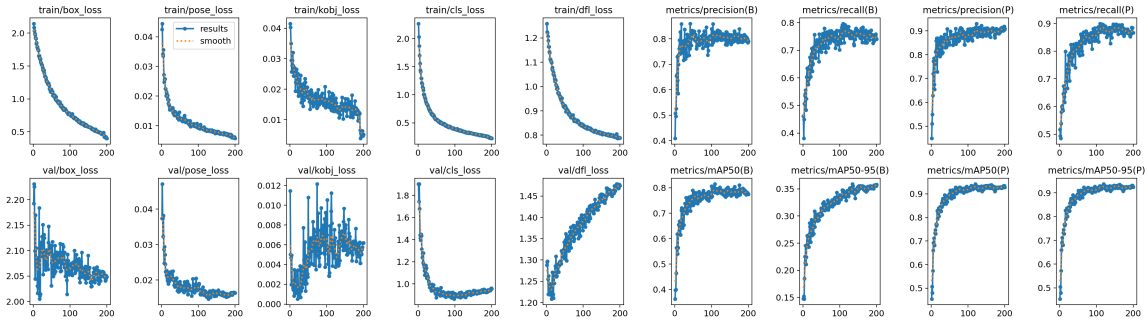

Figure 15: Training results of the pitch detector.

Another challenge present in the dataset - "*Campo futebol keypoint classes v15*" enhanced with 350 Soccerment images - is the uneven distribution of labels in the training set, as illustrated in Figure 16. Certain labels (i.e., pitch keypoints) appear infrequently, such as "D5", "D6", "M4", "O12", while two are entirely absent: "D11" and "O4".
This imbalance poses a significant obstacle to proper generalization, as the model lacks sufficient data to learn the features of these missing or rarely occurring points.

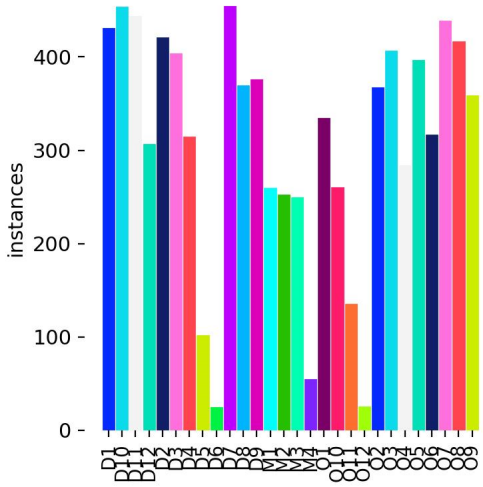

Figure 16: Labels distribution in the pitch detector training set.

Coming to the test results, the pitch detector achieved a mAP50 of 0.711 and a mAP50-95 of 0.318 for bounding boxes while it scored a mAP50 of 0.830 and a mAP50-95 of 0.829 for keypoints as reported, together with precision and recall values, in Figure 17.
These results confirm that the model's ability to successfully identify the right classes, i.e. detecting the correct pitch keypoints as objects, are improvable while, once predicted a bounding box, the model is quite able to correctly place the keypoint in the true line-crossing spot.

metrics/precision(B)	metrics/recall(B)	**metrics/mAP50(B)**	metrics/mAP50-95(B)	metrics/precision(P)	metrics/recall(P)	**metrics/mAP50(P)**	metrics/mAP50-95(P)	fitness
0.776	0.666	**0.711**	0.318	0.874	0.764	**0.830**	0.829	1.187

Figure 17: Test results of YOLOv8x - pose

Chapter 6

Homography

Homography is a powerful concept in projective geometry that describes a mapping between two planar surfaces. In the context of computer vision and image processing, it's used to relate points in one image plane to corresponding points in another image plane or a world plane. This transformation is particularly useful because it can account for perspective changes and geometric distortions that occur when viewing a planar surface from different angles.
The key properties of homography are:

- Linearity preservation: Straight lines in one plane remain straight in the transformed plane.
- Parallelism non-preservation: Parallel lines in one plane may not remain parallel after transformation, accounting for perspective effects.
- Incidence preservation: If a point lies on a line in one plane, its transformed point will lie on the transformed line in the other plane.

6.1 How does homography work?

Mathematically, a homography is represented by a 3x3 matrix H. For two corresponding points p and p' in homogeneous coordinates, the relationship is expressed as: [p' = Hp] where:

$$H = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}$$

and p and p' are represented in homogeneous coordinates:

$$(p = \begin{bmatrix} x & y & 1 \end{bmatrix}, \quad p' = \begin{bmatrix} x' & y' & 1 \end{bmatrix}).$$

The homography matrix H has 8 degrees of freedom as it is defined up to a scale factor (i.e., multiplying H by a non-zero scalar doesn't change the transformation). To understand how homography works in practice, its components are broken down below:

- Rotation and scaling: The upper-left 2x2 submatrix $[h_{11}, h_{12}; h_{21}, h_{22}]$ accounts for rotation and scaling.

