POLITECNICO DI TORINO

Master's degree course in Computer Engineering

Master's Degree Thesis

Computer vision for detecting and tracking players in basketball videos

Supervisor
prof. Andrea Giuseppe Bottino

Candidate
Sara Battelini
s244091

Internship Tutor
ESTECO SpA
dott. ing. Livio Tenze

Academic year 2019-2020

Summary

The use of Computer Vision in the sports industry is a very new, yet rapidly growing field. One very prevalent application of computer vision algorithms is to gather sports analytics to improve a team's performance. As a team sport, basketball relies heavily on strategy: for this reason coaches need to gather and analyse information and statistics on both their own team and their opponents, a process that can be both long and challenging when executed manually. This thesis work takes place inside a larger project carried out by *ESTECO SpA* which aims to aid coaches in analysing sports games through the use of different technologies. In particular, the purpose of the thesis is to design an algorithm that is able to automatically detect and track the players and the basketball court in broadcast videos: this is achieved through the use of Computer Vision and Deep Learning. After a comprehensive research regarding the state of the art in sports players tracking, a new algorithm was developed following the tracking-by-detection paradigm.

Firstly, an exploration of the best Convolutional Neural Networks for object and pedestrian detection was carried out, along with a study on the standard evaluation metrics. Following a fair comparison of these CNNs on the same basketball dataset a Cascade Mask RCNN, pre-trained on the pedestrian detection dataset Caltech, was chosen as the base detector for the project.

Secondly, a new algorithm was built that is able to robustly track the basketball field from a video frame to the next. It adopts a semi-automatic approach and relies on a combination of many simple Computer Vision algorithms. The court detection is used both to refine the detections, excluding the ones lying outside the playing field, and to create a homography of the basketball court, useful to project the players' positions onto a 2-dimensional model.

Lastly, a research on Multiple Object Tracking was conducted, and some promising algorithms were applied to the previously found detections, obtaining discreet results that will be further improved on in the future.

Contents

List of Figures

Part I

Introduction

Chapter 1

Introduction

The popularity of basketball has been growing steadily over the last few decades, making it one of the most prevalent sports around the world. Thanks to this rising popularity the market surrounding the sport is also spiking, raising the stakes for the playing teams. As a team sport, basketball relies heavily on strategy: for this reason coaches need to gather and analyse information and statistics on both their own team and their opponents. To help improve the performance of a team, analysing both individual players' movements and the overall formation of the team can provide very valuable insights for the team coach[1].

This thesis work takes place inside a larger project carried out by *ESTECO*. The aim of the project is to aid the coaches in the analysis of the games through the use of different technologies. In particular, this thesis focuses on the use of computer vision algorithms to obtain the automatic detection and tracking of players in video clips of basketball games.

People tracking, and consequently players tracking, is a subset of **Multiple Object Tracking (MOT)**. For this thesis work the *tracking-by-detection* paradigm has been employed, which involves first finding the detections of the objects to be tracked, and then linking those detections into tracks by means of a tracking algorithm. *Convolutional Neural Networks* have been analysed and used as a means to detect and track players. Furthermore, a semi-automatic algorithm has been developed that is able to detect and track the court in basketball clips, and generate a homography to project the court and players from the camera view to a 2-dimensional top-view model.

The thesis is organized as follows: chapter 2 introduces the concept of multiple object tracking and the state of the art regarding the use of computer vision in sports videos. The following part of the thesis describes how

the project was implemented: chapter 3 describes the project requirements and the employed dataset; chapter 4 focuses on player detection, comparing different state of the art convolutional neural networks for object detection and adapting the best detector to the project; chapter 5 describes the proposed algorithm developed for court detection and tracking, which is used to find the court homography and to refine the player detection step; chapter 6 is focused on bringing the previous steps together by means of a tracking algorithm. Finally chapter 7 ties up the thesis by presenting conclusions and future developments.

Chapter 2

Introduction to player tracking

2.1 State of the art

Tracking players in a game scenario is a challenging task for many reasons [2][3]:

- the videos are often filmed with a pan-tilt-zoom camera, so the background is not static;
- cluttering and occlusions are frequent and can be quite long;
- the appearance of players is ambiguous, since players of the same team wear uniforms of the same colors;
- players move fast and they have complicated motion patterns; furthermore they often assume poses that are not typically seen from people in common settings;
- players in sports videos move rapidly everywhere within the camera view, incurring severe scale changes, which degrade the performance of the tracker.

Over the last decade some research has been conducted regarding the use of new technologies to aid in tracking and analysing players' movements in sports videos. In 2013 Lu et al.[3] used a *Deformable Part Model (DPM)* to automatically locate sports players in video frames, a logistic regression classifier to classify them into teams, and *Maximally Stable Extremal Regions*

(MSER), *SIFT features*, and *RGB color histograms* for feature extraction. In 2014 Parsons and Rogers[4] proposed a MATLAB program which used computer vision tools like *MAP detectors* and image processing to detect and track basketball players. In 2015 Cheshire et al.[5] built a similar project, performing player detection through the use of *Histogram of Oriented Gradients (HOG)* and color detection. Zhu et al.[6] performed playfield detection by means of *Gaussian mixture models (GMMs)*, and used *Support Vector Machine (SVM)* and *particle filters* for player detection and tracking.

More recently, due to the rise of popularity and accuracy of neural networks, many projects turned to using *Convolutional Neural Networks (CNNs)* to perform player detection. Yoon et al.[7] used *YOLO*[8], a CNN-based multiple object detector, and proposed *Joy2019*, an algorithm that identifies players' jersey numbers and uses them to perform the tracking. Acuna[9] used *YOLOv2* for detecting players and performed tracking by means of *SORT*[10], a tracking algorithm based on simple techniques such as *Kalman Filter* and *Hungarian algorithm*. Ramanathan et al.[11] used a CNN-based object detector and a *KLT filter* for player tracking. Arbués-Sangüesa et al.[2] proposed a bottom-up approach that uses a CNN to detect anatomical key points, which are then used for pose estimation and the deriving player detection; subsequently they used both geometric features and deep learning features to perform feature extraction on the bounding boxes in order to match detections into tracks.

2.2 Multiple object tracking

Multiple Object Tracking (MOT) consists in following the trajectory of different objects in a sequence of images, usually frames of a video. The employment of MOT has found many uses within evolving technologies such as robots, autonomous vehicles, video surveillance, medical imaging and, in general, to aid humans in analysing video footage.

A comprehensive survey conducted by Ciaparrone et al.[12] found that the standard approach employed in MOT algorithms is *tracking-by-detection*: a set of detections are extracted from the video frames first, and then they are used to guide the tracking process. Furthermore, the majority of MOT algorithms perform either all or some of the following steps:

- **Detection stage**: this step is performed by an object detection algorithm, which will output a set of bounding boxes corresponding to the desired objects;

- **Feature extraction/motion prediction stage**: the detections are analysed to extract features regarding appearance, motion, interaction. Optionally, a motion predictor predicts the next position of each tracked target;
- **Affinity stage**: extracted features and motion predictions are used to compute a similarity/distance score between pairs of detections and/or tracklets;
- **Association stage**: the similarity/distance scores are used to associate detections into tracks.

With the recent rise of deep learning, MOT algorithms have started using neural networks in one or more of their stages to increase their accuracy.

2.3 Proposed method

This thesis work makes use of a *tracking-by-detection* algorithm and follows the standard steps. In particular the work was divided into three main sections: ***player detection***, ***court detection and tracking*** —this step is not strictly necessary for player tracking, but it aids in refining the detections and it is needed to perform the homography of the court and project the players' positions to a 2-dimensional court model— and ***player tracking***.

All the tests were performed using a cloud server provided by *Amazon Web Services (AWS)*. The server runs on Linux Ubuntu and features a NVIDIA Tesla K80 GPU with CUDA 9.0. The code for the project is written in **Python**[13], takes advantage of **Anaconda**[14] for environment and package managing, and makes special use of:

- the **OpenCV library**[15], used for image processing and computer vision algorithms;
- the **Pytorch**[16] and **Tensorflow**[17] frameworks, used to handle neural networks in the player detection and tracking steps.

The developed algorithm will be explained in detail in Part II.

Part II

Implementation

Chapter 3

Pre-processing

3.1 Project requirements

The work presented in this thesis aims to automatically detect and track players in the basketball field by analysing basketball broadcast videos. In these broadcast videos of basketball games several different kinds of shots are filmed. For example we can often see close-ups of the players or frontal shots, especially after one of the teams scores a point. These clips are usually just repetitions of already seen footage, but shown from another point of view. Furthermore, during broadcasting, statistics and other information might be shown on the screen, as well as advertisement segments.

Part of the wider project carried out by ESTECO SpA looked at the issue of automatically splitting broadcast videos into sections, and selecting those which are of interest to the coaches. These include mainly the clips shot by the main camera, which is a pan-tilt-zoom camera positioned pitch-side. This part of the project was carried out by another student during his internship through the use of computer vision algorithms. Unfortunately this part of the project was developed simultaneously to this thesis work, so it was not possible to use its results.

3.2 Dataset

Creating a machine learning dataset from scratch, especially in the case of multiple object tracking, is an extremely time-consuming and prone-to-errors process. In the particular case of this project, it would entail watching hours of video footage to select and extract the desirable clips, then using some

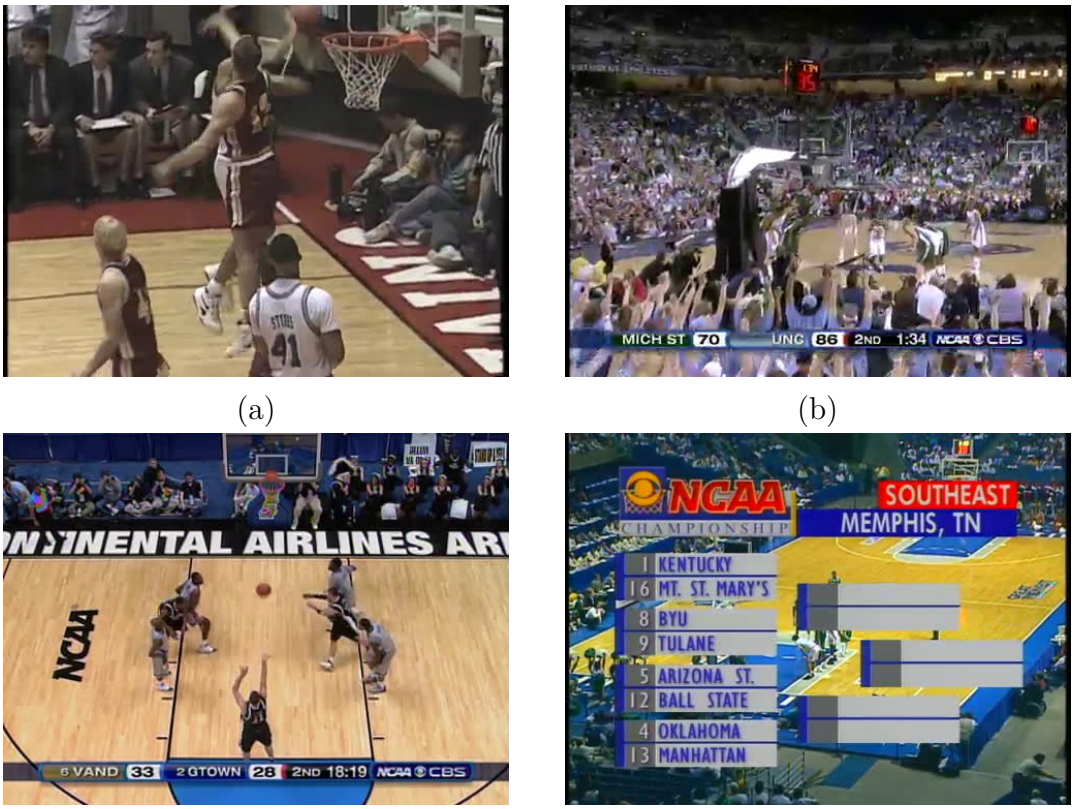
image annotation software like *labelImg*[18] to manually create the ground truth boxes for the players in each frame of the clip. Given the scope of this project, it was deemed more convenient to use an existing dataset, and instead focus on more important steps like detection and tracking.

After extensive research only one dataset was found that was both compatible with the project's requirements and freely available for use: the *Basketball Attention* dataset[11] was created for action recognition, but it also contains annotations of ground truth boxes for player tracking. Unfortunately this dataset is composed of clips extracted from YouTube videos of basketball games, which are not in high definition. Most of the frames in the dataset are **490x360** pixels —from now on these will be referred to as *LQ dataset*—, with a few ones being **1280x720** pixels —from here on called *HQ dataset*. After downloading the dataset a selection process has been performed to delete all clips that did not fit the project's needs (for examples refer to Figure 3.1, Figure 3.2), leaving a final dataset of 424 *HQ* frames and 6072 *LQ* frames.

3.3 Data format

Any dataset built for use of object detection algorithms needs a set of pictures to be analysed, and some information files with the relative ground truths. In machine learning *ground truth* is a term used to describe the ideal expected result. In the specific case of object detection the ground truths correspond to bounding boxes manually assigned by means of human observation to the objects that need to be detected. The ground truth is used both for training a neural network, which will use it as an example to learn how to create its own bounding boxes, and to test it by comparing the ground truths —*the expected result*— to the bounding boxes inferred by the network.

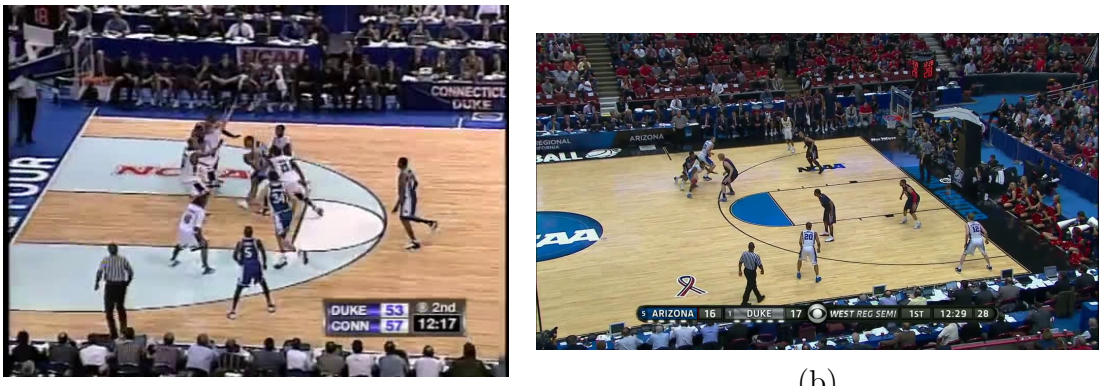
In the chosen dataset the ground truth information is stored in several **CSV** files, one for each frame. CSV, or *comma-separated values*, is a file format which allows to store data in a tabular format. An example of the data format can be seen in Figure 3.3. Since this is not the standard format when it comes to CNNs, a simple Python script was created to automatically transform all information to *COCO format* and unite it into a single **JSON** file. This process is done separately for the HQ and the LQ datasets, which will be treated as two different datasets from here onwards. COCO[19] is the most widely used benchmark API for object detection, so its data format is recognised by any object detection network. An explanation of the data



(a) (b)

(c) (d)

Figure 3.1: Examples of discarded frames: (a) is a close-up, (b) is a frontal shot (c) is a top view, (d) has graphics displayed over the scene.



