UCL Université catholique de Louvain

École polytechnique de Louvain (EPL)

epl

Center of attention prediction for automatic video framing in teamsport events

Dissertation presented by
Vladimir SOMERS

for obtaining the Master's degree in
Computer Science and Engineering

Supervisor(s)
Christophe DE VLEESCHOUWER

Reader(s)
Pascaline PARISOT, Damien DELANNAY , Laurent JACQUES

Academic year 2015-2016

Acknowledgements

Firstly, I would like to express my sincere gratitude to my thesis supervisor, Pr. Christophe de Vleeshouwer, for the numerous long meetings we have made throughout the year and for his patience, motivation and continuous guidance.

I would like to thank Pascaline Parisot, for the innumerable hours spent to help me install and understand all Keemotion's tools.

I would also like to thank Damien Delannay and Keemotion for giving me the opportunity to conduct my thesis on this concrete and fascinating subject.

Finally, I would like to express my profound gratitude to my girlfriend Coline and to my parents for providing me with unfailing support and continuous encouragement throughout the process of researching and writing this thesis.

Abstract

Keemotion is a spin-off from Université Catholique de Louvain proposing automated sport production. One of the key functionality of their product is the automated video framing of the filmed game: based on the sequences recorded by a fixed camera above the court, an algorithm is responsible for planning where the camera should look. The current solution implemented by Keemotion is based on a set of fixed rules and heuristics and, although it is rather effective for most situations, it showed some weaknesses in particular tricky game phases. In this work, we propose a method to predict an important information for the framing: the Center of Attention, which could be used to improve current Keemotion's system or implement a new framing strategy. The problem will thus be formulated as a supervised learning task: using a set of framing examples manually annotated and the associated visual characteristics extracted automatically from the scene, we train a regression function which will be able to predict the Center of Attention in new video sequences.

Table of contents

Introduction **1**

1 Problem formulation and data collection **3**
1.1 Keemotion's System . . . 3
1.1.1 Players detections . . . 4
1.1.2 Framing strategy . . . 5
1.2 Keemotion's Framing Strategy weaknesses . . . 5
1.2.1 Game phases categories . . . 6
1.2.2 The Center of Attention . . . 8
1.3 The data driven approach . . . 9
1.3.1 Related works . . . 9
1.3.2 Data collection . . . 11
1.3.3 The Detection Dataset . . . 12
1.3.4 The Annotation Dataset . . . 13

2 Feature Extraction **15**
2.1 Problems with the current dataset . . . 16
2.2 Detections filtering . . . 17
2.3 The Activity map . . . 18
2.4 The Velocity map . . . 20
2.4.1 The Hungarian Algorithm . . . 21
2.4.2 Averaging over a few frames . . . 24
2.4.3 Cells velocities . . . 27
2.5 From detections to players positions . . . 27
2.5.1 Hough Transform . . . 30
2.5.2 Detections clustering . . . 31
2.5.3 Players positions . . . 33
2.6 The Feature Map . . . 35

3 Learning **39**
3.1 Support Vector Regression 40
3.2 Multilayer Perceptron 41
3.3 Random Forest 42
3.4 Variable scaling 42
3.5 10-fold cross validation 43

4 Results and Validation **45**
4.1 Video sequences selection 45
4.2 Feature extraction process performance 47
4.2.1 Hough Transform and Filtering 47
4.2.2 Activity Map 49
4.2.3 Velocity Map 50
4.3 The Feature Dataset in numbers 50
4.4 Parameters Tuning 51
4.5 Results and Perspectives 54

Conclusion **57**

References

Introduction

Keemotion is a spin-off from Université Catholique de Louvain proposing automated sport production. Their core activity is focused on basketball. One of the key functionality of their product is the automated video framing of the filmed game: based on the sequences recorded by a fixed camera above the court, an algorithm is responsible for planning where the camera should look. This is a really challenging task as it requires the algorithm to be able to detect the region of interest at a given moment in time during the game. The current solution implemented by Keemotion is based on a set of fixed rules and heuristics and, although it is rather effective for most games phases, it showed some weaknesses in particular tricky game phases such as throw-in and counter attacks. Keemotion is therefore looking for another more general framing strategy which could handle any edge cases and be easily adapted to other sports such as ice hockey or soccer. The strategy proposed in this work is based on supervised learning and should be used, in the first instance, to improve the current strategy. This approach could however be extended to implement a brand new framing strategy.

As the goal of this master thesis is to improve the framing strategy currently used by Keemotion, we will focus our research on tricky game phases, for which the current approach gives poor results. We will therefore try to extract a new information from the recorded game, which is vital for a good framing: the Center Of Attention. The position of this Center Of Attention on the court will be predicted using a data-driven approach. The problem will thus be formulated as a supervised learning task: using a set of framing examples manually annotated and the associated visual characteristics extracted automatically from the scene, we train a regression function which will be able to predict the Center of Attention in new video sequences. Although this approach should be easily adapted to other team sports, the current work will mainly focus on basketball.

This paper is organized in four parts. In the first Chapter, we will review Keemotion's system, explain how we formulate our problem as a supervised learning and see

how to collect the data necessary for this approach. In the second Chapter, we will describe the feature extraction process used to transform raw data into meaningful feature vectors. In the third chapter, we will review three famous machine learning methods which will be used to train a regressor and predict the Center of Attention. In the last Chapter, we will provide some qualitative assessment of the feature extraction process and examine the performance of the three machine learning algorithms.

Chapter 1

Problem formulation and data collection

In this chapter, we will start to describe the system currently used by Keemotion and analyze the performance of their planning algorithm. A new method to improve Keemotion's algorithm will then be proposed. This method will be based on the prediction of a point on the court called 'Center of Attention'. After a quick revision of the state-of-the-art autonomous camera systems, we will describe how we formulate our problem of finding the Center of Attention in a basketball game sequence as a supervised learning task. We will then see how we gathered the data that were used to solve the problem.

1.1 Keemotion's System

The system currently used by Keemotion is composed of two fixed cameras positioned side by side above the long court edge. They provide two lateral views of the court, one for each side, which are combined into a single image of the whole basketball court. From now on, we will refer to these two cameras as a unique one. A first algorithm is applied to detect salient objects in the large image. This algorithm is described in Section 1.1.1. A second algorithm is then applied to automatically zoom on the relevant part of the image, producing the final image seen by the user. This algorithm is described in Section 1.1.2.

1.1.1 Players detections

The algorithm used by Keemotion to detect players in a video frame works as follow. Firstly, a foreground mask is created using a Background Subtraction technique. This technique is illustrated in Figure 1.1[1].

Background Substraction is usually performed by subtracting an image of the background of the scene to the analyzed image. A threshold is then applied on the resulting image such that a big pixels difference gives a white pixel in the foreground mask and a small pixels difference gives a black pixel in the foreground mask. This foreground mask is therefore a black and white image where the white parts correspond to the salient objects and the black parts corresponds to the background.

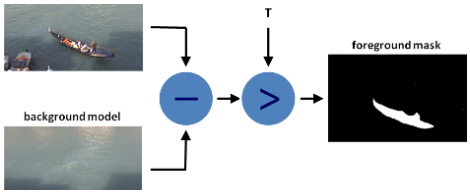
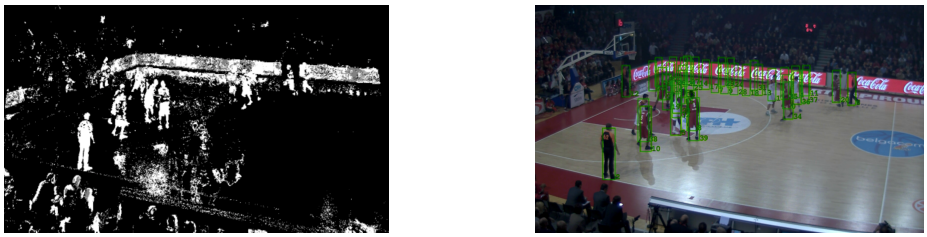


Fig. 1.1 A toy example of Background Subtraction.

A *salient object* is defined here as an element that differ from the background. Some of the salients objects in the foreground mask correspond therefore to interesting elements for the game such as the ball or the players. However, some other salient objects correspond to elements that are less interesting such as the referees and the coaches or even to uninteresting elements such as the reflection of a lamp on the court or the light coming from a digital signboards.