- Translation: The elements h_{13} and h_{23} represent translation in x and y directions.
- Perspective distortion: The elements h_{31} and h_{32} account for the perspective effect.

6.2 How is the homography matrix obtained?

Obtaining the homography matrix involves solving a system of equations based on point correspondences between two planes.
The subject of this section is a step-by-step explanation of how the matrix is calculated.
Points correspondence The first step is to identify at least four pairs of corresponding points between the two planes. Let's denote these pairs as (x_i, y_i) in the first plane and (x'_i, y'_i) in the second plane, where i = 1, 2, 3, 4, ...

Setting up the equations For each pair of corresponding points, we can write two equations:

$$\begin{aligned} x'_i &= \frac{h_{11}x_i + h_{12}y_i + h_{13}}{h_{31}x_i + h_{32}y_i + h_{33}} \\ y'_i &= \frac{h_{21}x_i + h_{22}y_i + h_{23}}{h_{31}x_i + h_{32}y_i + h_{33}} \end{aligned} \tag{1}$$

These can be rearranged into linear equations:

$$\begin{aligned} x'_i(h_{31}x_i + h_{32}y_i + h_{33}) &= h_{11}x_i + h_{12}y_i + h_{13} \\ y'_i(h_{31}x_i + h_{32}y_i + h_{33}) &= h_{21}x_i + h_{22}y_i + h_{23} \end{aligned} \tag{2}$$

These equations are then compiled into a linear system $Ah = 0$, where A is a $2n \times 9$ matrix (n being the number of point pairs) and h is a vector of the 9 elements of H.

Solving the System There are several methods to solve this system but, since the default method in the project implementation is the least squares method we will just cover this one[1].

Refinement The initial solution can be then refined using non-linear optimization techniques like Levenberg-Marquardt algorithm to minimize reprojection error.

Normalization and Conditioning The matrix H is usually normalized so that $h_{33} = 1$, as the homography is defined up to a scale factor.

Validation The computed homography is validated by checking the reprojection error, ensuring the determinant of H is non-zero and verifying that H preserves the cross-ratio of any four collinear points.

6.3 Application of homography in the football context

In the context of football analysis, homography plays a crucial role in mapping the positions of the ball from the camera view to a 2D representation of the pitch. This application is particularly

[1] PROSAC-based robust method, Least-Median robust method and Random Sample Consensus are other solving methods.

relevant to this project, where it serves as the final step in the designed computer vision pipeline.

As mentioned in the introduction (1), the pipeline architecture consists of three main blocks:

- Ball detection
- Pitch keypoints detection
- Homography application

The homography transformation is the last component that ties together the information obtained from the first two steps to achieve accurate ball localisation on the pitch radar.

After the ball detector and the pitch detector identify respectively the ball and at least four pitch keypoints the homography matrix is computed exploiting these pitch keypoints which serve as the corresponding points between the image plane and the standardized 2D pitch model. Now this matrix encapsulates the transformation between the camera view and the 2D radar.
Once the homography matrix is computed, it can be applied to the ball's position in the frame (detected in the first block of the pipeline) to map its coordinates from the image plane to the standardized 2D pitch representation.

Chapter 7

The pipeline

After the detailed description of its three components, in this chapter the design of the pipeline is outlined and its implementation broken down step by step.

As Figure 18 shows, the pipeline gets an image as input and passes it to the two fine-tuned models. The ball detector and the pitch detector process the image in parallel and return respectively:

- The centre of the bounding box that identifies the ball
- The pixel coordinates of the pitch keypoints.

Subsequently, the pitch coordinates are passed to the homography module which fits the homography matrix and finally the ball coordinates are multiplied by the matrix components in order to obtain their translated coordinates on the pitch model.

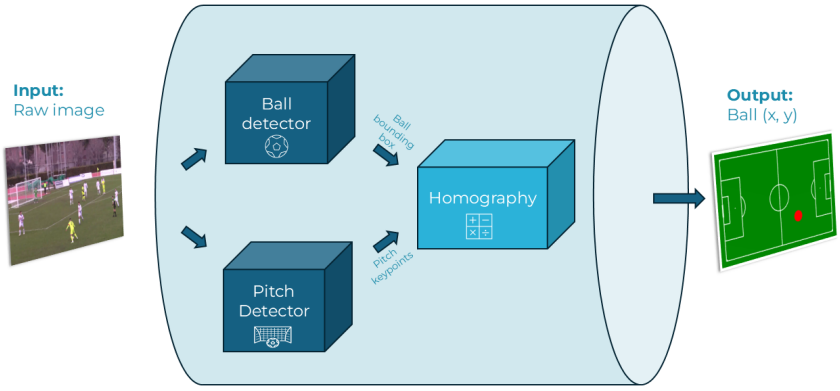

Figure 18: The design of the ball localisation pipeline

7.1 Implementation

The entire pipeline is implemented in Python exploiting PyTorch and OpenCV libraries. In particular, the input image is loaded as a matrix and transformed into a tensor with three color channels

(Red, Green, Blue); both the detection models are PyTorch models saved in the `.pt` format; the homography module is borrowed from OpenCV and uses the `cv.findHomography()` method to calculate the matrix and the `cv.perspectiveTransform()` to apply the transformation to the ball pixel coordinates[1]; finally the result of the homography transformation is outputted as a tuple $(ball_x, ball_y)$.