(a) (b)

Figure 3.2: Examples of selected frames: (a) is 490x360 pixels, (b) is 1280x720 pixels.

format can be seen in Figure 3.4.

,youtube_id,time,x,y,w,h,id
217522,2zBswlkuSFI,1846.845,401.792,356.904,97.152,188.424,person_6_01838069567
217523,2zBswlkuSFI,1846.845,532.608,343.152,72.704,178.272,person_25_01838069567
217524,2zBswlkuSFI,1846.845,553.856,354.24,77.056,170.208,person_15_01838069567
217525,2zBswlkuSFI,1846.845,596.096,224.928,95.36,154.872,person_20_01838069567
217526,2zBswlkuSFI,1846.845,666.112,216.576,52.736,145.008,person_27_01838069567
217527,2zBswlkuSFI,1846.845,823.552,134.856,56.32,168.912,person_16_01838069567
217528,2zBswlkuSFI,1846.845,983.168,238.536,57.344,202.248,person_28_01838069567

Figure 3.3: Examples of the information file of a single frame: *time* is the time in the video corresponding to the frame, x, y are the coordinates of the top-left corner of the bounding box, w, h are width and height of the bounding box, *id* is the id of the player, necessary for the tracking.

```
annotation{
   "id"                : int,
   "image_id"          : int,
   "category_id"       : int,
   "segmentation"      : RLE or [polygon],
   "area"              : float,
   "bbox"              : [x,y,width,height],
   "iscrowd"           : 0 or 1,
}

categories[{
   "id"                : int,
   "name"              : str,
   "supercategory"     : str,
}]
```

Figure 3.4: Explanation of the information and format required by the COCO standard.

Chapter 4

Pedestrian Detection

4.1 Multiple Object Detection

Nowadays neural networks are widely used in the computer vision field, performing a number of tasks that can be collectively described with the term *object recognition*[20]. Within object recognition we can distinguish three main types of tasks:

- **Image Classification** involves predicting the class of an input image and assigning a label to it;
- **Object Localisation** refers to identifying the location of one or more objects in an image and drawing a bounding box around them;
- **Object Detection** combines the previous two tasks to localise and classify one or more objects in an image.

As stated in section 2.3 this project uses the *tracking-by-detection* paradigm to build the basketball player tracker. Therefore the first step is performing object detection. An extensive literature research has been carried out to choose the best convolutional neural network to use for the purpose of this thesis.

4.2 Evaluation metrics

In the machine learning field it is extremely important to have some standard evaluation metrics that can be used to compare different neural networks. In the case of object detection a set of metrics have been defined starting from two simple concepts[21]:

- **Confidence score** is the probability that a bounding box contains an object. It is usually predicted by a classifier.

- **Intersection over Union (IoU)** is defined as the area of the intersection divided by the area of the union of a predicted bounding box B_p and a ground-truth box B_{gt}:

$$IoU = \frac{area(B_p \cap B_{gt})}{area(B_p \cup B_{gt})} \tag{4.1}$$

These two concepts are used to define how a detection is to be considered:

- **True positive (TP)** if $confidence > threshold_1$, $IoU > threshold_2$;

- **False positive (FP)** if $confidence > threshold_1$, $IoU < threshold_2$ or $confidence < threshold_1$, $IoU > threshold_2$;

- **False negative (FN)** if $confidence < threshold_1$ for a detection that should correspond to a ground truth;

- **True negative (TN)** if $confidence < threshold_1$ for a detection that was not supposed to detect anything.

From these basic concepts we can derive the most common evaluation metrics, namely:

- **Precision** is the number of true positives divided by the sum of true positives and false positives:

$$\frac{TP}{TP + FP} \tag{4.2}$$

- **Recall** is the number of true positives divided by the sum of true positives and false negatives (this sum is equal to the number of ground truths):

$$\frac{TP}{TP + FN} \tag{4.3}$$

- **Precision-recall curve** is a curve indicating the association between the recall (x-axis) and precision (y-axis), drawn by setting the confidence threshold at different levels;

- **Recall-IoU curve** is a curve indicating the association between the IoU (x-axis) and recall (y-axis), drawn by setting the IoU threshold at different levels;
- **Average Precision (AP)** is the precision averaged across all unique recall levels, and can be derived from the precision-recall curve;
- **Mean average precision (mAP)** is the mean of average precisions over all object classes;
- **Average recall (AR)** is the recall averaged over all $IoU \in [0.5, 1.0]$ and can be computed as two times the area under the recall-IoU curve;
- **Mean average recall (mAR)** is the mean of average recalls over all object classes.

Object detection algorithms can be compared by means of object detection challenges, i.e. yearly competitions where new algorithms must perform detection on the same provided dataset and are evaluated using the same metrics. The two most famous object detection challenges are **COCO (Common Objects in Context)**[19] and **Pascal VOC**[22]. The Pascal VOC challenge relies on the mAP metric, defined using a single IoU threshold of 0.5, while the COCO challenge defines several mAP metrics, calculated using different IoU thresholds and across different object scales.

```
Average Precision (AP):
  AP                  % AP at IoU=.50:.05:.95 (primary challenge metric)
  AP^IoU=.50          % AP at IoU=.50 (PASCAL VOC metric)
  AP^IoU=.75          % AP at IoU=.75 (strict metric)
AP Across Scales:
  AP^small            % AP for small objects: area < 32^2
  AP^medium           % AP for medium objects: 32^2 < area < 96^2
  AP^large            % AP for large objects: area > 96^2
Average Recall (AR):
  AR^max=1            % AR given 1 detection per image
  AR^max=10           % AR given 10 detections per image
  AR^max=100          % AR given 100 detections per image
AR Across Scales:
  AR^small            % AR for small objects: area < 32^2
  AR^medium           % AR for medium objects: 32^2 < area < 96^2
  AR^large            % AR for large objects: area > 96^2
```

Figure 4.1: Evaluation metrics used for COCO challenge.

4.3 State of the art

Deep learning for computer vision is a relatively new field; as such, it has seen massive improvement over the last few years. This section will present a brief overview of the latest CNNs proposed for the task of object detection[23]. We can differentiate networks into two main categories based on their approach to object detection: two-stage models and one-stage models.

Two-stage models are inspired by image classification: they divide the task of object detection into two smaller tasks, performing firstly a *region proposal* and then applying *image classification* to each proposed region. **RCNN**[24], proposed in 2014, uses selective search to extract around 2000 regions from an image, which are then fed one by one to a CNN for feature extraction and image classification. In the following years other networks were developed inspired by RCNN: **Fast RCNN**[25] uses the whole image and its region proposals as input to its CNN in a single forward propagation instead of feeding the regions one by one; this change saves disk space and improves the speed of the process. **Faster RCNN**[26] improves upon its predecessors by removing the selective search algorithm, which constituted a bottleneck, and letting the region proposal prediction to be carried out by a separate neural network. **Cascade RCNN**[27], introduced in 2017, extends Faster RCNN by presenting a sequence of detectors trained with increasing IoU thresholds, and it manages to avoid the problems of over-fitting at training and quality mismatch at inference. **Mask RCNN**[28] extends Faster RCNN by adding a branch for predicting an object mask in parallel with the existing branch for bounding box recognition. **HTC (Hybrid Task Cascade)**[29] is inspired by Cascade RCNN and Mask RCNN, but instead of performing cascaded refinement on detection and segmentation separately, it interweaves them for a joint multi-stage processing; furthermore HTC adopts a fully convolutional branch to provide spatial context, which can help distinguishing hard foreground from cluttered background.

One-stage models bypass the region proposal step, usually by dividing the image into a NxN grid: each grid cell is then responsible for the object whose center falls into that grid. Thanks to their simpler architecture one-stage models usually require less computational resources and are faster than two-stage models, making them more appealing for use in real-time applications. While two-stage models generally show higher accuracy than one-stage models, the latter category is quickly catching up due to the growing computing ability of machines. In one-stage models a single convolutional neural network predicts the bounding boxes and the class probabilities for these

boxes. **YOLO (You Only Look Once)**[8] was the first attempt to build a real-time object detector. Each image is divided into a grid of SxS and for each cell N bounding boxes and their confidence scores are predicted. The confidence reflects the accuracy of the bounding box and whether the bounding box actually contains an object (regardless of class). YOLO also predicts the classification score for each box for every class in training[30]. **SSD (Single Shot Detector)**[31] achieves a good balance between speed and accuracy by running a convolutional network on the input image only once and calculating a feature map, then running a small 3×3 sized convolutional kernel on this feature map to predict the bounding boxes and classification probability. **RetinaNet**[32] improves upon the other one-stage detectors by proposing a new loss function that reduces class imbalance during training.

All of the aforementioned convolutional neural networks have been trained and tested using the COCO dataset, which is comprised of around 330K images containing 1.5 million object instances across 80 object categories[19]. This means the CNNs are able to detect and assign labels to many different objects. Sometimes a more specific task is needed, as is the case for pedestrian detection. In ***pedestrian detection*** the network doesn't need to know dozens of object categories, but is instead required to be extremely accurate in recognizing a single category: human beings. For this reason some CNNs have also been trained —or fine-tuned— and tested on pedestrian detection datasets like *Caltech*[33], *CityPersons*[34], *EuroCity Persons*[35], *CrowdHuman*[36], and *WiderPedestrian Challenge*[37]. Among the research conducted in the pedestrian detection field some very promising results have been achieved by **Pedestron**[38] and **MGAN**[39].

4.4 Network comparison

To decide which neural network to use for the project, some of the most commonly used and best performing networks have been tested using the basketball dataset. The results, calculated using the main COCO API[19] evaluation metrics, are presented in Table 4.1, while a visual example of the results can be seen in Figure 4.2.

When reviewing the evaluation results it must be taken into account that the tested CNNs are trained for detecting either multiple object classes or pedestrians, not basketball players specifically. This means the models will detect, along with the players on the court, also the audience, the referees, and any other people in the frame. For this reason, average precision values

are quite low —there are many *false positives* because the ground truths exist only for the players. Nonetheless these results are indicative of the performance of the networks relatively to each other, so they can be taken into consideration when choosing how to proceed.

By taking into account both the numerical evaluation and the visual results, **Pedestron's Cascade Mask RCNN trained on Caltech** was chosen to perform the player detection. What was considered most important when making this decision was the network's ability to detect the players even when very occluded or in unusual poses —most networks tend to perform worse when people don't appear in the upright pose which is common of pedestrians. A short computation time, while always important, is not essential for this project, since its objective is not to build a real-time application.

Table 4.1: Performance of CNNs on the basketball dataset. The CNNs were tested on both the HQ and LQ dataset. The time field indicates the seconds to process one image.

Detector	Dataset	AP_{HQ}	AR_{HQ}	$time_{HQ}$	AP_{LQ}	AR_{LQ}	$time_{LQ}$
Cascade Mask RCNN	Caltech	0.249	0.464	1.79	0.279	0.528	0.62
Cascade Mask RCNN	WiderPedestrian	0.249	0.464	1.80	0.279	0.528	0.63
HTC	CityPersons	0.208	0.458	1.21	0.235	0.448	0.52
Cascade Mask RCNN	CrowdHuman	0.216	0.446	1.51	0.225	0.510	0.71
RetinaNet	CityPersons	0.424	0.440	0.79	0.191	0.454	0.22
Faster RCNN	COCO	0.216	0.435	0.76	0.206	0.367	0.26
SSD	COCO	0.192	0.388	0.60	0.138	0.342	0.13
MGAN	CityPersons	0.105	0.335	0.77	0.078	0.295	0.40
YOLOv4	COCO	0.010	0.256	2.09	0.016	0.253	0.71
Cascade Mask RCNN	EuroCity	0.050	0.262	1.30	0.013	0.160	0.46
Faster RCNN	EuroCity	0.035	0.229	0.58	0.029	0.173	0.21

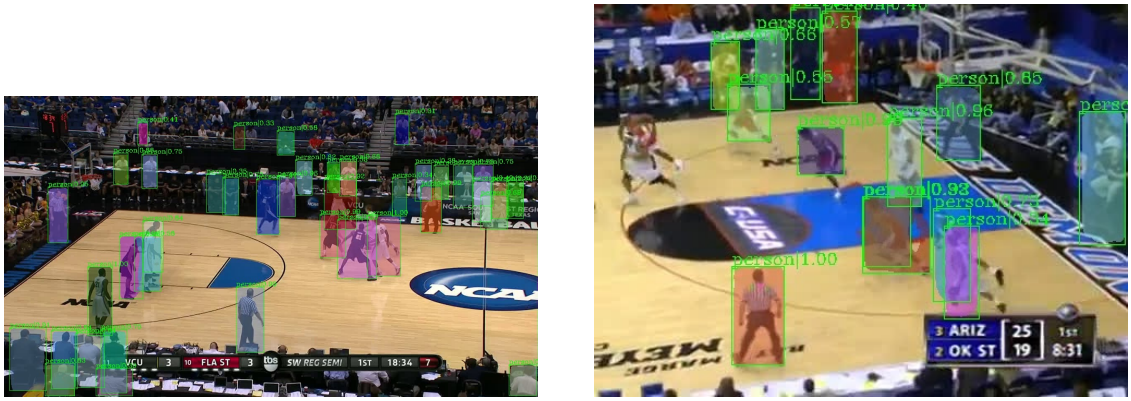
4.5 Transfer learning

A possible course of action to obtain the detection of only the players would be to perform *transfer learning* and fine-tune the chosen CNN to the basketball dataset. Transfer learning consists in taking features learned on one problem, and adapting them to a new, similar problem. In this case this