The second step is performed as follows. A virtual box with approximately the size of a player is centered on each white pixel of the foreground mask. For a given white pixel $p_{ik} = [u_{i,k},\ v_{i,k}]$ at frame f_i, where i is the index of the frame in the video and k is an index given to each white pixel, if the ratio r_{ik} between the amount of white and black pixels inside the box is greater than a given threshold, then there's probably a player in this part of the image. We will therefore record the coordinate $[u_{i,k},\ v_{i,k}]$ of the pixel as a probable player's position in the image, i.e. a *detection* $d_{i,k} = [u_{i,k},\ v_{i,k}]$. The set $D_i = \{d_{i,1}, d_{i,2}, \ldots, d_{i,n}\}$ of the n best detections recorded at frame f_i is the final output of this algorithm. Figures 1.2, which was found in [14], shows an example of the two steps described above.

[1] http://docs.opencv.org/master/d1/dc5/tutorial_background_subtraction.html#gsc.tab=0



(a) (b)

Fig. 1.2 An example of foreground mask (a) and the corresponding detections and boxes (b).

1.1.2 Framing strategy

The algorithm used to crop the image follows a fixed rule, which can be summarized as a trade-off between two major factors: *Completeness* and *Closeness*. As described in [7], Completeness stands for the "*integrity of view rendering*". From a more formal point of view, it can be seen as the "*percentage of salient objects included in the image*". In our case, the salient objects are the detections obtained using the algorithm presented in the last Section. The second factor, the Closeness, defines "*the fineness of detail description*", i.e. the sensation to be close to the objects in the image. From a more formal point of view, it can be seen as "*the average pixel area used for rendering salient objects*". The idea behind this algorithm is therefore to include as much players as possible in the cropped image while maintaining a view as close as possible to the players. Figure 1.3 gives two examples of such a framing strategy. The algorithm's performance is therefore strongly related to the quality of the players detections.



(a) A wide view (b) A close view

Fig. 1.3 Some examples of image cropping using Keemotion's algorithm.

1.2 Keemotion's Framing Strategy weaknesses

In practice, the framing strategy implemented by Keemotion is quite effective: most game phases involve all players standing on a particular bounded playing area of the

court and the algorithm can therefore easily include all of them in the framing. There are however some situations during the game where this strategy gives poor results. These are mostly situations where the *Center of Attention*[2] is far from the *Center of Mass* of the group of players.

Definition 1.2.1. The *Players Center of Mass*, or centroid, is defined as the equally weighted average of all players positions on the court.

In fact, it is possible to observe 7 categories of game phases which are badly framed by the Keemotion's algorithm. These categories are listed in the next Section. A little description is given for each of them, together with a visual example and a discussion about a possible improvement to the framing.

1.2.1 Game phases categories

On Figure 1.4, you can see that the framing is not perfect. We are in an attack situation and the view should be a little bit closer to the players under the basketball hoop, because the player on the right of the image and the referee at the bottom are not really important for the action.



Fig. 1.4 An attack phase.



Fig. 1.5 A counter attack

On Figure 1.5, we have a counter attack, where the action has quickly moved from the left to the right part of the basketball court. Here, only the players taking part in this counter attack should be kept in the framing. In counter attack situations situations, we should therefore put more weight forward, i.e. on the right in this example.

[2]We will define this term in much more details in a subsequent chapter.

On Figure 1.6, you can see another very badly framed example. The Algorithm chose a wide view although there are no player on the right part of the court. This situation occurs sometimes because of *false player detections*. As described in Section 1.1.1, these false detections can be caused by a referee, a coach or anybody who's not a player. There can be also caused by the reflection of a lamp on the court or by the digital signboards.



Fig. 1.6 False detections



Fig. 1.7 A Transition phase

Figure 1.7 shows a game phase which occurs quite often. This is a transition phase where ball possession just changed from one team to the other one. All players goes therefore from one side of the court to the other side. In most cases, the ball carrier is at the back of the group of players and the framing is wide (sometimes, the ball carrier is even not included in the image). We should therefore try to move the framing a little bit more to the back of this group of players.

Another important basketball phase is the *jump ball*, which is a method used to begin or resume play. Figure 1.8 shows an example of such a phase, occurring at the start of the game. The Center of Attention is clearly between the ball and the two players which attempt to gain control of it. In these situations, it would be interesting to have a closer view to the action, in order to enjoy as much as possible the confrontation between the two players.



Fig. 1.8 A jump ball



Fig. 1.9 A Throw-In

Figure 1.9 shows a Throw-in phase, where only a few players participate to the action: the players around the ball carrier. This phase is badly framed because we don't see the ball carrier, although the other players on the left of the court, which are not participating to the action, are included in the image.

Figure 1.10 shows a Throw-In, which is a very precise situation regarding the positions of the elements we want to include in our image. These elements are the player throwing the ball and the basketball hoop. An ideal framing should therefore only include these two elements, giving a much more accurate view of the action.



Fig. 1.10 A Free-Throw

The categories described above were created subjectively in order to cluster and label tricky game phases with similar properties in the final dataset. This labeling will be useful to evaluate the performance of a new strategy in a particular situation.

1.2.2 The Center of Attention

In order to improve the framing strategy for the tricky game phases described above, we need to provide a new information to Keemotion's algorithm: The Center of Attention.

Definition 1.2.2. The *Center of Attention* is the region of interest on the basketball court at a given point in time during the game. It's generally the area on which the the audience's gaze is focused. The Center of Attention is formally defined as a 2D coordinate on the basketball court. We will often use the shortcut COA to refer to it.

In almost every situation, the Center of Attention is the position of the ball. Figure 1.11 shows the COA for two phases described in Section 1.2.1.

The goal of this master thesis is to find a method to predict the Center of Attention in a given video frame. The approach used will be explained in the next Section.

Once the Center of Attention is infered in a given video frame, it should be used as an additional information to improve Keemotion's framing strategy. However, this



(a) ROI in a Trow-in phase (b) ROI in a free throw phase

Fig. 1.11 Some examples of Center of Attention, indicated by a green cross on the image.

thesis will only focus on the method used to infer the COA. We will not explore strategies to incorporate this new information in Keemotion's algorithm.

1.3 The data driven approach

The problem of finding the Center of Attention in the image has been formulated as the optimization of a regression function. This regression function will be trained on several feature vectors to infer the position of the COA. These feature vectors will be constructed using characteristics extracted automatically from the scene with Keemotion's tools and then be manually annotate with a COA. From now on, we will refer to the dataset containing all these processed feature vectors as the *Feature Dataset*, to the dataset of automatically extracted characteristics as the *Detection Dataset* and to the dataset of manual annotations as the *Annotation Dataset*. The exact nature of these processed feature vectors and the way we compute them will be explained in Chapter 2. A more precise formulation of the problem as a regression function will be given in Chapter 3.

In this section, we will start to describe some state-of-the-art works done on the subject and give a brief comparison with this thesis. We will then explain how the Detection Dataset and the Annotation Dataset were created and give a more formal description of the information they contain.

1.3.1 Related works

Autonomous cameras systems are commonly used to broadcast live events such as cooking shows, sport events or lectures. Paper [5] presents a review of the autonomous camera systems developed over the past twenty years. These systems involves generally

three main tasks: camera controlling, camera selection and camera planning. The problem we are interested in is the camera planning, which aims at determining where the camera should look. The exact parameters to be determined by the planning algorithm depends on the system used. It can be, for example, a PTZ camera, where the output parameters are a pan angle, a tilt angle and a zoom factor. In Keemotion's case, we have to define a virtual cropping of a large fixed image of the basketball court. A lot of different method have been used to address planning, and each of them depends on the *context* and the *information* available.

The *context* refers to the type of event recorded (lectures, cooking shows, sport events) and the timing constraint: some system must work in an online, realtime framework while other systems work in an offline fashion and can therefore address planning at a given point in time by taking into account past and future events.

The *information* available vary also from a system to another. Most of them use low level data, such as object detection or tracking, collected using computer vision algorithms. More recent strategies use also higher level information such as game state, action recognition and audience gaze direction for sport events or contextual information of the script for cooking shows. This higher level information can be provided by a human expert or be extracted automatically using advanced methods such as machine learning.

Most of these systems use hand-crafted rules and heuristics to infer planning parameters and, according to [6], machine learning method are generally used as features extraction technique and not to formulate the camera planning directly as a supervised learning problem. An example of this approach is [11], where players tracking data where used to build a global motion vector field on the ground and camera planning was inferred using regions of convergence in the vector field.