The ball detector and the pitch detector can run in parallel or sequentially with the first mode being suggested for reducing inference time.

Figure 19 and 20 show an example of a ball localisation taken from an internal demo which allows to upload an image, applies the pipeline to it and return a display of the radar with the ball spot and its coordinates.

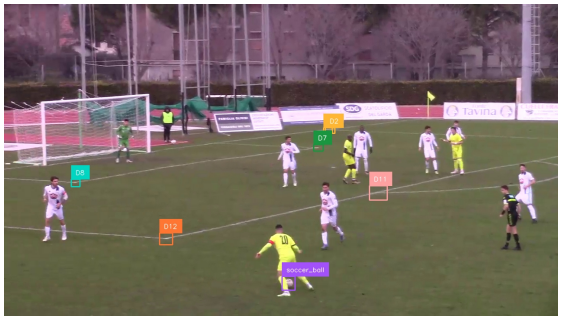

Figure 19: A frame of quality D with the ball and keypoints detections.

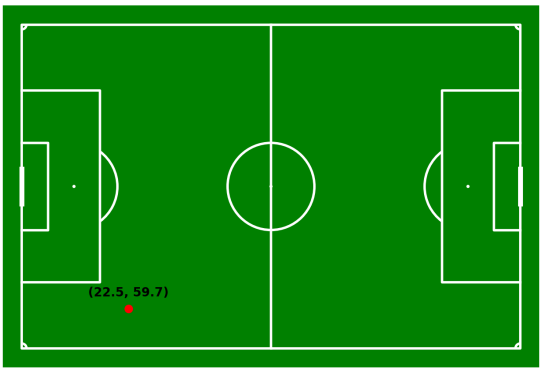

Figure 20: The radar display of the annotated frame. The red dot indicates the ball position.

[1]Read more about homography in OpenCV at https://docs.opencv.org/4.x/d1/de0/tutorial_py_feature_homography.html

Chapter 8

Results

8.1 Correct pipeline execution

The first set of results that needs to be reported is the grade of success of the pipeline execution that means how many times the models perform correct detections and the homography matrix can be calculated.

Figure 21 illustrates the distribution of outcomes following the pipeline execution. This analysis was conducted on a test set of 4'000 images, with equal representation across four quality types (1'000 frames per type).

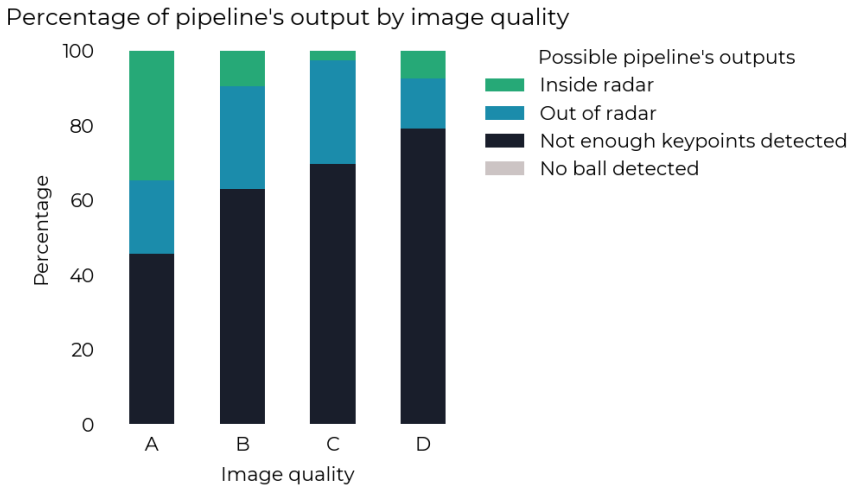

Figure 21: Results of the execution of the pipeline for 4'000 evenly sampled images.

When both the detectors correctly identify the ball and the right keypoints the sphere is positioned inside the pitch in the projected location (in green).
When the ball is correctly identified but the pitch detector erroneously switches some keypoints, the correspondence between the pitch in the image and the 2D pitch model can be distorted, consequently affecting the ball's coordinates as well. Another situation in which the ball goes out of radar is when

it is airborne, as the homography transformation is unable to accurately represent positions above the projected plane (in light blue).
If the pitch detector fails to identify at least four keypoints or the image does not contain at least four of them, the homography matrix cannot be calculated because it would lack of the necessary linear equations to find the system's solutions and thus the pipeline fails (in dark blue).
Finally, seldom the ball detector is not able to detect the ball (indeed it never happened in the test illustrated in Figure 21) and, obviously, in those rare cases the pipeline cannot output a ball localisation.