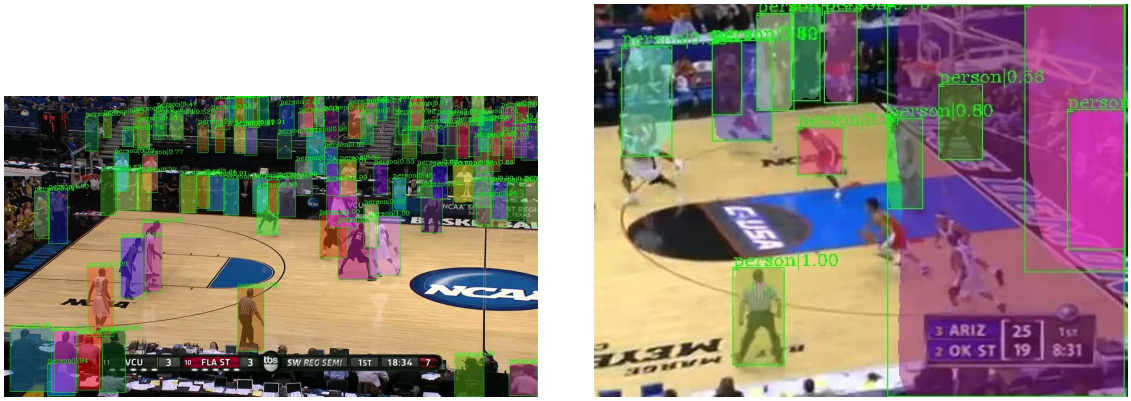
would mean taking the CNN, which is pre-trained on a pedestrian dataset, and training it again on the basketball dataset. Transfer learning is used because training a neural network from scratch requires a dataset of enormous size, which is not often available. For this reason networks are usually first trained on a standard dataset like Open Images[40] or COCO, and then transfer learning is performed with a smaller dataset that is specific to the required task. Two common ways of performing transfer learning are[41]:

- **Feature extraction**, which involves using the representations learned by a previous network to extract meaningful features from new samples. This is done by adding a new classifier, which will be trained from scratch, on top of the pre-trained model, so that the previously learned feature maps can be repurposed for the dataset.
- **Fine-tuning**, which means unfreezing some of the top layers of the base model and training both the unfrozen layers and the newly-added classifier layers. This allows to "fine-tune" the higher-order feature representations in the base model in order to make them more relevant for the specific task.

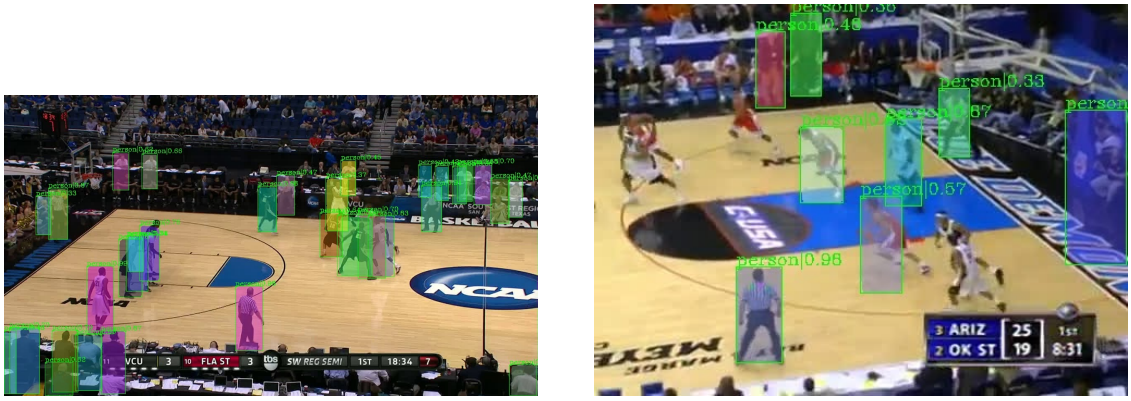
Unfortunately the available dataset was deemed unfit to perform transfer learning successfully: since the chosen CNN seems to be very effective in detecting players even under unfavourable conditions like occlusions and unusual poses, re-training it with a dataset that is quite small and contains such low quality images would not only be useless, but could even prove detrimental to its accuracy. For this reason the process of isolating the bounding boxes corresponding to the players on the court was carried out by using a combination of simpler computer vision algorithms, as will be explained in the next chapter.