Among all the works that have been done on planning for team sport and particularly for basketball, [6] has the most similar approach with this thesis. According to the authors, they were the first to use a data driven approach to learn the relationship between players locations and planning parameters.

In this work, they first built feature vectors using noisy tracking data. Three different features representation were proposed. The first one is simply the players centroid. The second is a heat map, which is a grid on the basketball court recording the number of players in each cell using soft binning. They try different resolution for this heat map : 2×1, 4×2 and 6×3 cells. The last one is a spherical representation of the heat map, with only one dimension. The heat map and spherical map can be seen on Figure 1.12c and 1.12d. The labeling of these feature vectors was done using

an exemplar camera planning performed by a human expert. The parameters of the PTZ camera for this exemplar work were estimated automatically via video analysis.

Finally, the supervised learning task of predicting camera parameters from the features vectors was expressed as a regression, where the output is only the pan angle of the PTZ camera. They explored three machine learning algorithms : *Linear Least Squares Regression*, *Support Vector Regression* and *Random Forest Regression*.

In this thesis, we will use a feature extraction strategy similar to the heat map presented in this paper. We will however add other features, in order to provide more sensing information to the machine learning algorithm. Another major difference between [6] and this thesis is the situations analyzed. In [6], the authors train their regressor using one continuous video of a basketball game. The video contains therefore mostly "normal" situations, unlike our dataset which will contain mostly tricky situations. Another difference is the tracking data used. The algorithm used by the authors to generate players detections seems to produce more reliable and less noisy detections than Keemotion's algorithm. The other difference is the parameter predicted: they use the angle of the camera and we will use the x-component of the center of attention on the court. The last difference is the manual annotations. In their case, a human expert camera operator controlled the camera to generate the labeling of their video sequence. In our case, the center of attention was manually annotated on the image using a simple *SDL* interface.

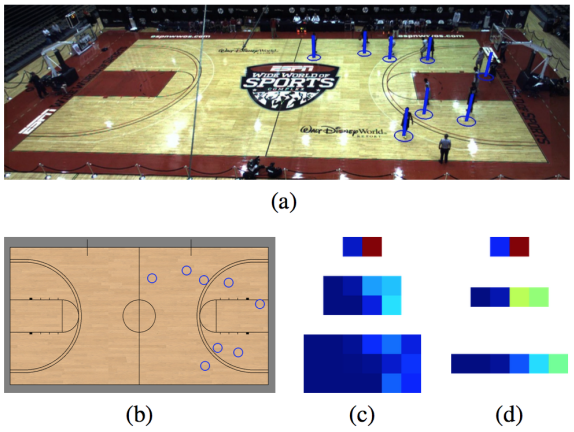


Fig. 1.12 Feature vector described in [6].

1.3.2 Data collection

We have formulated the problem of finding the Center of Attention in the image as a Supervised Learning task. Supervised Learning[3] is the machine learning task of inferring a function from labeled training data. The training data consist of a set of training examples. In our case, each training example is composed of the data

[3]https://en.wikipedia.org/wiki/Supervised_learning

collected at one video frame, i.e. at one specific point in time during the basketball game. In supervised learning, each of these examples must be a pair consisting of an input object (a feature vector) and the desired output value. As explained before, we call this labeled training data the Feature Dataset. In order to create it, we will first need to build two other datasets:

1. The **Detection Dataset**, which is composed of the information automatically extracted from each video frame with Keemotion's algorithm. The data contained in this Detection Dataset is described in Section 1.3.3.

2. The **Annotation Dataset**, which is composed of the information manually annotated at each dataframe. The manual annotation process is described in Section 1.3.4.

The Detection Dataset and Annotation Dataset were build using the same set of video sequences. These videos sequences were provided by Keemotion and were recorded in 3 different stadiums across Europe. It may seem that these two dataset could be combined into a unique one. The reason of this separation will be explained later.

1.3.3 The Detection Dataset

In order to build the Detection Dataset, we will gather data that can be extracted automatically from the scene and which is suspected to contain evidence about the Center Of Attention location. The only data available for the time being are the players detections. These players detections were extracted automatically from the videos using the algorithm described in Section 1.1.1. Of course, some other data could be useful to infer the COA. We will discuss that point further in Chapter 4.5.

As explained in Section 1.1.1, Keemotion's detection tool provide a set of detections $D_i = \{d_{i,1},\ d_{i,2},\ \ldots,\ d_{i,n}\}$ for each video frame f_i. Each detection $d_{i,k}$ is a position $[u_{i,k},\ v_{i,k}]$ in the image coordinate system. However, it is not convenient to work in this coordinate system, each position $[u_{ik},\ v_{ik}]$ is therefore projected onto the court, giving a new position $(x_{i,k},\ y_{i,k})$ expressed in a more general coordinate system. Each time we refer to a detection $d_{i,k}$, we therefore refer to the tuple:

$$d_{i,k} = (x_{i,k}, y_{i,k}) \tag{1.1}$$

, where $(x_{i,k},\ y_{i,k})$ is the coordinate of the detection on the court coordinate system.

These detections are also grouped into sets of detections $D_i = \{d_{i,1},\ d_{i,2},\ \dots,\ d_{i,n}\}$ recorded at the same video frame. As a reminder, i is defined as the index of the frame f_i in the video. The timestamp[4] of frame f_i is referred to as t_i. From now on, we will use the term **frame** both to refer to a video frame f_i and to the set of detections D_i.

Assuming we build our dataset using m video frames, the m sets of detections together with their respective timestamps forms the Detections Dataset DD:

$$DD = \begin{bmatrix} t_1 & D_1 \\ t_2 & D_2 \\ \vdots & \vdots \\ t_m & D_m \end{bmatrix} = \begin{bmatrix} t_1 & d_{1,1} & \dots & d_{1,n} \\ t_2 & d_{2,1} & \dots & d_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ t_m & d_{m,1} & \dots & d_{m,n} \end{bmatrix} = \begin{bmatrix} t_1 & (x_{1,1}, y_{1,1}) & \dots & (x_{1,n}, y_{1,n}) \\ t_2 & (x_{2,1}, y_{2,1}) & \dots & (x_{2,n}, y_{2,n}) \\ \vdots & \vdots & \ddots & \vdots \\ t_m & (x_{m,1}, y_{m,1}) & \dots & (x_{m,n}, y_{m,n}) \end{bmatrix} \tag{1.2}$$

For a given video sequence, there is a fixed number n of detections at each frame. This number may however vary from one stadium to another. For example, in the stadium Wels, $n = 100$. For Bologna, $n = 80$.

The time step between two frames is generally a few hundredth of a second but the exact number changes from one stadium to another.

1.3.4 The Annotation Dataset

As its name suggests, the Annotation Dataset is composed of the manually annotated information. As explained earlier, in supervised learning, each of the training examples must be labeled with the desired output value. In our case, this output value is the Center Of Attention[5], which is a 2D coordinate on the basketball court. The labeling was done using a tool implemented in C on top of Keemotion's SDL viewer. Given a frame of the video sequence, we manually indicate the COA as a 2D point in the image. This point is then projected onto the basketball court coordinate system using a transformation matrix provided by Keemotion. These matrices are automatically generated for each frame thanks to a camera calibration tool. This transformation step is performed because we need the learning of the COA to be general and independent of the camera position.

In order to build a complete dataset, another position expressed in the court coordinate system wad added. This second coordinate indicates a region of the court which is not the visual Center of Attention but which is still interesting to include in

[4]The time at which the frame was recorded.
[5]The COA was defined in Section 1.2.2.

the framing because it allows to have a more complete view and therefore a better understanding of the action being played. From now on, we will refer to this point as the Secondary Point Of Attention SCOA.

Regarding the majors two factors involved in a good framing, which have been defined in Section 1.1.2, we can therefore see the COA as an indication to achieve *Closeness* and the SCOA as an indication to achieve *Completeness*. The SCOA was finally not used in this thesis, but it's still included in the datasets and could be useful for future work. Figure 1.13 shows an example of these two annotated points. The situation depicted here is a counter attack. The Center of Attention is therefore the ball carrier, but we also want to include in the framing the players at the forward of the counter attack, because they are important to understand the action and they will very probably take part in the action in the near future.