8.2 Ball position accuracy

The second aspect to evaluate is the accuracy of localisation when the pipeline successfully executes all three of its modules.
To measure this, the pipeline was run on every frame in the Soccerment's ball detection dataset and then metrics based on two subsets of the resulting detections were calculated.
In particular, among the 45'880 frames predicted, 5'508 times the pipeline was correctly executed and from those 5'508 images the two subsets were obtained by taking only the samples were the error was bounded down to two thresholds.

The predefined pitch model reference (see 6.3) applied for the test is given by the coordinates system used in Soccerment's proprietary tagging tool which is a 100×100 pitch and thus we can expect correct detections to have x and y coordinates at most equal to 100. Hence, before extracting the two test subsets, an initial filter was applied to the 5'508 frames in which ball localisation occurred by excluding the predictions that fell outside the soccer field's boundaries, i.e. by removing any prediction with an x or y coordinate greater than 100 in order to reduce the negative effect of the explicitly wrong detections due to the keypoints switch issue (see 8.1).
Then, the detections dataset was refined by applying the actual thresholds that bound the error between predicted and tagged ball position to 141.42 and to 15 units creating two separate subsets of test observations.

The choice of showing the results on these two subsets is explained respectively by the need to show a raw estimation that highlights the effect of a sub-optimal pitch detector model with a bound equal to the maximum achievable error distance in the pitch[1] and by the necessity to filter out very bad detections occurred because of keypoints switch. Therefore, we keep only the detections in which both the ball and the pitch detector are likely to have correctly identified the right objects.

The subset of predictions where the error distance is bounded to 141.42 pitch units contains the 78.5% of all the executed detections (4'324 out of 5'508) and records a not surprisingly high mean error of 44.02, a median error of 42.13 with a standard deviation of 25.30 units.
The plots below show the distribution of errors using a box plot and the relation between true and predicted coordinates in the scatter plots highlighting in red, as a comparing reference, the linear fit of true and predicted coordinates obtained in the second subset.

[1] The maximum error distance achievable is scored when the ball is in a corner but is predicted to be on the opposite corner. It is calculated, through the Pythagorean theorem, as $\sqrt{100^2 + 100^2} = 141.42$

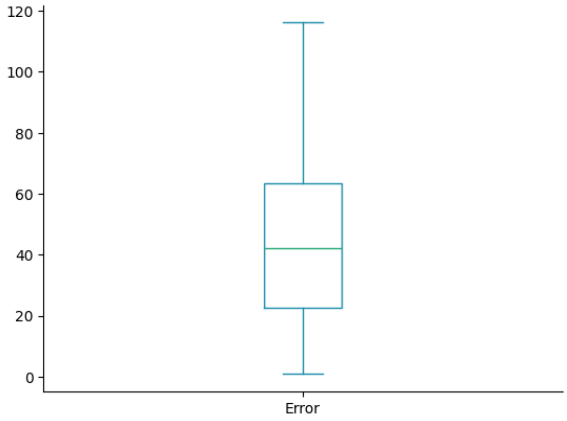

Figure 22: Boxplot of the test errors for error < 141.42 units.

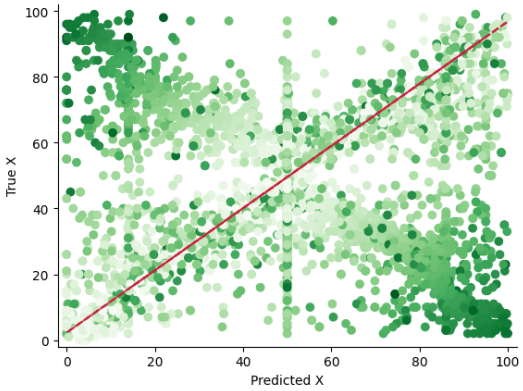

Figure 23: Scatter plot of the tagged and predicted X coordinates for error < 141.42 units.

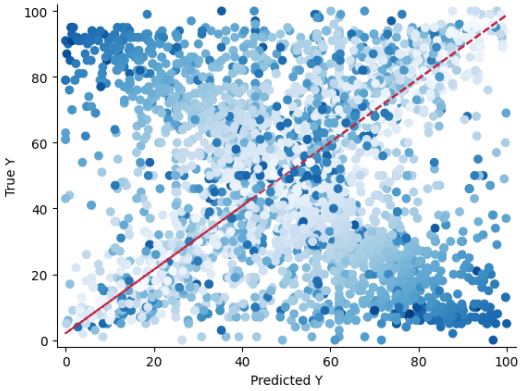

Figure 24: Scatter plot of the tagged and predicted Y coordinates for error < 141.42 units.

Instead, the subset of predictions where the error distance was bounded to 15 pitch units contains the 12.18% of all the executed detections (671 out of 5'508) and achieves a more reliable mean error of 8.74 units, a median error of 8.89 with a standard deviation of 3.84 units.
The following plots - Figures 25, 26 and 27 - show again the distribution of the errors in this subset (which of course are compressed in a smaller range by definition) and the relation between true and predicted x and y coordinates interpolated by the linear fits of the predictions on the true annotations which are the same drawn above in red in Figures 23 and 24.
This trend line helps visualizing the difference in the predictions between the two test subsets and indeed, by comparing the plots, we can notice that the predictions in the larger subset (the one with error bounded to 141.42 units) are more scattered and do not follow an increasing trend, thus highlighting the impact of the pitch detector's misidentifications.