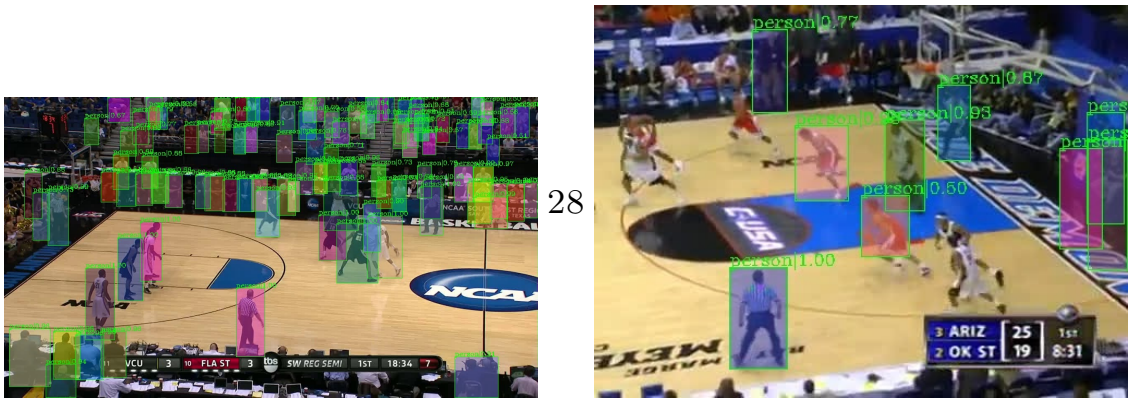
(a) Cascade Mask RCNN - Caltech



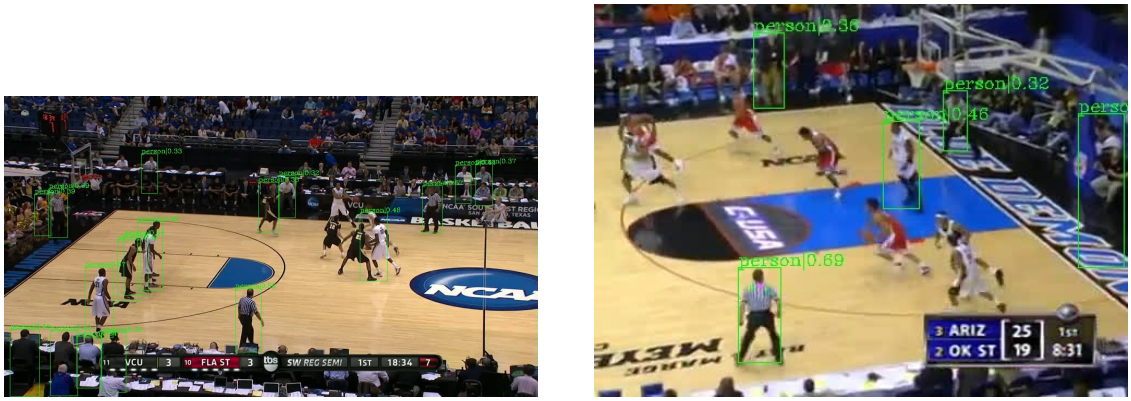
(b) Cascade Mask RCNN - WiderPedestrian



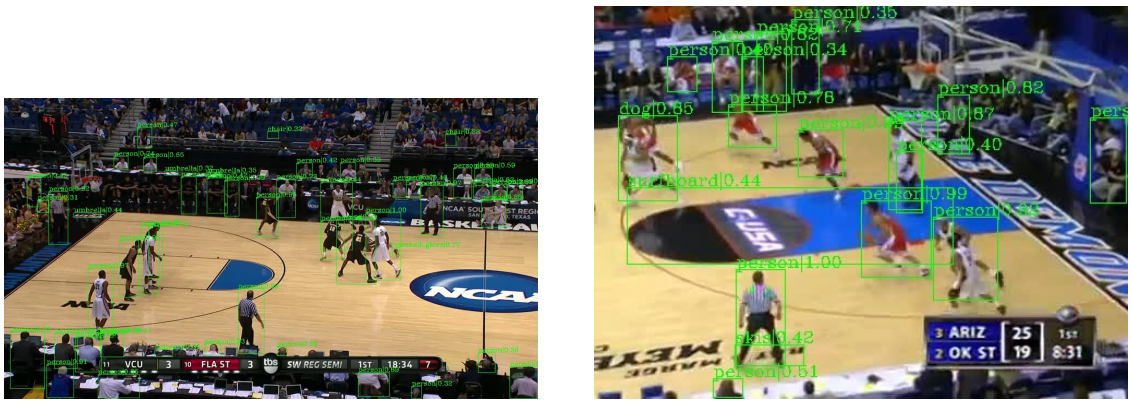
(c) Hybrid Task Cascade - CityPersons



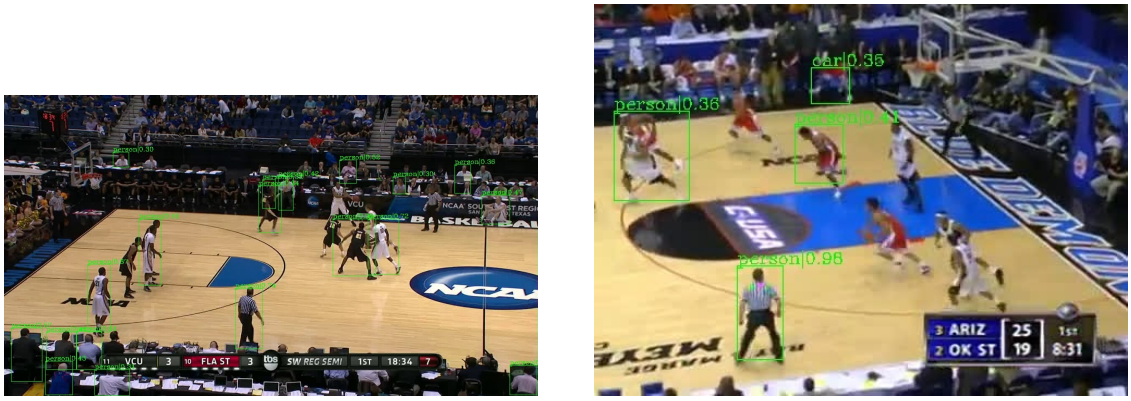
(d) Cascade Mask RCNN - CrowdHuman



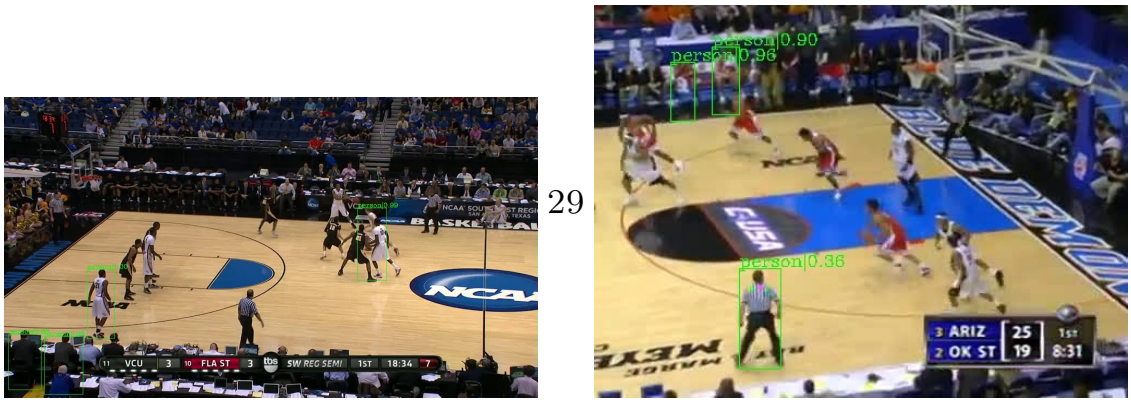
(e) RetinaNet - CityPersons



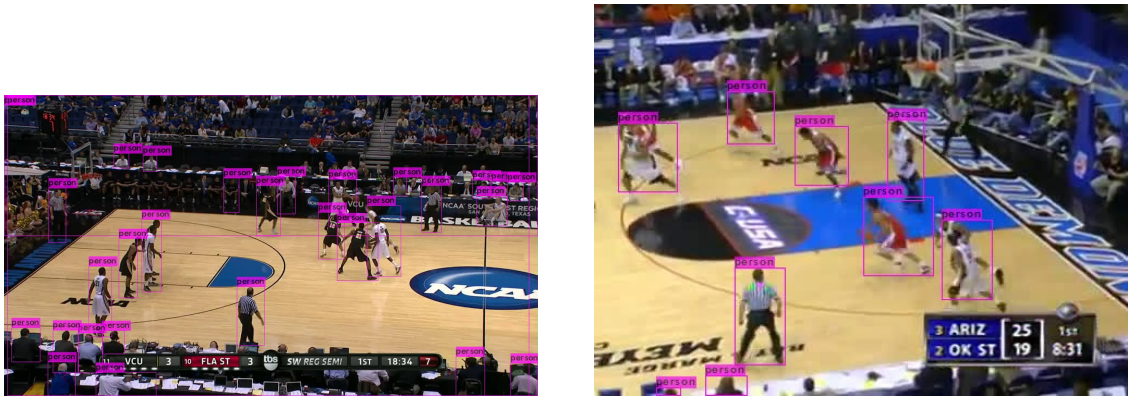
(f) Faster RCNN - COCO



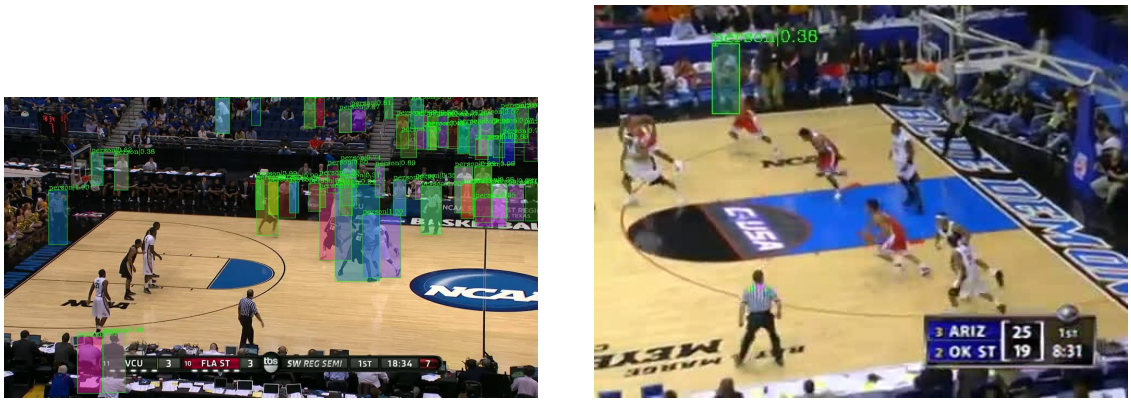
(g) Single Shot Detector - COCO



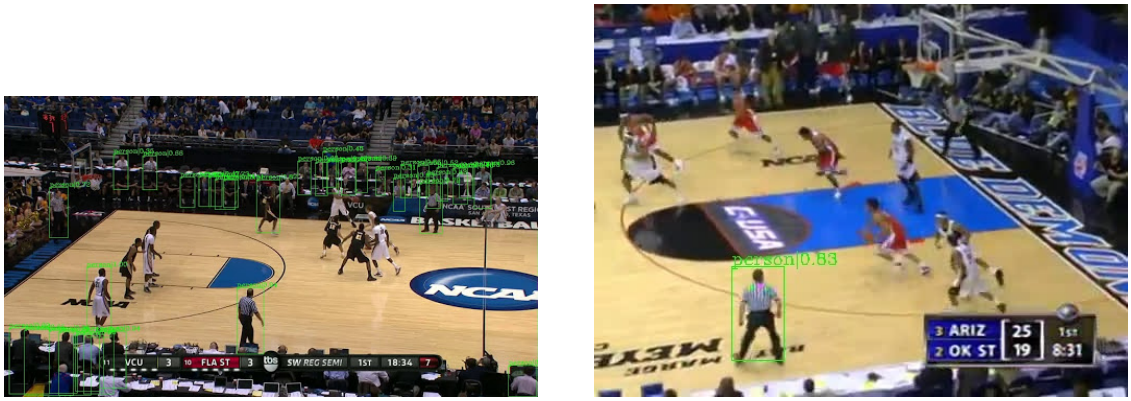
(h) MGAN - CityPersons



(i) YOLOv4 - COCO



(j) Cascade Mask RCNN - Eurocity Persons



(k) Fast RCNN - Eurocity Persons

Figure 4.2: Examples of detections performed by the tested CNNs.

Chapter 5

Court detection and tracking

5.1 Objectives

The next step in the project was to develop an algorithm able to automatically detect and track the basketball court. There are two reasons why this step is needed:

1. By identifying the court area in each frame the bounding boxes corresponding to basketball players can be isolated, ignoring all detections that are not inside the playing field;

2. Some known reference points are needed to build the *homography* of the basketball court; the homography is a transformation that maps the points in one image to the corresponding point in another image. It is needed to project the players' positions in the 2D space, which is important for analysing the players' movements during the game.

5.2 Related work

Some research has been carried out over the last few years about automatic detection of the playing field in several different sports. Wen et al.[42] achieved a robust calibration by considering all video frames simultaneously to reconstruct the *panoramic basketball court*, followed by warping the panoramic court to a standard one **(1)**. Lu et al.[3] made use of *Canny edge detector*[43] to find the edges of the basketball field, using previously detected

bounding boxes to drop edges caused by players, and then applied an *Iterative Closest Points*[44] algorithm to estimate the homography **(2)**. Gupta et al.[45] combined point, line and ellipse matches to get the homography of an ice hockey field; to find the reference matches they initialised the system by choosing a *set of key-frames* (images with overlapping features to cover the whole range of camera motion) and manually chose point correspondences between the key-frames and the geometric model to estimate the homography for all the key-frames; finally, for each new frame they identified the closest key-frame, estimated the homography between them, and obtained the final homography between the new frame and the reference model by concatenating the homography between the frame and its key-frame, and that between the key-frame and the reference model **(3)**. Hess et al.[46] used *Harris-affine detector* and *SIFT descriptor* to detect and describe invariant image features (features that, in theory, should be equally detected in two images of the same object related by a reasonable degree of affine transformation) in American football videos; these features were then used to match the frames to the 2D field **(4)**. Pourezza et al.[47] extracted the field lines of a soccer field through a two-step process: firstly they detected the grass area based on the *histogram of Hue component* in the HSI color space; then a combination of white-pixel detection —performed by analysing the blue component of the input frame in RGB space— and edge detection —achieved through the use of *Sobel operator*— applied to the grass area was used to find good candidates for field lines **(5)**. Fu et al.[48] detected white line pixels through color filtering, width test and line structure constraint, then performed Hough transform[49] to extract the court lines in basketball fields; the camera calibration was then obtained with the intersection points of the extracted court lines and their corresponding points in the court model **(6)**. Parsons et al.[4] detected the court pixels by means of a *MAP detector*, trained by using training masks on the first 10 frames of the video: by isolating the known court pixels over multiple frames, the MAP detector learns to recognize the average RGB color values of the court pixels **(7)**. Hayet et al.[50] tracked a set of locally salient features, detected through a *Harris detector*, with a *KLT tracker*; they then used the tracked points to determine the frame-to-frame homographies **(8)**. Farin et al.[51] identified the white pixels of the court lines through *color detection* and other constraints, then found a set of line candidates using a Hough transformation on the white pixels; line candidates were assigned to lines in the court model using a combinatorial search **(9)**. Liu et al.[52] used the assumption that the playfield color dominates most shots in sports videos by analysing histogram of the

frame in CbCr space, finding its peak and the connected region; finally they adopted *Gaussian mixture model (GMM)* in order to model the playfield color **(10)**.

5.3 Unsuccessful attempts

Many different approaches were attempted in order to tackle the detection of the court and of its lines, which are needed to find the reference points to be used in the homography.

Unfortunately trying to detect the field and the field lines by finding pixels of certain colors like in **(5)**, **(6)**, and **(9)** proved unfeasible: firstly, the field lines are not always white in basketball, since the regulations only state thay must be in a contrasting color to the field; furthermore, even trying to find these lines by means of thresholding or edge detection does not provide a good result for the available dataset, since these lines often appear too thin due to low image quality and other factors. There is also no regulation regarding the color of the basketball court, therefore different basketball stadiums are colored quite differently, making it impossible to detect the court by looking for a certain color range. An example of this can be seen in Figure 5.1.

Some attempts to detect the field lines by means of *thresholding*, *Canny edge detection*, and *Hough transform* were made, but the results were very unreliable: while for some images this process worked quite well, for others it detected either too many lines, or not enough. To perform homography at least four points are needed, and their correspondence must be known; furthermore, to obtain a valid homography no three of these points can lie on the same line. In the case of too few lines detected it was impossible to find these points as intersections of the lines, while if too many lines were found it was a very difficult task to automatically decide which ones were the right ones to be used for finding the reference points. Some examples of this approach can be found in Figure 5.2.

Isolating the court by finding the most dominant color was also unsuccessful: due to the fact that some basketball courts have more than one color, sometimes the dominant color of the image is not the dominant color of the court. This problem is especially common when the camera view closes up on the players, as can be seen in Figure 5.3. Nevertheless an attempt was carried out following a similar approach. The image colors were divided by using *K-means clustering* —this was tried both on the RGB and on the HSV color spaces— and the largest cluster was determined. A variant of this step

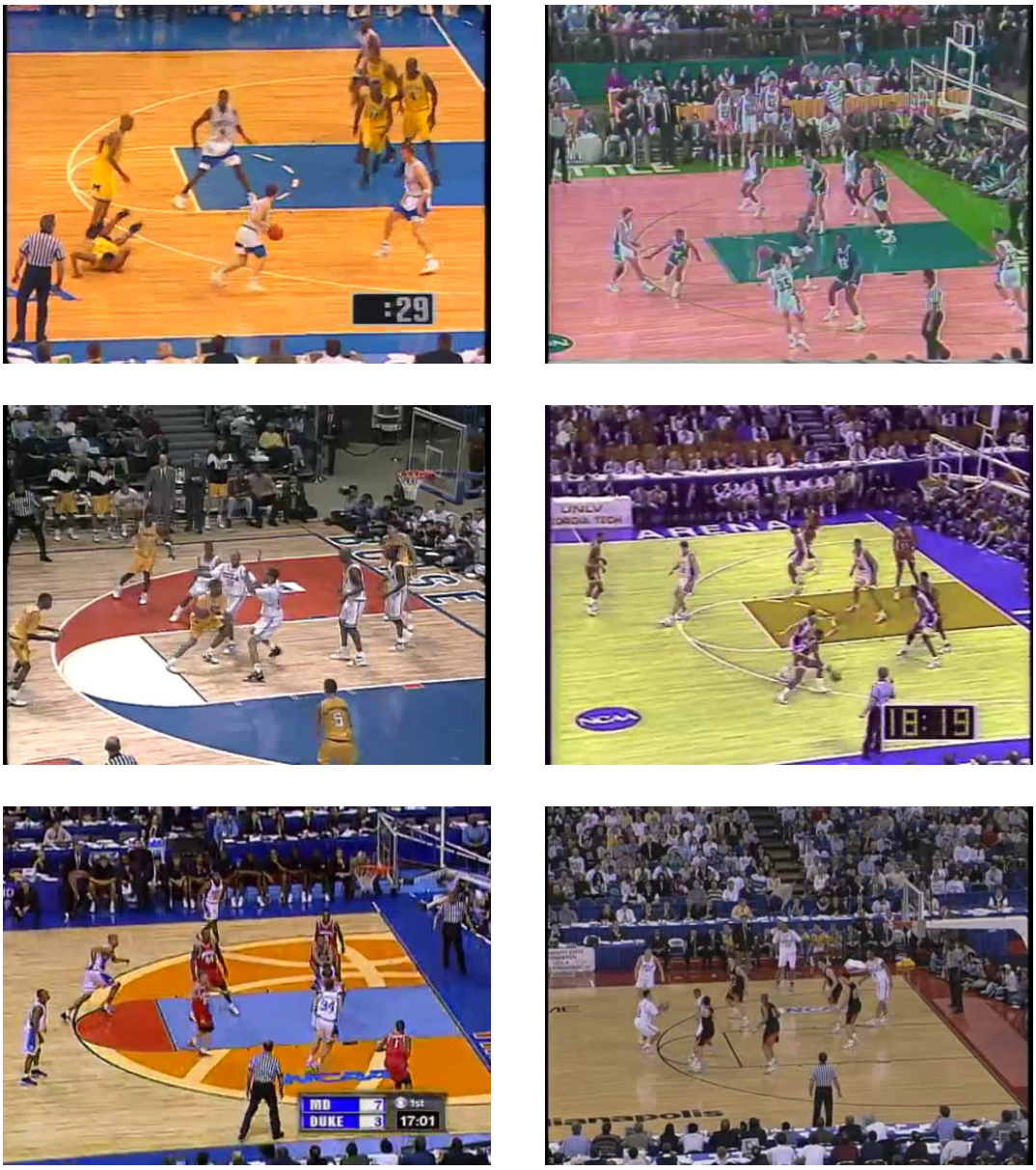
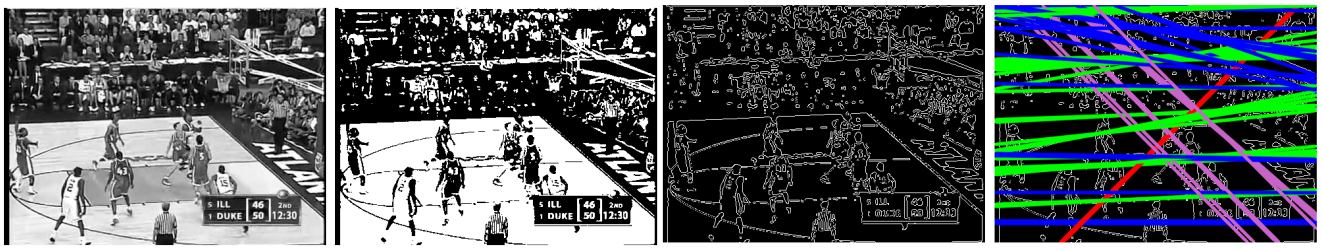
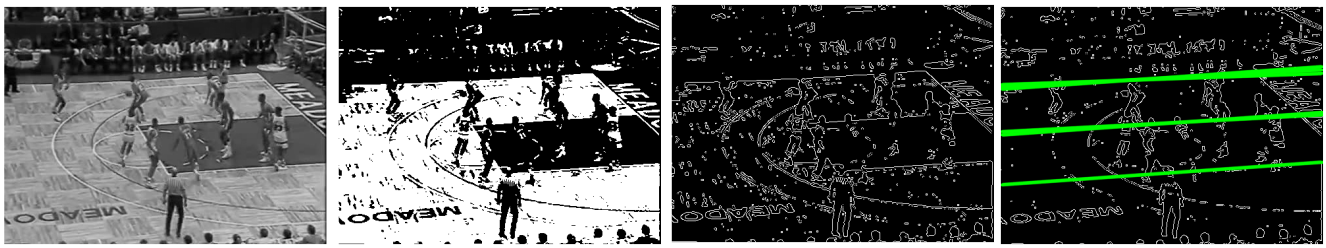


Figure 5.1: Examples of different basketball courts: each one has very different colors and color combinations.

was also created, which ignored the highest quarter of the picture when finding the dominant cluster —the reason being that the higher part of a frame is usually occupied by the stands. The dominant cluster was considered to correspond to the field color. The following steps were the same as in the previous attempt: thresholding, Canny edge detection, and Hough transform.



(a) Too many lines detected: difficult to isolate correct ones.



(b) Not enough lines detected: there are no intersections.

Figure 5.2: Detecting lines through thresholding, Canny edge detector and Hough line transform.

These algorithms were applied to three different variants of the frame:

- the clustered image, i.e. an image where each pixel of the original frame was colored based on the cluster it belonged to;
- an image in which all pixels not belonging to the dominant cluster were zeroed —in the hope that they would not interfere with the subsequent line detection—, thus leaving the original pixels only in some parts of the picture;
- an image where all pixels not belonging to the dominant cluster were zeroed, and all pixels belonging to the dominant cluster were colored with the cluster's center value.

An example of the process can be seen in Figure 5.4. Unfortunately, due to the previously explained problems, none of these cases gave a satisfactory result.

Following a similar approach to **(2)** (obscuring the detected bounding boxes to avoid noise from the players when performing line detection) didn't

Figure 5.3: Examples of frames where the court color is not dominant.