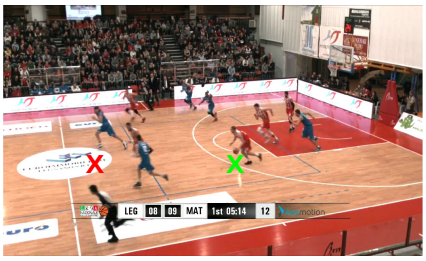


Fig. 1.13 The COA, indicated with a green cross and the SCOA, indicated with a red cross.

The timestamps of the l annotated frames together with their two interest points form the Annotation Dataset AD:

$$AD = \begin{bmatrix} t_1 & COA_1 & SCOA_1 \\ t_2 & COA_2 & SCOA_2 \\ \vdots & \vdots & \vdots \\ t_l & COA_l & SCOA_l \end{bmatrix} = \begin{bmatrix} t_1 & (x_1^{COA}, y_1^{COA}) & (x_1^{SCOA}, y_1^{SCOA}) \\ t_2 & (x_2^{COA}, y_2^{COA}) & (x_2^{SCOA}, y_2^{SCOA}) \\ \vdots & \vdots & \vdots \\ t_l & (x_l^{COA}, y_l^{COA}) & (x_l^{SCOA}, y_l^{SCOA}) \end{bmatrix} \tag{1.3}$$

Time interval

The time between two successive dataset examples here is not the same as the one we had for the Detection Dataset DD. In fact, only a subset of the examples of the DD were manually annotated with the Center of Attention: a time interval of half a second was chosen between two annotated frames. They are two reasons for this: first, the COA is not very different for two successive frames, because the time has only progressed from a few hundredth of a second. Annotating two successive frames is therefore useless because we do not collect new information. Secondly, the annotation process is done manually and is therefore very slow. We should therefore try to avoid useless work and only annotate a frame if it brings new information.

Chapter 2

Feature Extraction

In the previous chapter, we have seen how the Detectios Dataset DD and the Annotation Dataset AD were collected using video sequences and detections provided by Keemotion. As a reminder[1], the Detection Dataset is a set of m examples, where each example is composed of a timestamp and a set of n detections, where n varies from one stadium to another.

$$DD = \begin{bmatrix} t_1 & (x_{1,1}, y_{1,1}) & \dots & (x_{1,n}, y_{1,n}) \\ t_2 & (x_{2,1}, y_{2,1}) & \dots & (x_{2,n}, y_{2,n}) \\ \vdots & \vdots & \ddots & \vdots \\ t_m & (x_{m,1}, y_{m,1}) & \dots & (x_{m,n}, y_{m,n}) \end{bmatrix} \tag{2.1}$$

The Detection Dataset is however too coarse to be used directly as an input to a regression algorithm. We must therefore process it in order to build a fixed size, smooth and meaningful Feature Dataset. This Features Dataset is a set of feature vectors, one for each example of the Annotations Dataset, which will be used as examples to train the regression algorithm. In this chapter, we will start to analyze the Detection Dataset and its weaknesses. We will then see how to transform each set of detections in DD into an Activity Map and a Velocity Map. We will also see how the Detection Dataset can be preprocessed to provide a less noisy and much more accurate information. We will finally see how the Activity and Velocity Maps are combined to form the feature vectors of the Feature Dataset.

[1] A much more detailed description can be found in Section 1.3.2

2.1 Problems with the current dataset

One of the first problem we can observe with the DetectionsDataset is the huge amount of false detections. Unfortunately, there is no optimal strategy to deal with these false detections. We will however try to do some filtering to remove them. The strategy used will be explained in Section 2.2.

Another big issue with the Detection Dataset is the varying number of detections. There are two main reasons for this varying number of detections. First of all, the system installed by Keemotion is parametrized differently from one stadium to another, which means that the number of detections provided at each frame changes also. Secondly, the filtering strategy presented in the next Section will remove a varying number of detections at each frame. A varying number of detections is a problem because, in order to apply some regression algorithm on the dataset, we need a fixed number of features per example. We will therefore use a different representation of the detections to solve this problem: the Activity Map. This representation is explained in the Section 2.3.

The Detection Dataset is also a poor representation of the players distribution on the court because it is a represention with a lot of symmetry. The reason is that there's no specific order in a set of detections D_i, so if we use the n detections $\{d_{i,1}, d_{i,2}, \dots, d_{i,n}\}$ (with $d_{i,j} = (x_{i,j}, y_{i,j})$) as a $2 \times n$ sized feature vector, there is a total of $n!$ different feature vectors representing the exact same players distribution on the court. The new representation we will present in Section 2.3 has the advantage to transform these $n!$ different detections combinations into the same feature vector.

Another issue with the current dataset is the varying number of detections per player. Some players can produce up to eight detections on the court, while other players produce only one detection. Players with several detections may therefore have bigger impact in the learning process comparing to the others, which is an issue because all players should have the same weight when trying to predict the COA. We will therefore try to solve this problem by using only one detection per player. The method used to achieve this is based on the Hough Transform and is explained in Section 2.5.

The last big problem with each example of the Detections Dataset is the lack of information about the past: we don't know how the player evolves in a more temporal context. This temporal data is important because it allows us to represent the motions of the players on the court and therefore to add an information containing many clues regarding the Center of Attention position. The strategy used to add this temporal information is the computation of player's velocities using successive frames. These

velocities are then combined into a Velocity Map. We will detail this approach in Section 2.4.

2.2 Detections filtering

The main drawback of the algorithm used by Keemotion[2] to detect the players is the huge amount of noisy detections[3]. As these noisy detections do not give information regarding the COA we are trying to infer, they should be removed. Unfortunately, there is no easy way to distinguish a detection produced by a player from a false detection. However, the most painful false detections, which are the one produced by the light reflections on the court and by the digital signboards, always appears on the right and the left edges of the court (regarding the camera position) and along a straight line parallel to the long court edge at the back of the court.

The strategy used by Keemotion to deal with these noisy detections is to define a new area inside the court and to remove all detections outside it. This area is chosen manually for each stadium.

Figure 2.1 show an example of noisy detections. In this example, the filtering area is delimited by the left and right edges of the court and the dotted lines in the upper and lower part of the court.

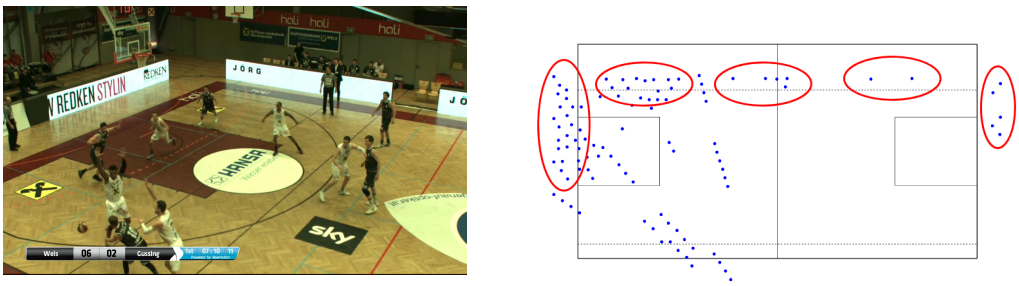


Fig. 2.1 A frame with some noisy detections indicated by red circles.

The detections filtering can however be really painful in situations where the most important players for the action are outside of the area, because we would lost detections that are important to predict the COA.

[2]This algorithm was described in Section 1.1.1.

[3]As a reminder, these noisy detections come from the reflection of a lamp or a digital signboard on the court.

2.3 The Activity map

The Activity Map is a set of cells recording the *activity* in each part of the basketball court. It allow us to give a more convenient representation of the detections and has the advantage of being of fixed size. Here's a more formal definition:

Definition 2.3.1. The *Activity Map* is a set of $M \times N$ cells scattered on the basketball court. These cells are placed at the center of the squares formed by a $M \times N$ uniform grid. Each cell c_i, with $i \in [1, M \times N]$, is a tuple (x_i, y_i, a_i), where (x_i, y_i) is the position of the cell in the court coordinates and a_i is a number recording the activity on that part of the field.