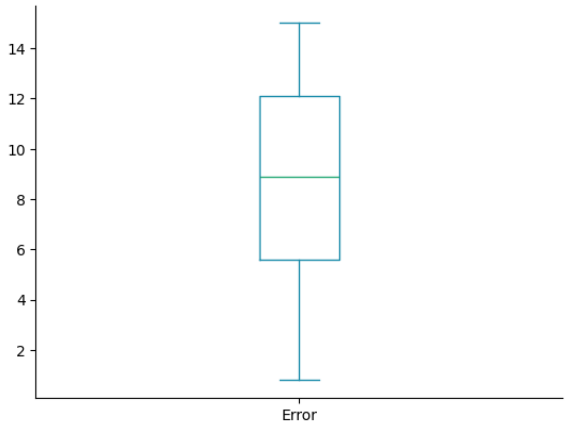

Figure 25: Boxplot of the test errors for error < 15 units.

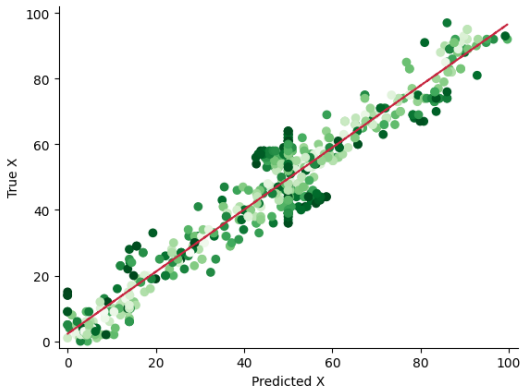

Figure 26: Scatter plot of the tagged and predicted X coordinates for error < 15 units.

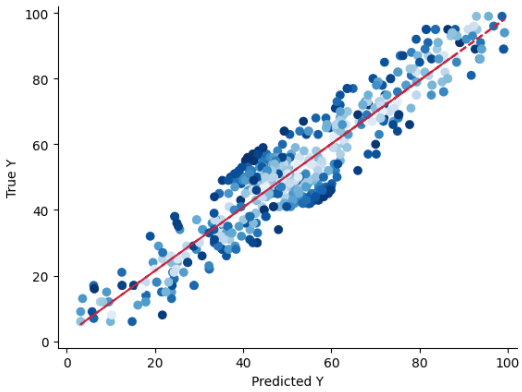

Figure 27: Scatter plot of the tagged and predicted Y coordinates for error < 15 units.

Even though conducted on a significantly smaller proportion of detections, this last test gives a more reliable order of magnitude of how much we can expect the pipeline to be inaccurate when the detection process is correctly executed using a similar pitch reference model. On this point, as a final remark, note that, even if the ball is perfectly localised, part of the error is always imputable to the differences in the distances between the pitch reference model and the actual pitch observed in the frame. Indeed, in lower categories which are not covered by broadcast operators and which therefore almost the totality of non-broadcast footage is produced, pitches dimensions are very heterogeneous among the sports facilities. Thus, such a rigidity poses the conditions to always have a minimum localisation error which is possible to overcome only by customizing the reference dimensions to the real ones of the recorded pitch.

8.3 Inference time

The last result concerns the latency of the pipeline or in other words how much does it take all the three blocks of the pipeline to be executed and return their output.

The ball detector and the pitch detector can predict images both with CPU and GPU and thus were tested on both. In Table 1 the expected inference time in both the hardware types for both the models are presented.
The homography calculation - implemented as described in 7.1 - can, instead, be executed only with CPU and has an expected runtime of $171\,\mu s \pm 30.2\,\mu s$ that is an almost instantaneous computation, thus it is not reported in the table below.

	CPU (11th Gen Intel® Core™ i7-1165G7)	**GPU** (Tesla T4)
Ball detector (YOLOv10m)	$365\,ms \pm 17.1\,ms$	$13.2\,ms \pm 2.8\,ms$
Pitch detector (YOLOv8x-pose)	$997\,ms \pm 299\,ms$	$42.4\,ms \pm 9.6\,ms$

Table 1: Inference time (in milliseconds) for ball and pitch detectors on different hardware

By summing the runtime of the two models we can obtain a fair estimation of the time for a single pipeline execution (since the homography application is a very fast computation) and is observable that when the system runs on CPU it easily exceeds 1 s of runtime while when computations are moved to a GPU the parallelism can decrease inference time down to less than 0.01 s allowing for real-time localisation.

Chapter 9

Conclusions and future work

In this thesis, an automated pipeline was developed for extracting soccer ball coordinates from non-broadcast images, leveraging object detection, keypoints detection, and homography techniques. The objective was to speed up the tagging process in soccer video analysis by automatically determining the ball location in frames captured during gameplay.
The pipeline consists of three main components:

- **Ball detection** using a fine-tuned YOLOv10 model
- **Pitch keypoints detection** employing a YOLOv8-pose model
- **Homography transformation** to map ball coordinates from the image plane to a standardized 2D pitch representation.