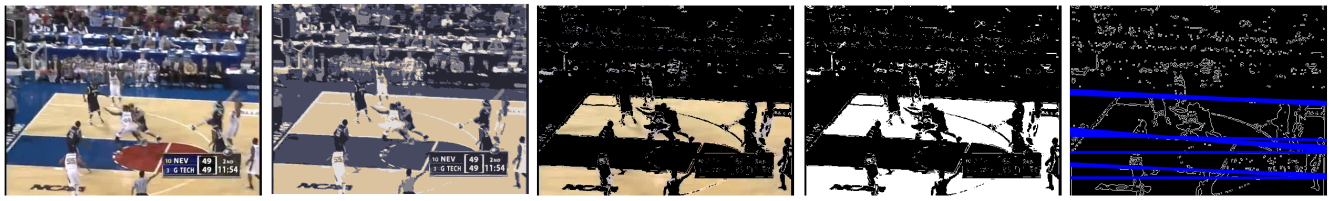
yield any better results. In basketball videos the camera shots are often quite close-up so the players appear quite big: blackening the corresponding bounding boxes obscured too much of the field lines as well, making it hard for line detection algorithms to work properly.

Using the technique explained in **(1)** was impossible, since it needs each clip to contain a view of all parts of the court, while clips in the available dataset usually only focus on one half of the court. **(4)** and **(8)** didn't work either, since unfortunately the *SIFT* and *Harris* algorithms weren't able to find good invariant point features in the basketball images, placing most SIFT points on the audience —since the spectators might move independently of the camera, these points are not reliable— and on the superimposed graphics showing the teams' names and scores —the position of these graphics is constant throughout the video, so these points are useless. An example of the points found through these algorithms can be seen in Figure 5.5. **(7)** was ruled out because it requires the creation of training masks for the first 10 frames of each new video —each time the basketball court changes the *MAP detector* must be re-trained to learn its average colors. An interesting

approach derived from this idea would be to train a CNN to be able to automatically recognize the basketball court and its field lines. Due to time constraints and lack of an appropriate dataset this path was not explored, but it can be taken into consideration for future development.

(a) Image is converted to HSV, k-means clustering is performed, non-dominant colors are blackened, pixels of dominant cluster are flattened to cluster center, bounding boxes are obscured.

(b) Image is kept in RGB, k-means clustering is performed, all pixels colors are flattened to their cluster center, clustered image is turned to gray and thresholded.



(c) Image is kept in RGB, k-means clustering is performed, non-dominant colors are blackened, pixels of dominant cluster keep their color, image is thresholded.

Figure 5.4: Detecting lines through k-means clustering, thresholding, Canny edge detector and Hough line transform.

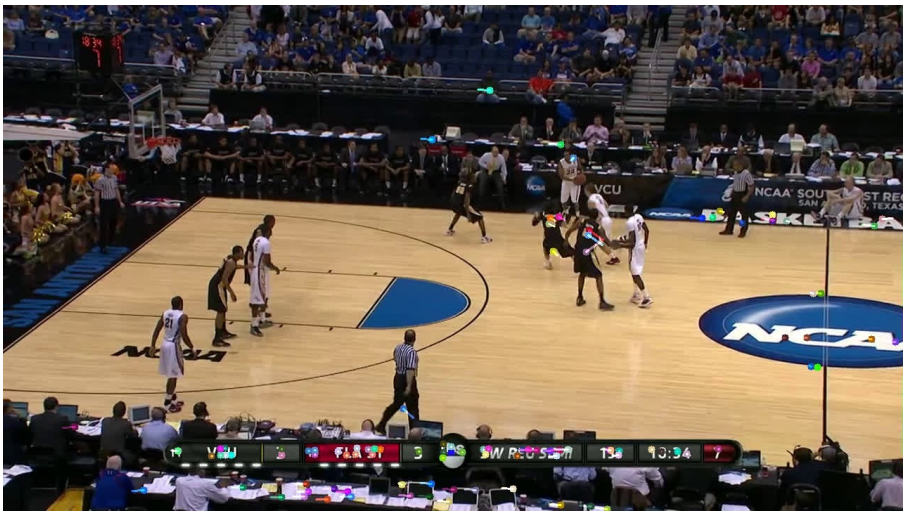


Figure 5.5: Example of found invariant point features.

5.4 Proposed algorithm

Broadcast basketball videos are usually shot from a **single pan-tilt-zoom camera** positioned on the side of the court. The camera is fixed in place, but it has a wide range of motion, so the view of the court must be tracked from frame to frame. Unfortunately the camera position is not explicitly fixed by the game rules, so different stadiums can film from slightly different positions. Since there are no camera attributes to rely on, it is impossible to perform a single manual calibration and use it for all videos, or even just for all frames in a clip.

After trying many different approaches, a process including both *manual* and *automatic* techniques was developed for detecting and tracking the basketball court. A distinction can be made in the behaviour between the first frame of a clip and all following ones.

5.4.1 First frame

A *manual calibration* is needed for the first frame of each clip. The need for a manual input from the user arises from the fact that any fully automatic approach was deemed too unreliable. When the analysis of a new clip is starting, the user is required to select **twelve points** on the first frame: these points correspond to twelve reference points with fixed positions on the reference 2D court —the reference image is the same for all videos. The

user is asked for the points in order, and can either select their location by pressing LMB (the left mouse button) on the desired spot or discard them —if the point is not visible in the current frame— by pressing RMB (the right mouse button) anywhere on the frame. This process only needs to happen once per clip: the points are saved in a *.txt* file, so if the same video needs to be analysed on other occasions, there will be no need to redo this step. The reference image with the positions of the points can be seen in Figure 5.6.

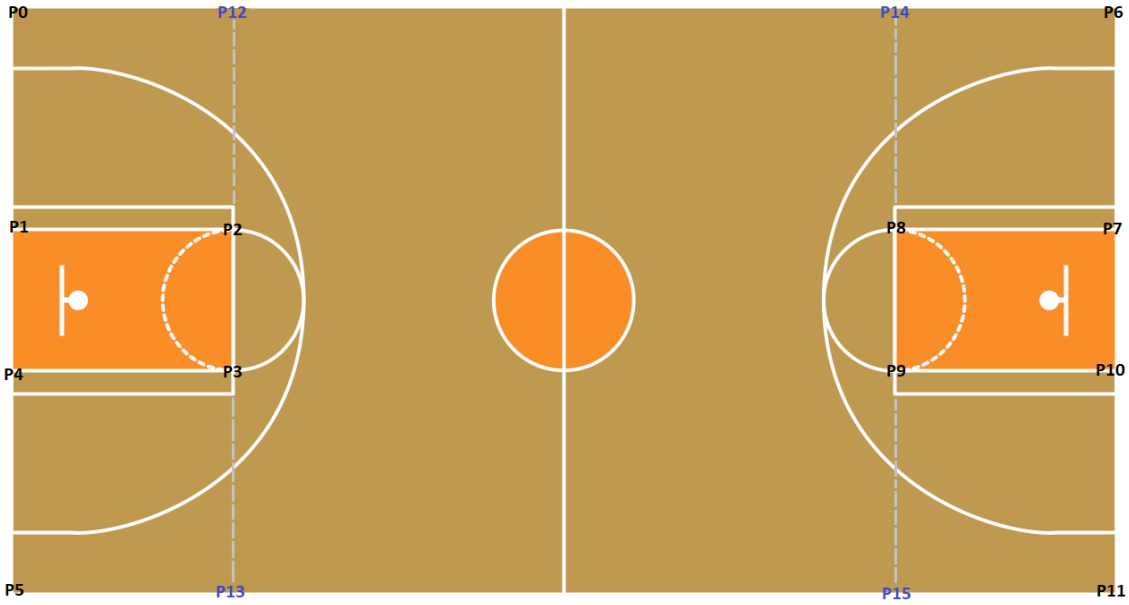


Figure 5.6: 2D basketball court with the reference points.

The collected twelve points (marked in black) are then used to construct the field lines: they are found either through Hough transform or by mathematically finding the lines joining the selected points. The last four points (marked in blue) are found as interceptions of those lines. The lines found by Hough are supposed to be accurate because they must pass some tests —their angle must be inside a certain range, and they must pass in the near vicinity of some of the points selected by the user. During this process the positions of the first twelve points can also change slightly, since points found as interceptions of lines detected by Hough transform are deemed more accurate than points given as input by the user —due to the possibility of human error. A visual example of this process can be seen in Figure 5.7.

These sixteen points —though not all of them will be valid, only the ones visible in the frame— are stored in an array and used to construct the first homography, by using the valid points in the frame and the correspondent

reference points. The homography can be easily found by inputting these two sets of points in *OpenCV*'s function **findHomography()**. The homography will be used to project the players' positions onto the 2D court model. A reverse homography is also computed —i.e. a homography from the reference image to the current frame— and used to create a mask for the frame. This mask is created by projecting the end lines of the 2D court model onto the frame and zeroing all pixels outside of the playing field. It will be used to discard the detections corresponding to the audience —i.e. all bounding boxes outside the playing field (see Figure 5.8). If the homography is considered valid —the validity check will be explained shortly— it is time to proceed to the following frame.

5.4.2 Following frames

The position of the court in the following frame is computed by **optical flow** using the iterative *Lucas-Kanade* method with pyramids. Optical flow is the pattern of apparent motion of image objects between two consecutive frames caused by the movement of the objects or the camera[53].

This algorithm is often used in conjunction to a corner detector like *Harris* or *Shi-Tomasi* but, since these algorithms don't work well on the provided dataset, the points to be tracked are found using a different approach: 100 points are interpolated on each of the straight lines present in the previous frame, and the optical flow is calculated on them. The reason for this choice is that the points along those lines will be more easily trackable, as the area surrounding them is more specific than for points found anywhere else inside the court —because we have the contrast between the line and the court colors— and will be more reliable than other points, which may for example fall on the players or the audience, who might move unpredictably.

Different ways of using these points were attempted, for example using either the whole lines —from one side of the image to the opposite side— or just the parts that are effectively court lines in the picture —bounded by the corners of the playing field— and an attempt was made to use the previously computed bounding boxes of the players, deleting the points that fall inside them to avoid outliers —points which moved too much due to the movement of the player they fall on. Ultimately it was found that the best results were obtained using the whole lines and not using the bounding boxes, since this gives the highest number of points to track, and outliers are thus more easily detected and ignored when computing the homography with **RANSAC** algorithm —RANSAC is an algorithm for robust fitting of

models in the presence of many data outliers[54]. An example of the optical flow points is shown in Figure 5.9.

The optical flow produces two arrays, one filled with the points used in the optical flow algorithm, the other filled with the resulting position of those points in the new frame —only the points which have status=1 and error smaller than a chosen value are stored. These two sets of points are used to compute an inter-frame homography, from the previous frame to the current one; this is where RANSAC finds the inliers and uses them to compute the best possible homography. The resulting homography matrix is then used to transform the corner points from the previous frame to the current one, and then the process of finding the Hough lines and their interceptions is repeated in the current frame, in order to derive another set of points which should, theoretically, be more correct, since they are found using image features of the current frame.

These new corner points are used to compute the homography from the current frame to the reference 2D model, just like it was done for the first frame. If the homography is considered valid, the algorithm passes on to the next frame, using the current frame as reference (i.e. as the "previous frame" from which optical flow is computed); if it is not considered valid, we perform the previously explained process for the following frame, but without changing the reference frame —the non valid frame will be skipped. If the following frame's homography is considered valid, a "backwards" optical flow is computed from that frame to the skipped one, and a new homography is computed. If the homography is still not valid the frame is ignored. If two consecutive frames are not valid, those frames, along with all the following frames of the clip, are ignored, under the assumption that the optical flow from the last valid frame to the following frames would give bad results as well —maybe the camera moved too much between two frames, or the shot is too close-up and not enough corners are visible in the frames. This behaviour (ignoring all consequent frames) was used mainly for testing purposes, since the algorithm was dealing with a big amount of clips: in the final application it could be possible to ask the user to manually input the corner points again —in a middle-clip frame— if needed.

5.4.3 Homography validity check

The validity of the homography is checked through many consecutive steps, each performed only if the previous one results in an invalid homography:

1. The homography is computed from the current frame to the reference image using the points found through Hough line transform. The frame is warped to create a new image corresponding to the 2D basketball court view. In the warped image Hough transform is once again performed, and the lines are counted that abide to certain rules: they must be either horizontal or vertical (within a small margin) and pass through —or be in the vicinity of— at least one of the corner points found in the reference image. If more than one line of similar slope passes through the same point, only one line is considered. If at least three of these lines are found, the homography is considered valid. This low threshold is due to the fact that lines might be missing due to occlusion caused by players, or due to the distortion introduced when warping the image. (Figure 5.10a).

2. The same process as **(1)** is performed, but the homography is computed using the points found through manual input (in the first frame) or through optical flow (in following frames). While these points are usually very similar to the ones found by Hough transform, it is worth doing this check since there might have been some errors in the lines and points found using Hough.

3. If both previous points fail, the inverse homography of **(1)** is computed, and is used to warp the reference image to the current frame. Hough transform is applied to the warped image, and a similar (though inverted) process to the one explained in the previous points is performed: lines are accepted if their slope is within certain bounds, and if they pass through or near to at least one of the corner points found in the frame. In this case the checks on slope and vicinity to points are more constricting, since we don't expect to find inaccuracies due to occlusions or distortion —the image might be slightly distorted, but since it's a drawing the lines will still appear clear enough for Hough transform, if the homography is correct enough. (Figure 5.10b).

4. The same process as **(3)** is carried out, but using the points in **(2)**.

5. If none of the previous steps yields a valid homography, a final check is carried out using the homography found in the previous frame: if the

camera motion was small enough between the two frames, the previous homography could be valid for the current frame as well. The same checks as **(1)** and **(3)** are carried out using this homography.

The homography and the corner points which are considered valid are saved and used as reference for the following frame.