The cells are therefore uniformly distributed on the court. The activity a_i at a cell c_i is computed using all detection $\{d_1, d_2, \ldots, d_n\}$[4], where $d_j = (x_j, y_j)$[5] is the position of detection j on the court. The formula used to compute cell's i activity is the following :

$$a_i = \sum_{j=1}^{n} e^{-((x_i - x_j)^2 + (y_i - y_j)^2)} \cdot \sigma^{-2} \tag{2.2}$$

Where (x_i, y_i) are the cell's coordinates and j ranges over the n detections. Equation 2.2 can be interpreted as the sum of the *gaussian distances* from each detections to the cell. The Gaussian distance is anti-proportional to the Euclidian Distance. The activity is therefore a value in the range $[0, 3 \cdot n]$, with $3 \cdot n$ being the theoretical maximal value of a cell in a situation where the n detections are stacked on the same cell. The value of σ is chosen accordingly to the number of cells on the court, i.e. we need a bigger Gaussian and therefore a bigger σ to capture the activity when there are fewer cells. A big activity a_i in a cell c_i indicates the presence of a lot of detections around c_i. The grid of $M \times N$ cells gives therefore a representation of the distribution of the players on the court, whose accuracy depends on the number of cells used.

Figure 2.2 depicts a simple example of the activity computed in cell c_{17}. The colored peak on c_{17} is the gaussian function used to compute the activity. In this example, there are $9 \times 5 = 45$ cells, depicted by black points, and 6 detections, depicted by green points. The gaussian distances g_{d_i} between each detection d_i and the cell c_{17} are indicated above them. The activity a_{17} recorded in c_{17} is simply the sum of all these distances, as defined in Equation 2.2.

[4]These detections are the set of detections D_i recorded at a given frame f_i, but we drop here the index of the frame to ease notations.

[5]For more details, see Section 1.3.2

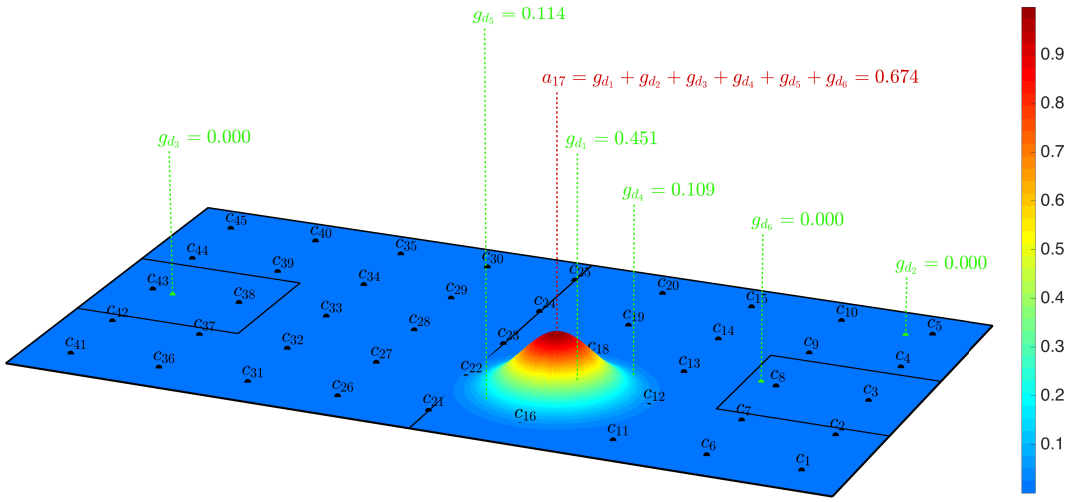
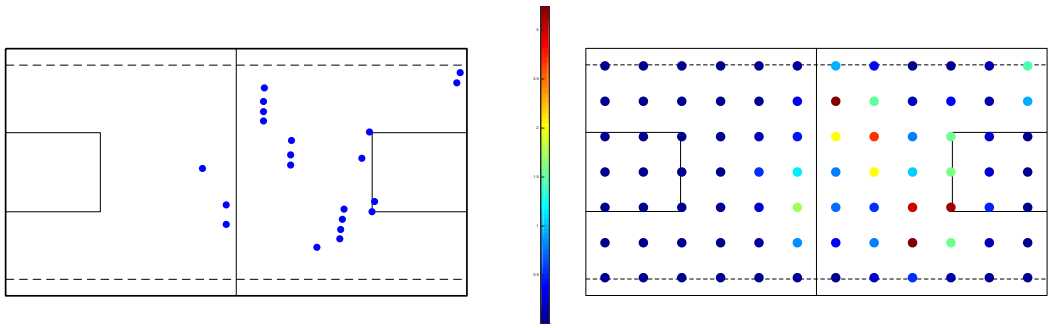


Fig. 2.2 Equation 2.2 applied on few detections to compute the activity in cell c_{17}.

As the same grid of cells is used for each example of the Detections Dataset, the only information of the set of cells that must be kept in the Features Dataset are the activities a_i. The exact method used to compute the Features Dataset will be explained further in Section 2.6.

Figure 2.3 shows a set of detections and the corresponding activity map on the right. Each cell is represented by a circle. Each cell's value is represented by a color. The value corresponding to each color is depicted by the *colormap bar* on the left of Figure 2.3b.



(a) A set of detections. (b) The corresponding Activity Map.

Fig. 2.3 An example of Activity Map composed of 12×7 cells.

As explained before, the main advantage of this approach is to provide a fixed length representation of players disposal. Thanks to the gaussian function, we also have

a continuous features space, which means that a small variation in detections positions results in a small variation in the feature space. A more discrete representation of the players disposal, such as counting the number of players in each grid square[6] do not have this property.

There are three important parameters to tune when using such a grid of cells:

1. **M**, the number of cells along the x axis.
2. **N**, the number of detections along the y axis
3. σ, the only parameter for the computation of gaussian distances.

Using a small amount of cells allows us to have a smaller feature vector. If the feature vector is too big, the learning time of the machine learning algorithm can become impractical. A small amount of cells allows also to smooth the information and therefore to limit the impact of noisy detections or other imperfections in the raw data. However, it's important to have a rather precise grid of cells in order to capture all the information contained in players positions. For example, during a Throw-in, most of the players position themselves around the free throw lane (the painted box beneath the basket), which gives a very specific and precise pattern. A small number of cells cannot capture a pattern like this one.

2.4 The Velocity map

The Activity Map described in the previous section contains information on the distribution of the players on the court at a specific frame, i.e. at a specific point in time. However, another very important information is missing here: the motions of the players. This information is very important to understand in which direction the action is evolving. This could greatly help to distinguish situations with a lot of movements such as a *Counter Attack* or a *Transition Phase* from more static situations such as a *Free-Throw* or a *Jump Ball*[7]. We will therefore add speed information to our cell map by computing a speed vector in each cell. This vector depicts the average direction and intensity of player's velocities around that cell.

In order to create the Velocity Map, we must first find a way to compute the velocity of each detection. The only information we have to achieve this are the detections

[6] This strategy is similar to the one described in [6], see Section 1.3.1.
[7] As a reminder, these situations where described in Section 1.2.1

positions over successive frames. We will therefore try to compute the speeds of detections at time t_i using the detections at time t_{i+1}. However, given a detection $d_{i,j}$ from the set of detections $D_i = \{d_{i,1}, d_{i,2}, \ldots, d_{i,n}\}$ at time t_i, we need to know where this detections moved at time t_{i+1}, i.e. what is its corresponding detection $d_{i+1,k}$ from the set of detections $D_{i+1} = \{d_{i+1,1}, d_{i+1,2}, \ldots, d_{i+1,n}\}$. As the two sets D_i and D_{i+1} are independent and the detections do not have a specific order inside these sets, we do not necessarily have $j = k$. The first step is therefore to find a mapping between the two sets D_i and D_{i+1}. The mapping is defined as vector map_i^{i+1} of size $n = |D_i|$ where $map_i^{i+1}(j)$ is the index of the detection in D_{i+1} corresponding to the detection of index j in D_i. This means that $d_{i,j}$ is mapped to $d_{i+1,map_i^{i+1}(j)}$. We will sometimes refer to this vector simply as *map* in order to ease notations.