The ball detector was trained on a comprehensive dataset of 34'043 images, achieving a mAP50 and mAP50-95 scores of 0.868 and 0.599 respectively. Notably, the fine-tuned YOLOv10 model exhibited an inference time of just 355 ms on a standard CPU, making it suitable for real-time applications.
The pitch detector, a YOLOv8-pose model fine-tuned on an expanded dataset of 1'122 images including 350 manually annotated non-broadcast frames. While the model attained mAP50 scores of 0.711 and 0.830 for bounding boxes and keypoints respectively, the uneven distribution of keypoints labels in the training set posed challenges for generalization.
The homography module leverages the detected pitch keypoints to compute a transformation matrix mapping points between the image and pitch planes. This enables the translation of ball coordinates from pixel space to a predefined pitch model.

The pipeline was evaluated under three aspects: frequency of correct execution, localisation accuracy and inference time.

The first assessment about the frequency of correct execution was performed by testing the pipeline on a subset of 4'000 frames. Out of these 4'000 the system successfully executed all the three steps for ball localisation in 37.4% of the frames, positioning the ball accurately within the pitch 14.5% of the times. The ball detector never failed a detection, however, high localisation errors arose when the pitch detector misidentified keypoints or the ball was airborne.
Localisation accuracy was appraised by running the pipeline over the entire dataset on a 100×100 pitch and then filtering for detections with an error distance less than 15 pitch units in order to analyse only the detections which are likely not to be vitiated by keypoints misdetection. The system

achieved a mean error of 8.74 pitch units with a standard deviation of 3.84.
Finally, the appraisal of inference time highlighted the efficiency benefits of GPU acceleration. While CPU-based execution exceeded 1 second per frame, GPU parallelization reduced runtime to under 10 milliseconds, enabling real-time performance.

Future work

Looking ahead, several avenues for improvement and future work are apparent. Foremost is the need to enhance the pitch detector's robustness and reliability through training on a more extensive and diverse set of annotated images. Expanding the dataset should help mitigate issues stemming from the unbalanced distribution of keypoints labels.

An adjustment to improve localisation error would be to obtain the dimensions of the recorded pitch and input them dynamically in the pipeline to customize the pitch reference model in order for it to stick the reality.

On the infrastructure front, deploying the pipeline as a microservice architecture could yield significant benefits. By running the system on a dedicated machine equipped with a GPU and exposing functionality through an API, tagging operators could access accelerated ball localisation capabilities without requiring specialized hardware.

Additional refinements could explore techniques for handling challenging scenarios, such as occlusions, airborne balls, and atypical camera angles.

Investigating advanced tracking methods and temporal information integration might further improve continuity and consistency across frames.

Conclusive words

This thesis lays the groundwork for an automated system capable of significantly streamlining the tagging process. With the proposed enhancements and architectural optimizations, the pipeline has the potential to become a valuable tool in the world of soccer analytics allowing lower rank teams and less wealthy clubs to buy or collect advanced data at a minor cost by reducing manual effort and enabling a more efficient data collection.

Bibliography

[1] Luca Pappalardo, Paolo Cintia, Alessio Rossi, Emanuele Massucco, Paolo Ferragina, Dino Pedreschi, and Fosca Giannotti. A public data set of spatio-temporal match events in soccer competitions. *Scientific Data*, 6, 10 2019. doi: 10.1038/s41597-019-0247-7.

[2] Joakim O. Valand, Haris Kadragic, Steven A. Hicks, Vajira Thambawita, Cise Midoglu, Tomas Kupka, Dag Johansen, Michael A. Riegler, and Pål Halvorsen. Automated clipping of soccer events using machine learning. In *2021 IEEE International Symposium on Multimedia (ISM)*, pages 210–214, 2021. doi: 10.1109/ISM52913.2021.00042.

[3] J. Brooks, M. Kerr, and J. Guttag. Developing a data-driven player ranking in soccer using predictive model weights. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16)*, pages 49–55. ACM, 2016.

[4] P. Cintia, L. Pappalardo, D. Pedreschi, F. Giannotti, and M. Malvaldi. The harsh rule of the goals: Data-driven performance indicators for football teams. In *Proceedings of the 2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pages 1–10. IEEE, 2015.

[5] P. Cintia, S. Rinzivillo, and L. Pappalardo. Network-based measures for predicting the outcomes of football games. In *Proceedings of the 2nd Workshop on Machine Learning and Data Mining for Sports Analytics (MLSA)*, pages 46–54. ACM, 2015.

[6] Tom Decroos, Jan Van Haaren, and Jesse Davis. Automatic discovery of tactics in spatio-temporal soccer match data. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD '18)*, pages 223–232, London, United Kingdom, 2018. ACM.