5.5 Results

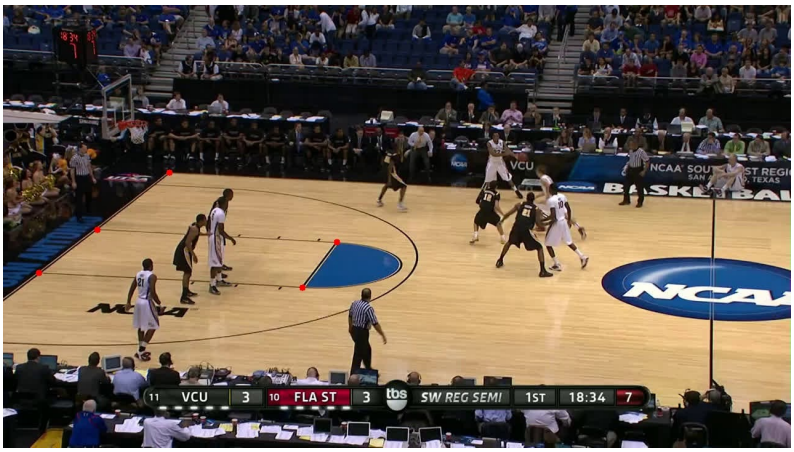
The performance of the proposed algorithm was tested by performing once again **COCO evaluation** on the dataset. Only 2 out of 424 HQ frames were ignored (i.e. their homography was deemed *not valid*). The COCO evaluation resulted in **AP @ [IoU=0.50-0.95] = 0.299** and **AR = 0.444**. The total process lasted *5.59 seconds per frame*. Out of 6072 LQ frames 177 were ignored. The COCO evaluation resulted in **AP @ [IoU=0.50-0.95] = 0.326** and **AR = 0.512**. The total process lasted *2.88 seconds per frame*.

Why are the results of the COCO evaluation so low, while the detections on the images seem very good visually? This result can be attributed to two main reasons:

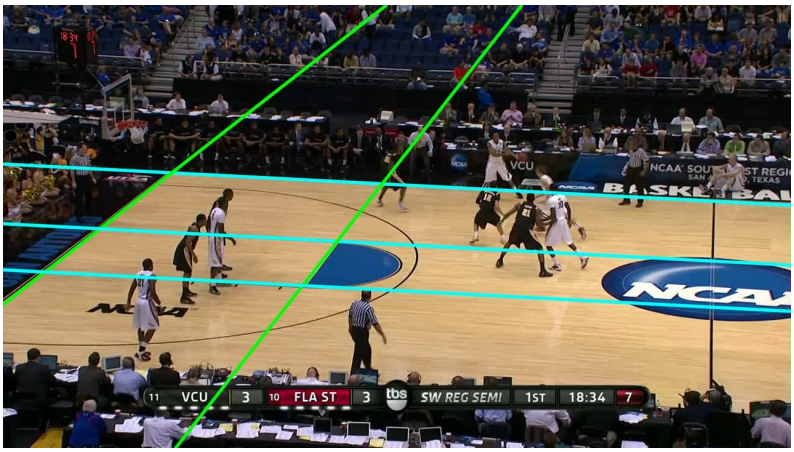
- The values are lowered by the fact that the referees are still being detected —when inside the playing field. This explains the low precision (since there are still some false positives).
- The available dataset has some incorrect annotations. This can be seen in images like Figure 5.11: some players are detected but do not have ground truth boxes, or some additional ground truths are wrongly annotated, usually in areas of high occlusion. It is also noticeable that oftentimes the ground truths have shapes that are not fully compatible with the player they refer to —e.g. the box does not fully contain the player due to its position, or the box is too big for the player— while the detected boxes tend to be more adapted to the player. This might explain why the AP is quite high for lower thresholds of IoU, but quite low for IoU=0.5-0.95. Unfortunately there is no way to control these issues when using a pre-existing dataset.

Time is also an issue: the process of court tracking and homography slows down the program quite a lot —probably because of the many tests to be performed in order to make the process more robust.

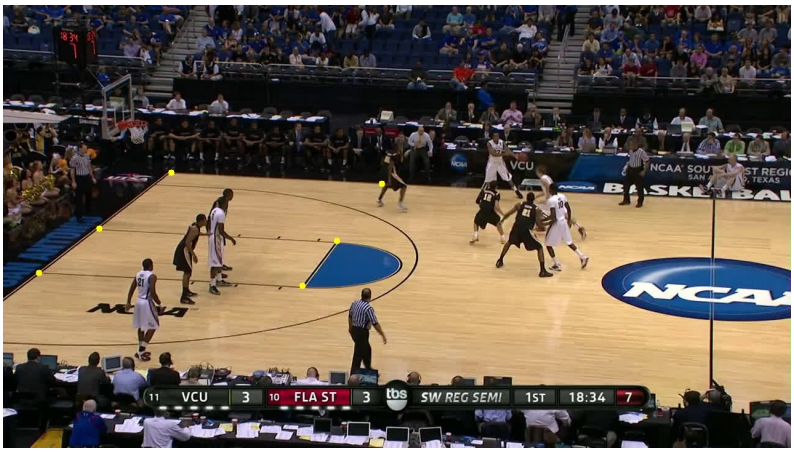
All in all this algorithm yields decent results for the scope of this project, but it can certainly be reviewed in the future to improve its performance and robustness.



(a) Points defined by user.

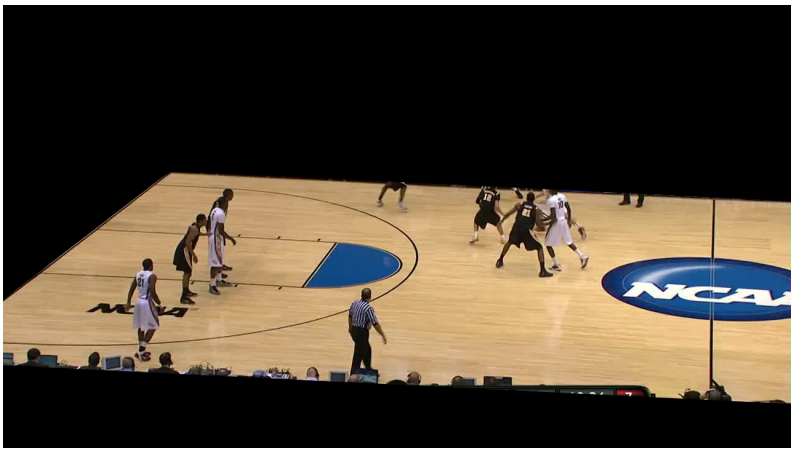


(b) Lines found from defined points.

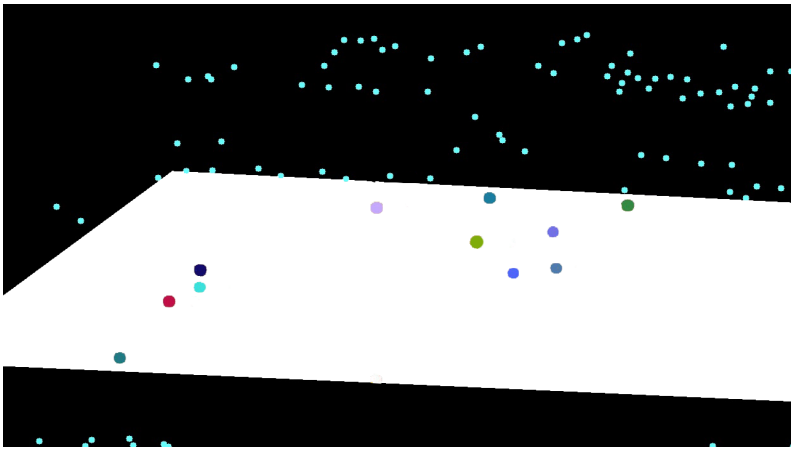


(c) New points, found as interceptions of lines.

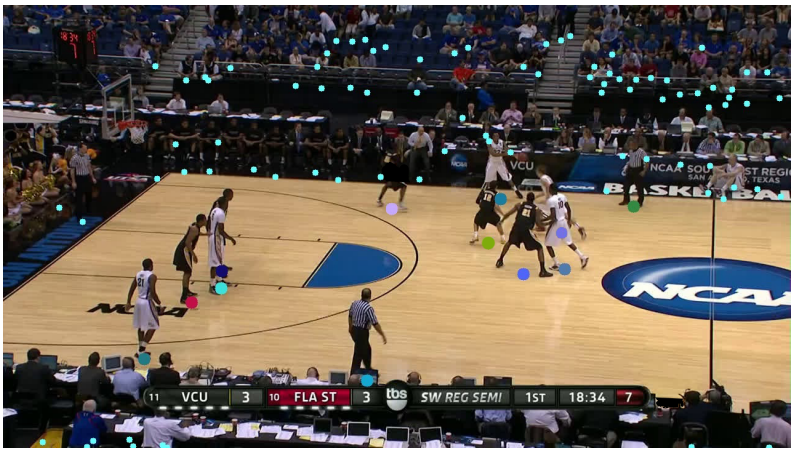
Figure 5.7: Reference points definition.



(a) The mask created through inverse homography.



(b) Bounding boxes that fall outside the playing field are small light blue dots.



(c) Only the bounding boxes of people inside the court (the big dots) are considered.

Figure 5.8: The detections are filtered based on inverse homography. The center point of the lower bound of each bounding box is considered for checking if a person is inside or outside of the court.

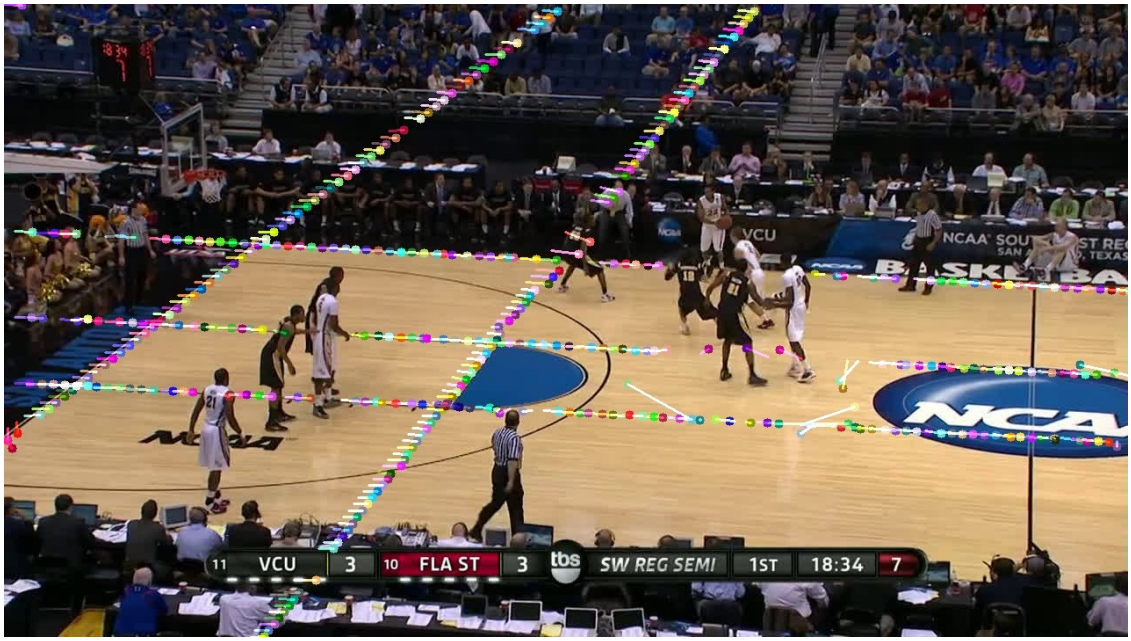
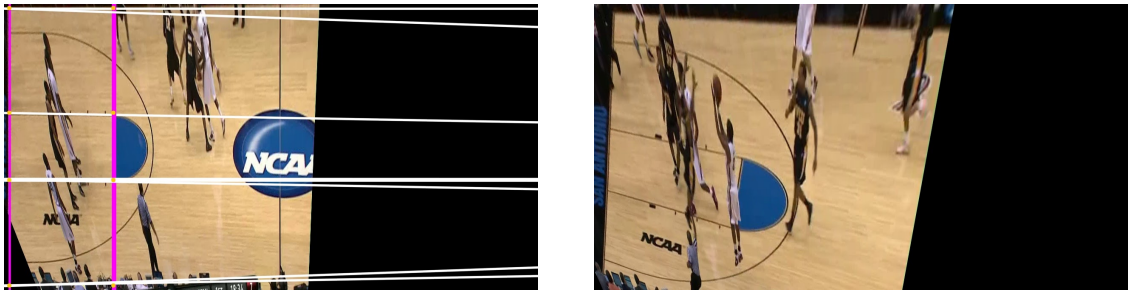
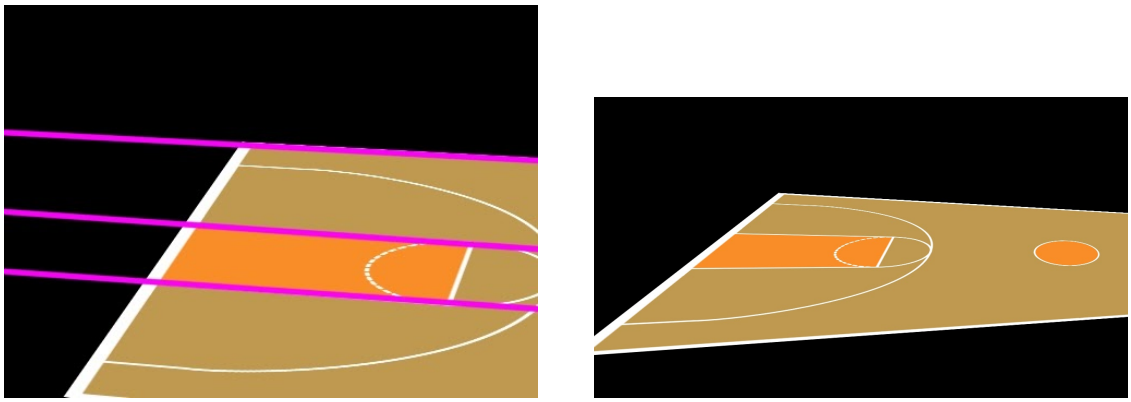


Figure 5.9: Example of optical flow between two frames.



(a) Homography check.



(b) Inverse Homography check.

Figure 5.10: Examples of the checks performed on the homographies: the two homographies on the right are valid, the two on the left are not.

Figure 5.11: Examples of wrong annotations in the dataset: detected bounding boxes are in white, ground truths are in purple.

Chapter 6

Player Tracking

6.1 State of the art

As introduced in section 2.2 MOT algorithms tend to follow some standard steps: *detection*, *feature extraction*, *motion prediction*, *affinity computation*, and *association*[12]. Nonetheless, with the increasing experimentation with the use of deep learning in MOT, these algorithms are becoming much more diverse as they employ different types of neural networks in different steps of the process. The most common step in which neural networks are utilised is the detection step, but lately their use has been experimented in other steps as well, leading to interesting results.