With this mapping, it's then easy to compute the speed of each detection: Given a detection $d_{i,j} = (x_{i,j}, y_{i,j})$ at time t_i and its corresponding detection $d_{i+1,map(j)} = (x_{i+1,map(j)}, y_{i+1,map(j)})$ at time $i+1$, the speed $s_{i,j}$ of detection $d_{i,j}$ is given by :

$$s_{i,j} = \frac{\sqrt[2]{(x_{i+1,map_i^{i+1}(j)} - x_{i,j})^2 + (y_{i+1,map_i^{i+1}(j)} - y_{i,j})^2}}{t_{i+1} - t_i} \tag{2.3}$$

However, we will also need the speeds expressed as vectors in the court coordinate systems. The x and y components[8] of the velocity of a detection $d_{i,j}$, $sx_{i,j}$ and $sy_{i,j}$, are therefore computed using:

$$(sx_{i,j},\ sy_{i,j}) = \left(x_{i+1,map_i^{i+1}(j)} - x_{i,j} \quad , \quad y_{i+1,map_i^{i+1}(j)} - y_{i,j}\right) \tag{2.4}$$

The mapping map_i^{i+1} is computed using the Hungarian algorithm, described in the next Section.

2.4.1 The Hungarian Algorithm

The Hungarian algorithm is a combinatorial optimization algorithm used to solve the assignment problem in polynomial time[9]. Here's a formal mathematical definition of the assignment problem:

Given two sets A and B of size n and a cost function $C : A \times B \to \mathbb{R}$, we try to find a *bijection* $f : A \to B$ such that

[8] As a reminder, the x axis is parallel to the long court edges and the y axis is parallel to the small court edges. The origin is placed at the upper left corner.

[9] https://en.wikipedia.org/wiki/Hungarian_algorithm

$$\sum_{a \in A} C(a, f(a)) \tag{2.5}$$

is minimized[10]. We will therefore formulate our problem of finding a mapping between detections at time t_i and detections at time t_{i+1} as an assignment problem, with D_i and D_{i+1} corresponding to A and B respectively. The cost function we will use is simply the Euclidian distance between two detection, i.e.

$$C(d_{i,j}, d_{i+1,k}) = \sqrt[2]{(x_{i+1,k} - x_{i,j})^2 + (y_{i+1,k} - y_{i,j})^2} \tag{2.6}$$

As there is a discrete number of detections, this cost function can be viewed as a two dimensional matrix C, where $C[j][k] = C(d_{i,j}, d_{i+1,k})$.

We will always compute the best assignement between two successive frames. The time interval between two frames is generally of only a few hundredth of a second. This is why the Euclidian distance seams to be the most appropriate cost function, because at human speed and in a few hundredth of a second, the detection $d_{i+1,k}$ corresponding to $d_{i,j}$ should be very close to it.

Figure 2.4 shows a simple example with two sets of detections on which the Hungarian is applied. Blue points represents detections at time t_i and red points time t_{i+1}. A black line is drawn between each pair of assigned detections. Table 2.1 shows the cost matrix corresponding to these sets of detections. The bold values shows the minimum assignment chosen by the algorithm. Table 2.4 shows the corresponding mapping vector *map*.

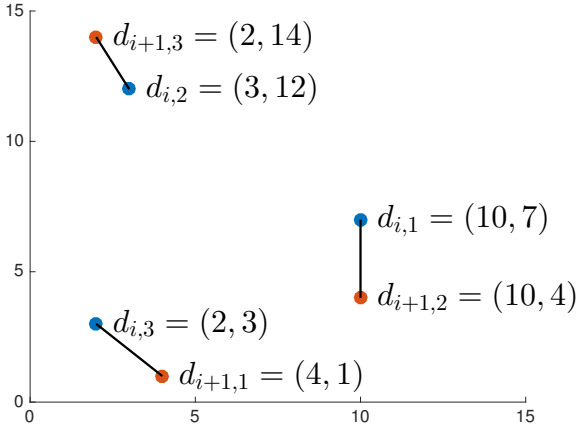


Fig. 2.4 The detections being mapped.

Table 2.1 The cost matrix.

C	$d_{i+1,1}$	$d_{i+1,2}$	$d_{i+1,3}$
d_{i1}	8.4853	**3.0000**	10.6301
d_{i2}	11.0454	10.6301	**2.2361**
d_{i3}	**2.8284**	8.0623	11.0000

Table 2.2 The map_i^{i+1} vector.

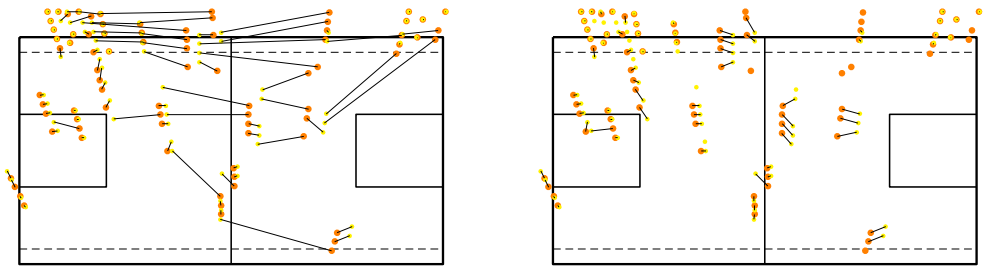
t_i	1	2	3
t_{i+1}	2	3	1

[10]https://en.wikipedia.org/wiki/Assignment_problem

The improved Hungarian Algorithm

Figure 2.5a shows an assignment on real detections. Yellow points are detections at time t_i and orange point are detections at time t_{i+1}. We can clearly see that this assignment is not correct although it's optimal regarding our cost function. This assignment is not correct because they are a lot of assignments going from the right to the left although the players are moving from the left to the right. This happens because in some parts of the court, there are more orange points than yellow points (such as in the upper right corner) and the algorithm must therefore map these orange points with yellow points that are far away, resulting in this incorrect mapping. The simplest way to fix this issue is to allow the algorithm to leave a detection unassigned. A unassigned detection is a detection which is not mapped to another detection and which is assigned a speed of $0\,m/s$. We chose to leave a detection unassigned when the cost of the best available mapping for this detection exceeds a specific threshold. The threshold should be chosen in order to distinguish realistic from unrealistic mappings. A threshold of $10\,m/s$ was finally chosen, which means that if the cost (i.e. the Euclidian distance) divided by the time interval is greater than $10\,m/s$, the detection is leaved unassigned. This improvement is also useful in situations where the number of detections is different at time t_i and at time t_{i+1}: a detection can be leaved unassigned if there are not enough detections in the other set.

As you can see on Figure 2.5b, the Hungarian algorithm improved with a threshold value gives far better results.



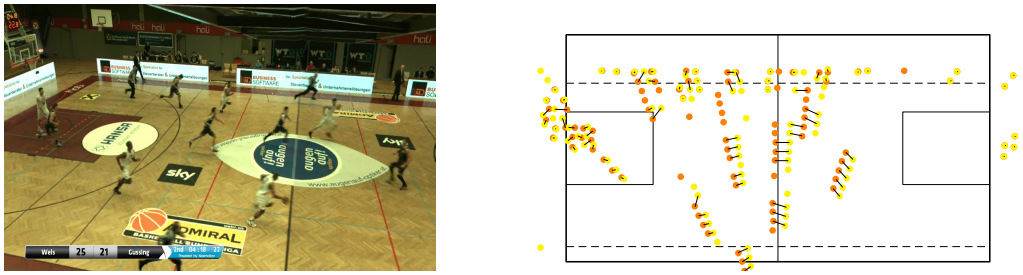
(a) Badly mapped detections (b) Mapping with improved Hungarian

Fig. 2.5 Comparison between classic and improved Hungarian.

Once we know the mapping from detections at time t to detections at time $t+1$, we can compute the corresponding speeds using Equation 2.3. As explained earlier, each unassigned detection is given a speed of $0\,m/s$.

This strategy gives good results when the players are isolated from each other, such as on Figure 2.6. When they are groups of players close to each other and going in

diverse directions, the mapping is much more difficult and therefore the strategy is less efficient.



(a) Video frame at time t (b) Mapping between D^i and D^{i+1}

Fig. 2.6 Another example of the Hungarian algorithm applied to a Counter Attack situation.

2.4.2 Averaging over a few frames

There are two main issues with the strategy described in the previous Section. The first problem is that it is based only on two frames and when they are too much noisy detections or missing detections from one frame to another, the results can be bad. The other problem is that the detections positions are rather imprecise[11] and this imprecision is directly reflected on the computed velocities.