[7] Silvio Giancola, Mohieddine Amine, Tarek Dghaily, and Bernard Ghanem. Soccernet: A scalable dataset for action spotting in soccer videos. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. IEEE, June 2018. doi: 10.1109/cvprw.2018.00223. URL http://dx.doi.org/10.1109/CVPRW.2018.00223.

[8] T. D'Orazio, M. Leo, P. Spagnolo, M. Nitti, N. Mosca, and A. Distante. A visual system for real time detection of goal events during soccer matches. *Computer Vision and Image Understanding*, 113(5):622–632, 2009. ISSN 1077-3142. doi: https://doi.org/10.1016/j.cviu.2008.01.010. URL https://www.sciencedirect.com/science/article/pii/S1077314208000441. Computer Vision Based Analysis in Sport Environments.

[9] Dennis Fassmeyer, Gabriel Anzer, Pascal Bauer, and Ulf Brefeld. Toward automatically labeling situations in soccer. *Frontiers in Sports and Active Living*, 3, 2021. ISSN 2624-9367. doi: 10.3389/fspor.2021.725431. URL https://www.frontiersin.org/journals/sports-and-active-living/articles/10.3389/fspor.2021.725431.

[10] Danilo Sorano, Fabio Carrara, Paolo Cintia, Fabrizio Falchi, and Luca Pappalardo. Automatic pass annotation from soccer videostreams based on object detection and lstm, 2020. URL https://arxiv.org/abs/2007.06475.

[11] A. Ekin, A.M. Tekalp, and R. Mehrotra. Automatic soccer video analysis and summarization. *IEEE Transactions on Image Processing*, 12(7):796–807, 2003. doi: 10.1109/TIP.2003.812758.

[12] Rajkumar Theagarajan and Bir Bhanu. An automated system for generating tactical performance statistics for individual soccer players from videos. *IEEE Transactions on Circuits and Systems for Video Technology*, 31(2):632–646, 2021. doi: 10.1109/TCSVT.2020.2982580.

[13] Jürgen Assfalg, Marco Bertini, Carlo Colombo, Alberto Del Bimbo, and Walter Nunziati. Semantic annotation of soccer videos: automatic highlights identification. *Computer Vision and Image Understanding*, 92(2):285–305, 2003. ISSN 1077-3142. doi: https://doi.org/10.1016/j.cviu.2003.06.004. URL https://www.sciencedirect.com/science/article/pii/S1077314203001231. Special Issue on Video Retrieval and Summarization.

[14] Håkon Maric Solberg, Mehdi Houshmand Sarkhoosh, Sushant Gautam, Saeed Shafiee Sabet, Pål Halvorsen, and Cise Midoglu. Playertv: Advanced player tracking and identification for automatic soccer highlight clips, 2024. URL https://arxiv.org/abs/2407.16076.

[15] Xin Gao, Xusheng Liu, Taotao Yang, Guilin Deng, Hao Peng, Qiaosong Zhang, Hai Li, and Junhui Liu. Automatic key moment extraction and highlights generation based on comprehensive soccer video understanding. In *2020 IEEE International Conference on Multimedia Expo Workshops (ICMEW)*, pages 1–6, 2020. doi: 10.1109/ICMEW46912.2020.9106051.

[16] T. D'Orazio, C. Guaragnella, M. Leo, and A. Distante. A new algorithm for ball recognition using circle hough transform and neural classifier. *Pattern Recognition*, 37(3):393–408, 2004. ISSN 0031-3203. doi: https://doi.org/10.1016/S0031-3203(03)00228-0. URL https://www.sciencedirect.com/science/article/pii/S0031320303002280.

[17] T. D'Orazio, M. Leo, N. Mosca, P. Spagnolo, and P.L. Mazzeo. A semi-automatic system for ground truth generation of soccer video sequences. In *2009 Sixth IEEE International Conference on Advanced Video and Signal Based Surveillance*, pages 559–564, 2009. doi: 10.1109/AVSS.2009.69.

[18] Anthony Cioppa, Silvio Giancola, Vladimir Somers, Floriane Magera, Xin Zhou, Hassan Mkhallati, Adrien Deliège, Jan Held, Carlos Hinojosa, Amir M. Mansourian, Pierre Miralles, Olivier Barnich, Christophe De Vleeschouwer, Alexandre Alahi, Bernard Ghanem, Marc Van Droogenbroeck, Abdullah Kamal, Adrien Maglo, Albert Clapés, Amr Abdelaziz, Artur Xarles, Astrid Orcesi, Atom Scott, Bin Liu, Byoungkwon Lim, Chen Chen, Fabian Deuser, Feng Yan, Fufu Yu, Gal Shitrit, Guanshuo Wang, Gyusik Choi, Hankyul Kim, Hao Guo, Hasby Fahrudin, Hidenari Koguchi, Håkan Ardö, Ibrahim Salah, Ido Yerushalmy, Iftikar Muhammad, Ikuma Uchida, Ishay Be'ery, Jaonary Rabarisoa, Jeongae Lee, Jiajun Fu, Jianqin Yin, Jinghang Xu, Jongho Nang, Julien Denize, Junjie Li, Junpei Zhang, Juntae Kim, Kamil Synowiec, Kenji Kobayashi, Kexin Zhang, Konrad Habel, Kota Nakajima, Licheng Jiao, Lin Ma, Lizhi Wang, Luping Wang, Menglong Li, Mengying Zhou, Mohamed Nasr, Mohamed Abdelwahed, Mykola