Originally tracking algorithms were based on data association and state estimation techniques based on computer vision tools such as **Kalman Filter** and **Hungarian algorithm**: such is the case for Bewley et al.[10]'s *SORT*. Bochinski et al.[55] presented a slightly different approach, taking advantage of high precision detections and the high frame rates of video footage to propose a simple *IOU tracker* which continues a track by associating the detection with the highest IOU to the last detection in the previous frame if a certain threshold is met. Wojke et al.[56] expanded SORT, integrating appeareance information into the algorithm by means of a CNN for feature extraction, which was trained on a pedestrian detection database; the new algorithm, which replaces the old association metric with a more informed one combining motion and appearance information, is called *DeepSORT*. Other studies have used deep learning for feature extraction: for example, Kim et al.[57] incorporated visual feature extraction into the *Multiple Hypothesis Tracking* algorithm using a pretrained CNN, and Yu et al.[58] used a modified version of *GoogLeNet*, pre-trained on a re-identification dataset, for

extracting features.

Some studies have introduced the idea of using **Siamese Neural Networks**, which consist of two identical artificial neural networks working parallelly in tandem to naturally rank similarity between inputs by comparing their outputs[59]. For example Kim et al.[60] introduced a tracking system based on similarity mapping by *Enhanced Siamese Neural Network (ESNN)*, which accounts for both appearance and geometric information; inspired by Siamese networks, Zhang et al.[61] proposed a loss function called *SymTriplet*, wich uses three CNNs with shared weights; in a similar fashion Son et al.[62] introduced *Quad-CNN*, which learns to associate object detections across frames using quadruplet losses by considering target appearances together with their temporal adjacencies for data association. Furthermore, sometimes **Recurrent Neural Networks (RNN)** and **Long Short-Term Memory (LSTM)** networks can be employed for motion prediction, as is shown by Sadeghian et al.[63]; another example of their use can be seen in Babaee et al.[64]'s method which, in order to preserve the ID number of targets after occlusion, reconstructs missed detection boxes, predicting the detections in next frames by learning the motion of targets using a RNN.

Among the top-performing algorithms in **MOT 2020** and **CVPR 2019** challenges many interesting new approaches are explored: Karthik et al.[65]'s *SimpleReID* is an unsupervised re-identification network which first generates tracking labels using SORT, and then trains a ReID network to predict the generated labels using cross-entropy loss; Ren et al.[66]'s approach, explicitly designed for handling crowded scenes, incorporates object counting into their model, and solves the multiple object detection and tracking simultaneously over the whole video sequence; Zhang et al.[67]'s *FairMOT* is a new baseline for one-shot MOT tracking which jointly detects objects and learns Re-ID features while avoiding the use of anchors, which were found to be the reason for the bad results in other one-shot algorithms; Bochinski et al.[68] extended the original IOU tracker by incorporating a visual single-object tracker to increase the robustness against missing detections; Bergmann et al.[69] introduced *Tracktor*, which tackles multi-object tracking by exploiting the regression head of a detector to perform temporal realignment of object bounding boxes.

6.2 Evaluation metrics

Like multiple object detection, MOT also requires a set of standard evaluation metrics to compare the performance of different tracking algorithms. Three main sets of metrics are commonly used for evaluating MOT algorithms:

1. *Classical metrics*[70]

   - **Mostly Tracked (MT)**: number of ground truth trajectories for which more than 80% of the trajectory is tracked;
   - **Mostly Lost (ML)**: number of ground truth trajectories for which more than 80% of the trajectory is lost;
   - **Fragments**: number of fragments of trajectories, i.e. result trajectories which are less than 80% of a ground truth trajectory;
   - **False trajectories**: number of false trajectories, i.e. result trajectories corresponding to no real object;
   - **Identity switches**: number of identity exchanges between a pair of result trajectories.

2. *CLEAR MOT metrics*[71]

   - **FP**: the number of false positives in the whole video;
   - **FN**: the number of false negatives in the whole video;
   - **FM**: the total number of fragmentations;
   - **IDSW**: the total number of ID switches;
   - **MOTA Score (accuracy)** —with GT being the number of ground truth boxes:

$$MOTA = 1 - \frac{(FN + FP + IDSW)}{GT} \in (-\infty, 1] \tag{6.1}$$

   - **MOTP Score (precision)** —where c_t denotes the number of matches in frame t, and $d_{t,i}$ is the bounding box overlap between the hypothesis i and its assigned ground truth object:

$$MOTP = 1 - \frac{\sum_{t,i} d_{t,i}}{\sum_{t} c_t} \tag{6.2}$$

3. *ID Scores*[72]
   Unlike the CLEAR MOT metrics, which measure performance by how often mismatches occur, these scores focus on how long the tracker correctly identifies targets. To obtain them a bipartite graph is created that associates one ground-truth trajectory to exactly one computed trajectory by minimizing the number of mismatched frames over all available data. The first set of graph vertices V_T has a regular node for each true trajectory, and a false positive node for each computed trajectory; the second set V_C has a regular node for each computed trajectory and a false negative for each true one. The costs of the edges are set in order to count the number of false negative and false positive frames in case that edge were chosen. After association there are four different possible pairs: if a regular node from V_T is matched with a regular node from V_C, a true positive ID is counted. Every false positive from V_T matched with a regular node from V_C counts as a false positive ID. Every regular node from V_T matched with a false negative from V_C counts as a false negative ID, and every false positive matched with a false negative counts as a true negative ID. **IDTP**, **IDFN**, and **ISFP** are the sums of the weights of the edges selected as, respectively, true positive, false negative, and false positive ID matches. The ID scores are then defined as:

   - **Identification precision**:

$$IDP = \frac{IDTP}{IDTP + IDFP} \tag{6.3}$$

   - **Identification recall**:

$$IDR = \frac{IDTP}{IDTP + IDFN} \tag{6.4}$$

   - **Identification F1**:

$$IDF1 = \frac{2\ IDTP}{2\ IDTP + IDFP + IDFN} \tag{6.5}$$

6.3 Employed algorithms

Unfortunately most of the methods mentioned in section 6.1 are not open source, and implementing them from scratch would prove to be an extremely difficult (if not impossible) task given the time and resources reserved for this work. Among the open source algorithms many are not suitable for this project, either because they were created for a different context —for example, many are built for use in mobile robots— or because they are not compatible with the other parts of this work. For this reason the **DeepSORT** algorithm was chosen to perform the player tracking. This choice was based on some key concepts:

- it is a simple and fast algorithm, which are very useful features since the previous steps of the project are quite complex and slow;
- it is very popular and has been shown to offer a good performance and to be fairly robust;
- its feature extractor is trained on a pedestrian identification dataset, which supposedly makes it good at recognising human features.

Briefly speaking, DeepSORT works by:

1. receiving a set of bounding boxes from a previous detection step;
2. generating, by means of a deep network, the appearance features from the image referring to said bounding boxes, and storing them in a single feature vector: these features will then be used for person re-identification;
3. building the tracks by means of Kalman filtering and Hungarian algorithm, with the help of the previously extracted appearance features.

Unfortunately the results obtained with this tracking algorithm were subpar: in particular, the algorithm often ignored players even though they had been previously detected by the detection algorithm (see Figure 6.1). The disappearance of a bounding box is sometimes expected, denoting the loss of a track, but in this case it occurred too often to be considered acceptable.

Under the assumption that this problem was mainly caused by the feature extraction step, the **SORT** algorithm was selected as a second choice. This algorithm is the predecessor of DeepSORT: it involves the same tracking methodology but foregoes the use of appearance features. Although this

Figure 6.1: Examples of vanishing bounding boxes: original detections are on the left, DeepSORT results are on the right.

method is a bit outdated, it is still quite popular: in the recent past many tracking algorithms have started to focus on the use of deep networks, but many others are being kept simpler, relying on the increasingly accurate object detections. With the use of SORT the problem of disappearing bounding boxes was mostly solved.

6.4 Results

The performance of both DeepSORT and SORT was tested using the most common MOT evaluation metrics[73]. The results can be seen in Table 6.1a and Table 6.1b respectively. Some comparison pictures can be seen in Figure 6.2. While better than those obtained with DeepSORT, the results found by using SORT are still not perfect.

Table 6.1: Performance of tracking algorithms on the basketball dataset.

(a) DeepSORT

	IDF1	IDP	IDR	Recall	Precision	MOTA	MOTP
HQ dataset	37.7%	52.6%	29.3%	37.6%	67.3%	17.2%	0.298
LQ dataset	38.0%	57.6%	28.3%	34.6%	70.4%	18.2%	0.293

(b) SORT

	IDF1	IDP	IDR	Recall	Precision	MOTA	MOTP
HQ dataset	51.7%	51.0%	52.5%	75.8%	73.7%	40.5%	0.290
LQ dataset	29.0%	18.8%	63.0%	88.4%	26.5%	19.2%	0.286

There are many factors that come together and hinder these algorithms' performance:

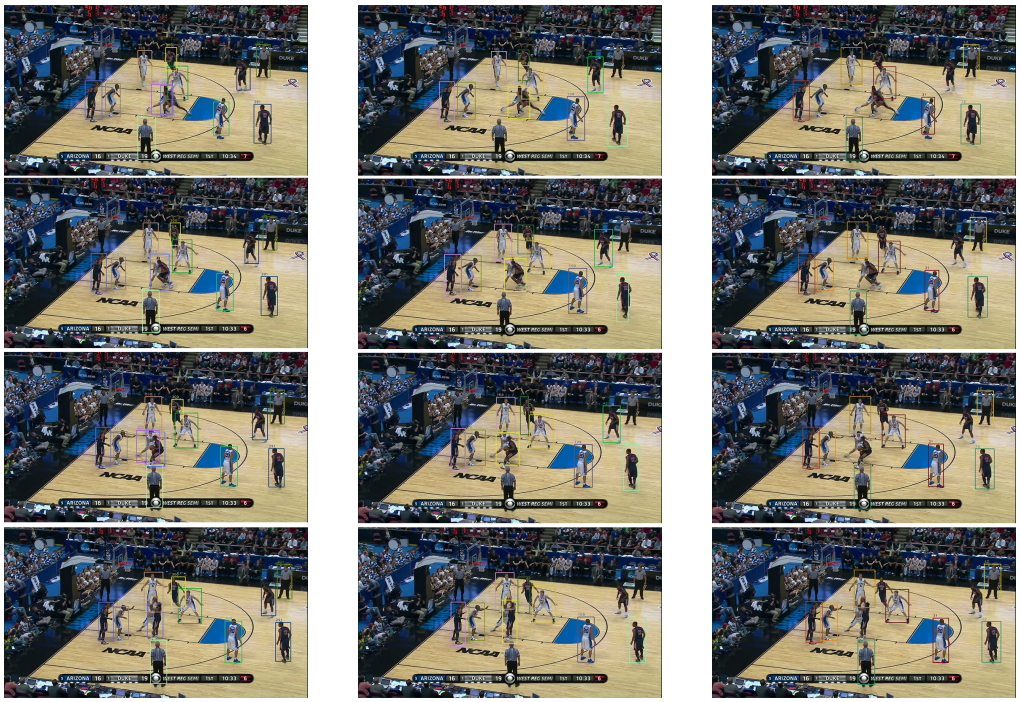
- as introduced in section 2.1, sports players tracking in general, and basketball players tracking in particular, is much more challenging than normal pedestrian tracking: this is due to the frequent and long occlusions, the close similarity between players, and their fast movement, complicated motion patterns and atypical poses;

- as mentioned in section 5.5, some bounding box correspondences are lost because of errors in the ground truth annotations of the dataset, while some false positives are still present because the referees are being detected along with the players; furthermore, it must be noted that the ground truths exist for all frames, but the tracking algorithms only initialise a track after a minimum number of detections are associated, so the detections on the first few frames are lost.

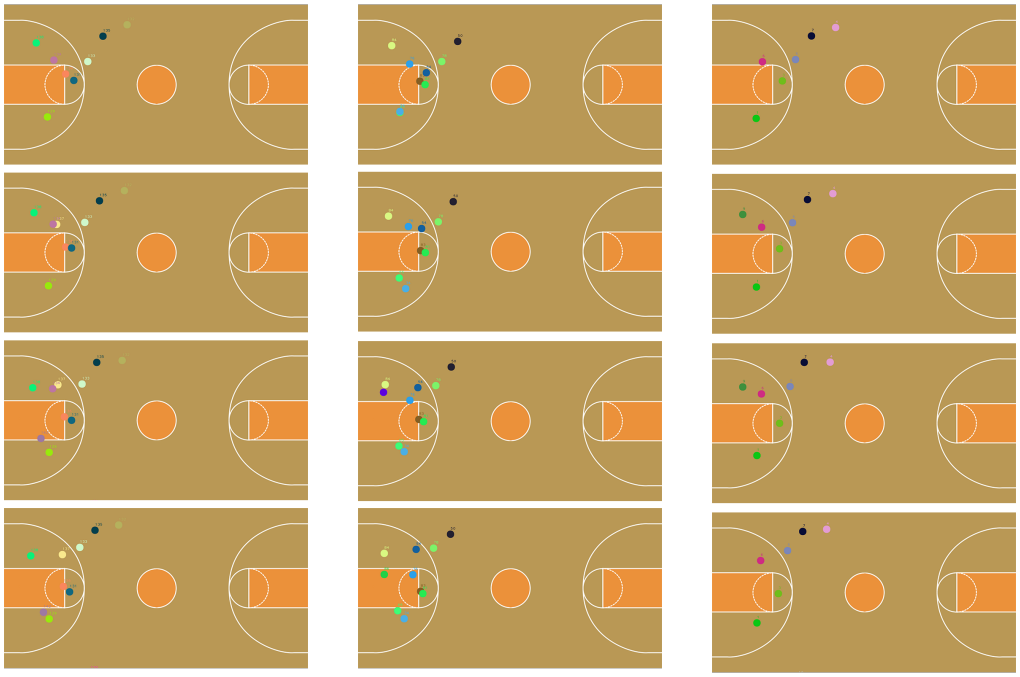
- the dataset is comprised of several subsets, each containing a set of frames extracted from a basketball game video: the tracking is thus applied to these sets of frames as if they were video clips. It is possible to observe that these frames have been extracted from their relative clips with a very low frame rate, which causes the players' positions and poses to change quite a lot from one frame to the next. This can rather hinder tracking algorithms, which often rely on the idea that an object will not move too much between consecutive frames to perform

track association —many algorithms use, among other things, IoU of bounding boxes between two frames to decide if they belong to the same track.

While not ideal, these results are good enough for the scope of this project. They will be improved upon in the future by means of further study and integration with other technologies, as will be explained in chapter 7



(a) Example of results on the video frames.



(b) Example of results on the 2D court model.