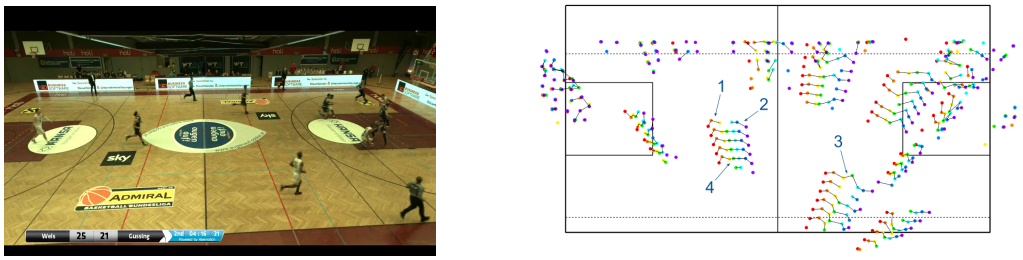
A way to improve the results is to compute and average the speeds using more frames, which allows to reduce the negative impact of missing and noisy detections, and to smooth inaccuracies.

In order to compute the speeds of the set of detections D_i at time t_i, we will use all sets of detections from D_{i-gap} to D_{i+gap}. We therefore use $2 \times gap + 1$ sets of detections. We then apply the Hungarian algorithm for each pairs of successive sets, i.e. between D_{i-gap} and $D_{i-gap+1}$, $D_{i-gap+1}$ and $D_{i-gap+2}$ and so on until $D_{i+gap-1}$ and D_{i+gap}. This corresponds to a set of $2 \times gap$ assignments : $\{map_{i-gap}^{i-gap+1}, map_{i-gap+1}^{i-gap+2}, \ldots, map_{i+gap-1}^{i+gap}\}$.

Figure 2.7 shows an example of this strategy applied on seven successive sets of detections, i.e. with $gap = 3$. The detections at these seven different timestamps $\{t_{i-3}, t_{i-2}, \ldots, t_{i+3}\}$ are represented respectively with the colors *red*, *orange*, *yellow*, *green*, *teal*, *blue* and *purple*. It is therefore the speeds of the green detections (time t_i) we are trying to compute. The black lines shows the detections being mapped. Here, we only try to map detections from successive frames. For example, blue detections (t_{i+2}) can

[11] Especially along the dimension corresponding to camera depth of field.

only be mapped with purple detections (t_{i+3}) and red detections (t_{i-3}) can only be mapped with orange detections (t_{i-2}).



(a) Video frame at time t

(b) Mapping from D^{i-3} to D^{i+3}

Fig. 2.7 Hungarian algorithm applied over seven successive frames: D^{i-3}, D^{i-2}, D^{i-1}, D^{i}, D^{i+1}, D^{i+2}, D^{i+3}.

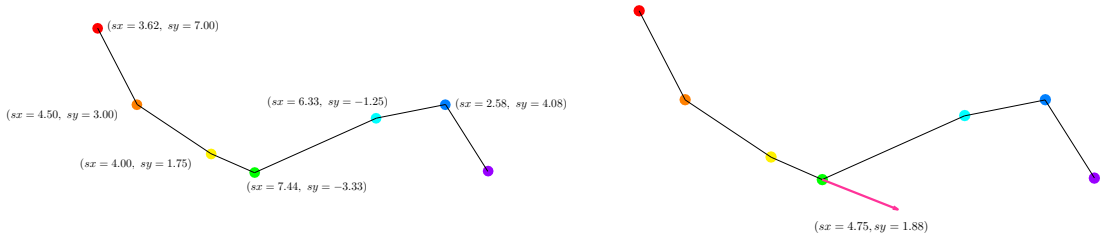
In order to compute the speeds of detections at time t_i, we will treat independently each *path* created by these mappings.

Definition 2.4.1. A *Path* p is an ordered set of detections that are connected thanks to successive mappings. We have therefore $p = \{d_{ij},\ d_{i+1,map_i^{i+1}(j)},\ d_{i+2,map_{i+1}^{i+2}(map_i^{i+1}(j))},\ \dots\}$.

Four different paths are indicated by they respective number on Figure 2.7b. As a path can contain at most one detection per frame, there is therefore a maximum of $2 \times gap + 1$ and a minimum of one detection per path. Path 1 and 2 are of size 3 and path 3 and 4 have the maximal size, i.e. 7.

Once all paths have been computed using the Hungarian Algorithm, the next step is to compute the average velocity of each path, and associate this velocity to the median detection, i.e. the detection at time t_i, because it is the velocities of detections at frame f_i we are trying to compute. Figure 2.8 shows an example of this process applied on a simple path. Figure 2.8a shows the velocities computed for each successive frame using the Hungarian algorithm, where sx is the x component of the velocity and sy is the y component. Figure 2.8b shows the average velocity, assigned to the detection at time t_i (the one in green). This average velocity is represented by the pink vector.

However, this strategy requires that all paths contains a detection recorded at time t_i, which is not the case in practice, as you can see on Figure 2.7. These incomplete paths are often caused by a missing detection. We must therefore find a way to handle such special cases. The simplest strategy would be to simply ignore them and therefore only use paths containing a detection of time t_i. However, we lose a lot of information



(a) Detections velocities (b) Average velocity

Fig. 2.8 Average velocity computed on a path.

by doing that, we will therefore adopt another strategy. We will compute the average velocity for each path and assign this average velocity to the detection which is the closest to t_i. Figure 2.9 shows an example of this strategy. We can see here that the detection of time t_{i+1} is missing. There are therefore two path instead of one. The average velocity computed in each path is assigned here to the detection closest to time t_i.

Hopefully, the problem of missing detections will be partially solved by the preprocessing step presented in the next Section.

The final output of this algorithm on a frame f_i is therefore a set of w velocities $\{v_{i1}, v_{i2}, \ldots, v_{iw}\}$, where w is also the number of paths computed using the Hungarian Algorithm over the $2 \times gap + 1$ successive frames. Here, we have

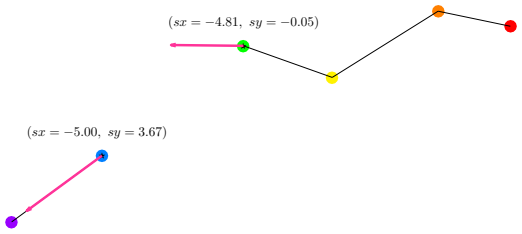


Fig. 2.9 Average velocity for an incomplete path.

$$v_{ij} = (x_{ij},\ y_{ij},\ sx_{ij},\ sy_{ij}) \qquad (2.7)$$

, with $(x_{ij},\ y_{ij})$ the position of the detection of path p_j closest to time t_i and $(sx_{ij},\ sy_{ij})$, the average speed on path p_j.

Of course, this Hungarian Algorithm averaged over a few frames could be improved or replaced by more complex and precise methods such as state-of-the-art tracking algorithms, but these methods are out of the scope of this thesis. The strategy described here was chosen because it's simple and sufficient for our goal which is to estimate locally players motions.

2.4.3 Cells velocities

Thanks to the Hungarian algorithm, we have computed the velocities of a set of detections at a given frame. We now want to spread this new information to the cells as we have already done with the activity. However, we have this time a more complex information: the velocities are vector quantities involving at least two scalar values. Each of the two components $(sx_{ij},\ sy_{ij})$ of the velocities v_{ij} have therefore been processed separately to compute two new values in the cells: a local average speed sx along the x axis and a local average speed sy along the y axis. The set of cells together with their velocities components form the Velocity Map: Here's a more formal definition, which is similar to the definition of the Activity Map, given in Section 2.3:

Definition 2.4.2. The *Velocity Map* is a set of $M \times N$ cells scattered on the basketball court. These cells are placed at the center of the squares formed by a $M \times N$ uniform grid. Each cell c_k, with $k \in [1, M \times N]$, is a tuple $c_k = (x_k, y_k, sx_k, sy_k)$, where (x_k, y_k) is the position of the cell in the court coordinates and $(sx_k,\ sy_k)$ are the x and y components of the velocity recorded on that part of the field.