Liashuha, Nikolay Falaleev, Norbert Oswald, Qiong Jia, Quoc-Cuong Pham, Ran Song, Romain Hérault, Rui Peng, Ruilong Chen, Ruixuan Liu, Ruslan Baikulov, Ryuto Fukushima, Sergio Escalera, Seungcheon Lee, Shimin Chen, Shouhong Ding, Taiga Someya, Thomas B. Moeslund, Tianjiao Li, Wei Shen, Wei Zhang, Wei Li, Wei Dai, Weixin Luo, Wending Zhao, Wenjie Zhang, Xinquan Yang, Yanbiao Ma, Yeeun Joo, Yingsen Zeng, Yiyang Gan, Yongqiang Zhu, Yujie Zhong, Zheng Ruan, Zhiheng Li, Zhijian Huang, and Ziyu Meng. Soccernet 2023 challenges results, 2023. URL https://arxiv.org/abs/2309.06006.

[19] Jinjun Wang, Changsheng Xu, Engsiong Chng, Xinguo Yu, and Qi Tian. Event detection based on non-broadcast sports video. In *2004 International Conference on Image Processing, 2004. ICIP '04.*, volume 3, pages 1637–1640 Vol. 3, 2004. doi: 10.1109/ICIP.2004.1421383.

[20] Shilong Liu, Zhaoyang Zeng, Tianhe Ren, Feng Li, Hao Zhang, Jie Yang, Qing Jiang, Chunyuan Li, Jianwei Yang, Hang Su, Jun Zhu, and Lei Zhang. Grounding dino: Marrying dino with grounded pre-training for open-set object detection, 2024. URL https://arxiv.org/abs/2303.05499.

[21] Tianheng Cheng, Lin Song, Yixiao Ge, Wenyu Liu, Xinggang Wang, and Ying Shan. Yolo-world: Real-time open-vocabulary object detection, 2024. URL https://arxiv.org/abs/2401.17270.

[22] Yian Zhao, Wenyu Lv, Shangliang Xu, Jinman Wei, Guanzhong Wang, Qingqing Dang, Yi Liu, and Jie Chen. Detrs beat yolos on real-time object detection, 2024. URL https://arxiv.org/abs/2304.08069.

[23] Camera calibration. https://www.soccer-net.org/tasks/camera-calibration. Accessed: 2024-09-09.

[24] Ultralytics. ultralytics. https://github.com/ultralytics/ultralytics. Accessed: 2024-09-08.

[25] Dillon Reis, Jordan Kupec, Jacqueline Hong, and Ahmad Daoudi. Real-time flying object detection with yolov8, 2024. URL https://arxiv.org/abs/2305.09972.

[26] Roboflow. football-ball-detection dataset, jul 2024. URL https://universe.roboflow.com/roboflow-jvuqo/football-ball-detection-rejhg. visited on 2024-09-09.

[27] Jonas Theiner and Ralph Ewerth. Tvcalib: Camera calibration for sports field registration in soccer, 2022. URL https://arxiv.org/abs/2207.11709.

[28] N. Homayounfar, S. Fidler, and R. Urtasun. Sports field localization via deep structured models. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4012–4020, Los Alamitos, CA, USA, jul 2017. IEEE Computer Society. doi: 10.1109/CVPR.2017.427. URL https://doi.ieeecomputersociety.org/10.1109/CVPR.2017.427.

[29] Roboflow. football-field-detection dataset, aug 2024. URL https://universe.roboflow.com/roboflow-jvuqo/football-field-detection-f07vi. visited on 2024-09-09.

[30] Roboflow. Campo futebol keypoint classes dataset, aug 2024. URL https://universe.roboflow.com/field-registration/campo-futebol-keypoint-classes. visited on 2024-09-10.

[31] A. Dutta, A. Gupta, and A. Zissermann. VGG image annotator (VIA). http://www.robots.ox.ac.uk/ vgg/software/via/, 2016. Version: X.Y.Z, Accessed: INSERT$_{D}ATE_{H}ERE$.

[32] Abhishek Dutta and Andrew Zisserman. The VIA annotation software for images, audio and video. In *Proceedings of the 27th ACM International Conference on Multimedia*, MM '19, New York, NY, USA, 2019. ACM. ISBN 978-1-4503-6889-6/19/10. doi: 10.1145/3343031.3350535. URL https://doi.org/10.1145/3343031.3350535.

[33] Anthony Cioppa, Adrien Deliège, Silvio Giancola, Bernard Ghanem, and Marc Droogenbroeck. Scaling up soccernet with multi-view spatial localization and re-identification. *Scientific Data*, 9: 355, 06 2022. doi: 10.1038/s41597-022-01469-1.