Figure 6.2: Examples of the player tracking results: SORT results are on the left, ground truths in the middle, DeepSORT results on the right.

Part III

Conclusions

Chapter 7

Conclusions and future work

Basketball player tracking presents many challenges with respect to common pedestrian tracking: the frequent and long occlusions, complicated motion patterns and unusual poses of the players, fast movement of both the players and the camera, and the ambiguous appearance of the players make recognising and tracking them a very difficult task.

This thesis proposes a *tracking-by-detection* algorithm that employs neural networks and computer vision to obtain the automatic detection and tracking of players on video clips of basketball games. Firstly, and open-source dataset containing basketball game clips and player tracking annotations was chosen and adapted to the project requirements. **Pedestron's Cascade Mask RCNN** trained on Caltech was chosen as the best option to detect the bounding boxes corresponding to people on each frame. A new semi-automatic algorithm was created to detect and track the basketball court, filter the bounding boxes, and project the players' position on a 2D court model. Finally two tracking methods were compared and **SORT**, a simple and fast tracking algorithm using a combination of Kalman filter and Hungarian algorithm, was applied to the detections, yielding discreet results.

Further work can be done on the developed system to overcome some of its limitations. One such limitation is the **processing time** of the algorithm: for both the player detection and the basketball court tracking time efficiency

was partly sacrificed in favour of accuracy and robustness. This choice was made because the scope of the project was to develop an off-line application to help analyse past games. If the need arises to convert the program for on-line use, e.g. for real-time analysis of an on-going game, a new equilibrium between time and accuracy must be found. A possible course of action in this case would be to change the detection network to a one-step CNN, since they tend to be quite faster than two-step CNNs —albeit slightly less accurate—, and to find a way to simplify the court tracking algorithm.

Another possible improvement to the project could be the addition of a **ball tracking** system which, coupled with the player tracking, would give an even broader overview of the game. Furthermore, an additional step could be added to **divide the players into teams**, possibly based on the dominant colors in their bounding boxes, which should reflect the color of their uniform; during this step the referees' detections could also be discarded based on their clothes color.

Another limitation is the player tracking algorithm. The **tracking results**, while acceptable for the project, would not be good enough for a commercial product. These results could be improved by studying and implementing a more complex tracking algorithm —keeping in mind, once again, the trade-off between time and accuracy— or by coupling the developed computer vision algorithm to other technologies. *ESTECO SpA* is, in fact, developing a proprietary system to physically track players through the use of **Ultra Wide Band (UWB)** hardware. This technology is able to provide very precise data on the position of objects and people in closed environments.

During the course of next year this project will continue thanks to a scholarship offered by Trieste's *Area Science Park*, and the developed algorithm will be integrated with the data offered by the UWB sensors, in the hopes of creating an accurate and robust application for sports games analysis.

Bibliography

[1] G. Thomas, R. Gade, T. B. Moeslund, P. Carr, and A. Hilton, "Computer vision for sports: Current applications and research topics," 2017.

[2] A. Arbués-Sangüesa, C. Ballester, and G. Haro, "Single-camera basketball tracker through pose and semantic feature fusion," 2019.

[3] W. Lu, J. Ting, J. J. Little, and K. P. Murphy, "Learning to track and identify players from broadcast sports videos," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 7, pp. 1704–1716, 2013.

[4] S. Parsons and J. D. Rogers, "Basketball player tracking and automated analysis," 2014.

[5] E. Cheshire, M.-C. Hu, and M.-H. Chang, "Player tracking and analysis of basketball plays," 2015.

[6] G. Zhu, C. Xu, Q. Huang, and W. Gao, "Automatic multi-player detection and tracking in broadcast sports video using support vector machine and particle filter," in *2006 IEEE International Conference on Multimedia and Expo*, pp. 1629–1632, 2006.

[7] Y. Yoon, H. Hwang, Y. Choi, M. Joo, H. Oh, I. Park, K. Lee, and J. Hwang, "Analyzing basketball movements and pass relationships using realtime object tracking techniques based on deep learning," *IEEE Access*, vol. 7, pp. 56564–56576, 2019.

[8] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," 2015.

[9] D. Acuna, "Towards real-time detection and tracking of basketball players using deep neural networks," 2017.

[10] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft, "Simple online and realtime tracking," *2016 IEEE International Conference on Image Processing (ICIP)*, Sep 2016.

[11] V. Ramanathan, J. Huang, S. Abu-El-Haija, A. Gorban, K. Murphy, and L. Fei-Fei, "Detecting events and key actors in multi-person videos,"

2015.

[12] G. Ciaparrone, F. Luque Sánchez, S. Tabik, L. Troiano, R. Tagliaferri, and F. Herrera, "Deep learning in video multi-object tracking: A survey," *Neurocomputing*, vol. 381, p. 61–88, Mar 2020.

[13] G. Van Rossum and F. L. Drake, *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009.

[14] Anaconda Software Distribution. https://anaconda.com.

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

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

[17] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pp. 265–283, 2016.

[18] Tzutalin, "Labelimg." https://github.com/tzutalin/labelImg, 2015.

[19] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár, "Microsoft coco: Common objects in context." https://cocodataset.org/, 2014.

[20] J. Brownlee, "A gentle introduction to object recognition with deep learning." https://machinelearningmastery.com/object-recognition-with-deep-learning/, 2019.

[21] N. Zeng, "An introduction to evaluation metrics for object detection." https://blog.zenggyu.com/en/post/2018-12-16/an-introduction-to-evaluation-metrics-for-object-detection/, 2019.

[22] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes challenge: A retrospective," *International Journal of Computer Vision*, vol. 111, pp. 98–136, Jan. 2015.

[23] K. Li, W. Ma, U. Sajid, Y. Wu, and G. Wang, "Object detection with convolutional neural networks," 2019.

[24] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," 2013.
[25] R. Girshick, "Fast r-cnn," 2015.
[26] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," 2015.
[27] Z. Cai and N. Vasconcelos, "Cascade r-cnn: Delving into high quality object detection," 2017.
[28] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn," 2017.
[29] K. Chen, J. Pang, J. Wang, Y. Xiong, X. Li, S. Sun, W. Feng, Z. Liu, J. Shi, W. Ouyang, C. C. Loy, and D. Lin, "Hybrid task cascade for instance segmentation," 2019.
[30] A. Sachan, "Zero to hero: Guide to object detection using deep learning: Faster r-cnn,yolo,ssd." https://cv-tricks.com/object-detection/faster-r-cnn-yolo-ssd/.
[31] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," *Lecture Notes in Computer Science*, p. 21–37, 2016.
[32] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," 2017.
[33] P. Dollár, C. Wojek, B. Schiele, and P. Perona, "Pedestrian detection: A benchmark," in *CVPR*, June 2009.
[34] S. Zhang, R. Benenson, and B. Schiele, "Citypersons: A diverse dataset for pedestrian detection," 2017.
[35] M. Braun, S. Krebs, F. B. Flohr, and D. M. Gavrila, "Eurocity persons: A novel benchmark for person detection in traffic scenes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1, 2019.
[36] S. Shao, Z. Zhao, B. Li, T. Xiao, G. Yu, X. Zhang, and J. Sun, "Crowdhuman: A benchmark for detecting human in a crowd," *arXiv preprint arXiv:1805.00123*, 2018.
[37] C. C. Loy, D. Lin, W. Ouyang, Y. Xiong, S. Yang, Q. Huang, D. Zhou, W. Xia, Q. Li, P. Luo, J. Yan, J. Wang, Z. Li, Y. Yuan, B. Li, S. Shao, G. Yu, F. Wei, X. Ming, D. Chen, S. Zhang, C. Chi, Z. Lei, S. Z. Li, H. Zhang, B. Ma, H. Chang, S. Shan, X. Chen, W. Liu, B. Zhou, H. Li, P. Cheng, T. Mei, A. Kukharenko, A. Vasenin, N. Sergievskiy, H. Yang, L. Li, Q. Xu, Y. Hong, L. Chen, M. Sun, Y. Mao, S. Luo, Y. Li, R. Wang, Q. Xie, Z. Wu, L. Lu, Y. Liu, and W. Zhou, "Wider face and pedestrian challenge 2018: Methods and results," 2019.
[38] I. Hasan, S. Liao, J. Li, S. U. Akram, and L. Shao, "Pedestrian detection: The elephant in the room," *arXiv preprint arXiv:2003.08799*, 2020.

[39] Y. Pang, J. Xie, M. H. Khan, R. M. Anwer, F. S. Khan, and L. Shao, "Mask-guided attention network for occluded pedestrian detection," 2019.

[40] A. Kuznetsova, H. Rom, N. Alldrin, J. Uijlings, I. Krasin, J. Pont-Tuset, S. Kamali, S. Popov, M. Malloci, A. Kolesnikov, T. Duerig, and V. Ferrari, "The open images dataset v4: Unified image classification, object detection, and visual relationship detection at scale," *IJCV*, 2020.

[41] "Transfer learning and fine-tuning." https://www.tensorflow.org/tutorials/images/transfer_learning.

[42] P. Wen, W. Cheng, Y. Wang, H. Chu, N. C. Tang, and H. M. Liao, "Court reconstruction for camera calibration in broadcast basketball videos," *IEEE Transactions on Visualization and Computer Graphics*, vol. 22, no. 5, pp. 1517–1526, 2016.

[43] J. Canny, "A computational approach to edge detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-8, no. 6, pp. 679–698, 1986.

[44] Z. Zhang, "Iterative point matching for registration of free-form curves and surfaces," *International journal of computer vision*, vol. 13, no. 2, pp. 119–152, 1994.

[45] A. Gupta, J. J. Little, and R. J. Woodham, "Using line and ellipse features for rectification of broadcast hockey video," in *2011 Canadian Conference on Computer and Robot Vision*, pp. 32–39, 2011.

[46] R. Hess and A. Fern, "Improved video registration using non-distinctive local image features," in *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–8, 2007.

[47] R. Pourreza, M. Khademi, H. Pourreza, and H. R. Mashhadi, "Robust camera calibration of soccer video using genetic algorithm," in *2008 4th International Conference on Intelligent Computer Communication and Processing*, pp. 123–127, 2008.

[48] T. Fu, H. Chen, C. Chou, W. Tsai, and S. Lee, "Screen-strategy analysis in broadcast basketball video using player tracking," in *2011 Visual Communications and Image Processing (VCIP)*, pp. 1–4, 2011.

[49] P. V. Hough, "Method and means for recognizing complex patterns," 12 1962.

[50] J.-B. Hayet and J. Piater, "On-line rectification of sport sequences with moving cameras," in *Mexican International Conference on Artificial Intelligence*, pp. 736–746, Springer, 2007.

[51] D. Farin, S. Krabbe, W. Effelsberg, *et al.*, "Robust camera calibration for sport videos using court models," in *Storage and Retrieval Methods and*

Applications for Multimedia 2004, vol. 5307, pp. 80–91, International Society for Optics and Photonics, 2003.

[52] Yang Liu, Shuqiang Jiang, Qixiang Ye, Wen Gao, and Qingming Huang, "Playfield detection using adaptive gmm and its application," in *Proceedings. (ICASSP '05). IEEE International Conference on Acoustics, Speech, and Signal Processing, 2005.*, vol. 2, pp. ii/421–ii/424 Vol. 2, 2005.

[53] "Opencv-python tutorials." https://opencv-python-tutroals.readthedocs.io/en/latest/index.html.

[54] M. A. Fischler and R. C. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Commun. ACM*, vol. 24, p. 381–395, June 1981.

[55] E. Bochinski, V. Eiselein, and T. Sikora, "High-speed tracking-by-detection without using image information," in *2017 14th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, pp. 1–6, 2017.

[56] N. Wojke, A. Bewley, and D. Paulus, "Simple online and realtime tracking with a deep association metric," 2017.

[57] C. Kim, F. Li, A. Ciptadi, and J. M. Rehg, "Multiple hypothesis tracking revisited," in *2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 4696–4704, 2015.

[58] F. Yu, W. Li, Q. Li, Y. Liu, X. Shi, and J. Yan, "Poi: Multiple object tracking with high performance detection and appearance feature," 2016.

[59] D. Chicco, "Siamese neural networks: An overview," in *Artificial Neural Networks*, pp. 73–94, Springer.

[60] M. Kim, S. Alletto, and L. Rigazio, "Similarity mapping with enhanced siamese network for multi-object tracking," 2016.

[61] S. Zhang, Y. Gong, J.-B. Huang, J. Lim, J. Wang, N. Ahuja, and M.-H. Yang, "Tracking persons-of-interest via adaptive discriminative features," vol. 9909, 08 2016.

[62] J. Son, M. Baek, M. Cho, and B. Han, "Multi-object tracking with quadruplet convolutional neural networks," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3786–3795, 2017.

[63] A. Sadeghian, A. Alahi, and S. Savarese, "Tracking the untrackable: Learning to track multiple cues with long-term dependencies," 2017.

[64] M. Babaee, Z. Li, and G. Rigoll, "Occlusion handling in tracking multiple people using rnn," in *2018 25th IEEE International Conference on Image Processing (ICIP)*, pp. 2715–2719, 2018.

[65] S. Karthik, A. Prabhu, and V. Gandhi, "Simple unsupervised multi-object tracking," 2020.

[66] W. Ren, X. Wang, J. Tian, Y. Tang, and A. B. Chan, "Tracking-by-counting: Using network flows on crowd density maps for tracking multiple targets," 2020.

[67] Y. Zhang, C. Wang, X. Wang, W. Zeng, and W. Liu, "A simple baseline for multi-object tracking," 2020.

[68] E. Bochinski, T. Senst, and T. Sikora, "Extending iou based multi-object tracking by visual information," in *2018 15th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, pp. 1–6, 2018.

[69] P. Bergmann, T. Meinhardt, and L. Leal-Taixe, "Tracking without bells and whistles," 2019.

[70] Bo Wu and R. Nevatia, "Tracking of multiple, partially occluded humans based on static body part detection," in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, vol. 1, pp. 951–958, 2006.

[71] K. Bernardin and R. Stiefelhagen, "Evaluating multiple object tracking performance: the clear mot metrics," *EURASIP Journal on Image and Video Processing*, vol. 2008, pp. 1–10, 2008.

[72] E. Ristani, F. Solera, R. Zou, R. Cucchiara, and C. Tomasi, "Performance measures and a data set for multi-target, multi-camera tracking," in *European Conference on Computer Vision*, pp. 17–35, Springer, 2016.

[73] C. Heindl, "Benchmark multiple object trackers in python." https://github.com/cheind/py-motmetrics.