The speeds $(sx_k,\ sy_k)$ are computed using equations similar to the one that was used for the activity in Section 2.3. Given the set of velocities $\{v_{i1}, v_{i2}, \ldots, v_{iw}\}$ described in Equation 2.7, the two speeds components sx_k and sy_k at cell c_k are computed as follows:

$$sx_k = \sum_{j=1}^{w} sx_{ij} \cdot e^{-((x_k - x_{ij})^2 + (y_k - y_{ij})^2)} \cdot \sigma^{-2} \tag{2.8}$$

$$sy_k = \sum_{j=1}^{w} sy_{ij} \cdot e^{-((x_k - x_{ij})^2 + (y_k - y_{ij})^2)} \cdot \sigma^{-2} \tag{2.9}$$

Figure 2.10 shows the computation of the Velocity Map on a frame f_i. Figure 2.10b shows the set of detections at time t_i, Figure 2.10c the hungarian computed over seven successive frames ($gap = 3$) and Figure 2.10d the corresponding Velocity Map. The pink points are the 15×7 cells and the pink arrows are the velocities in each cell.

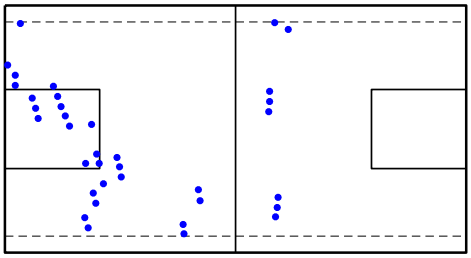
2.5 From detections to players positions

If we take a closer look at the detections provided by Keemotion's algorithm[12], we can see that, in most cases, the same player produces more than one detection and

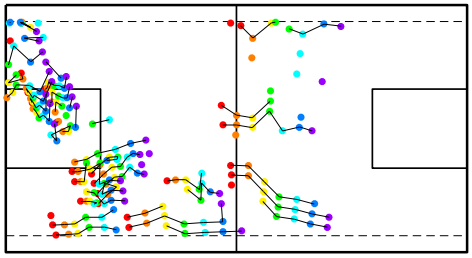
[12]Described in Section 1.1



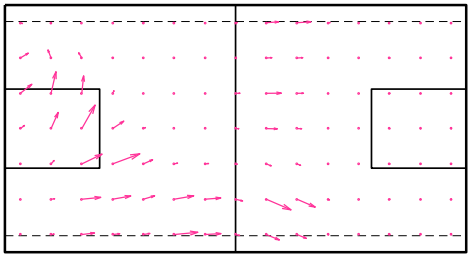
(a) Video frame f_i



(b) Detection set D_i



(c) Hungarian on seven successive frames



(d) The final velocity map

Fig. 2.10 An example of the process used to compute the Velocity Map.

these detections are always perfectly aligned with the camera. We can also see that the **number of detections per player** is not a constant. An example is shown in Figure 2.11. This number actually depends on three main parameters. The first one is the distance between the player and the camera. In fact, a player which is far from the camera will appear very small in the image captured and will therefore produce few detections. The second parameter is the saliency of the player with respect to the background: some player have a color similar to the background and are therefore hardly detected. The third parameter is the presence of obstacles between the player and the camera: sometimes a player is partly hidden by other players and produces therefore fewer detections.

The main problem with the varying number of detections per player is the bad influence on the precision of the Occupancy Map and the Speed Map. For example, a player with a lot of detections will have a great impact on the value of several cells in the Activity Map, while a player with fewer detections will have a smaller and disproportionate impact on the cells values. This is an issue because all players should have equally weighted impacts on the Activity Map.

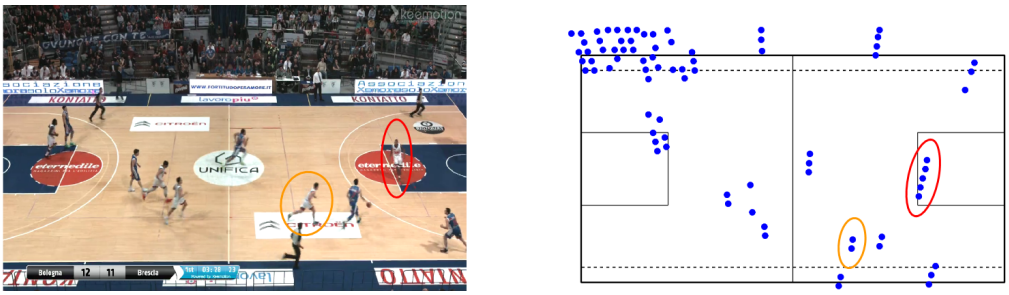


Fig. 2.11 An example of the varying number of detections per player.

The main problem we can encounter with the Speed Map is an incorrect estimation of the player's velocities around a cell. In order to see that, let's depict a simple example. Imagine a first situation where a slow player with five detections is positioned next to a cell c_i. In the second situation, we have a fast player with only one detection next to the cell c_i. In the first situation, the velocity in cell c_i is computed as the sum of five small velocities weighted by their gaussian distance to the cell. In the second situation, the velocity in cell c_i is computed as only one velocity weighted by its gaussian distance to the cell. It's therefore probable that the velocity in cell c_i is identical in the two situations although it should be bigger for the situation with the fast player.

The best way to avoid these situations is to have only one detection for each player on the court. We need therefore to find a way to cluster detections belonging to the same player and replacing them with only one wisely chosen detection. The strategy used the achieve this follows three steps:

1. Create groups of detections aligned with respect to the camera using Hough Transform.

2. Create subgroups of detections belonging to the same player.

3. Compute corresponding players positions.

These three steps are explained further in the next Sections. Although the Hough Transform is the algorithm used to perform the first step, we will use this name to refer to the whole process in the other Sections.

2.5.1 Hough Transform

The *Hough Transform* is a feature extraction technique used in image analysis to find imperfect instances of objects within a certain class of shapes by a voting procedure[13]. This technique works by creating an accumulator array in the shape's parameter space (also called Hough space). Each point in this accumulator array corresponds therefore to a unique instance of the shape. Once the voting procedure is done, the maximas in the accumulator array denotes the best candidates for the desired shape.

The Hough Transform is commonly used to detect straight lines in an image. These lines are generally described by two parameters (θ, ρ), where *Theta* is the angle between the line and the horizontal axis, and *Rho* is the perpendicular distance between the line and the origin. The accumulator array is therefore a two dimensional array ranging over all possible (θ, ρ) pairs (whose ranges depend on the image size).

We will use an algorithm inspired from the Hough Transform for straight line to cluster detections belonging to the same player.

A naive approach would be to apply directly the basic Hough Transform on all detections. Each detection would therefore vote for all straight lines passing through it. However, we could greatly improve this strategy thanks to a simple observation: as you can see on Figure 2.12, the detections belonging to the same player are always aligned with respect to the camera, because a player standing upright can only generates detections along a straight line going from the camera to his position. The naive approach is therefore inefficient because we are only interested in a subset of the straight lines depicted by the Hough space: the subset composed of all lines passing through the camera position. Consequently, we will use a simplified Hough space with one parameter: the angle formed by the line and the camera axis (the y axis). The corresponding accumulator array is therefore a $1 \times t$ vector, where each *bin* b, correspond to an angle θ_b (in degree) in the range $[-90, 90]$. The number of *bins* t depends on the precision used. We can have, for example, one bin at each 0.5 degree step. In that case, $t = 360$.

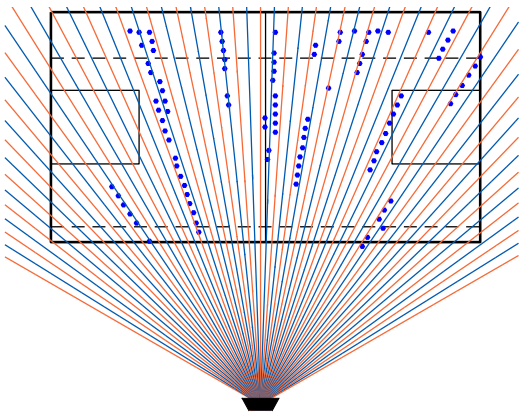


Fig. 2.12 Camera's straight lines.

[13] https://en.wikipedia.org/wiki/Hough_transform

We can finally come up with the following algorithm, which is in fact the Hough transform applied on the simplified parameter space:

For each detection d_j in the current frame $D_i = \{d_{i1},\ d_{i2},\ \ldots,\ d_{in}\}$, we compute the angle θ_b of the line going from d_j to the camera and we vote for the corresponding bin b in the accumulator array. Hopefully, as we know the exact position of the camera[14], the computation of this angle is straightforward. At the end of the process, when each detection d_j has voted for a line, the peaks in the accumulator array denote the lines along which there are a lot of detections. They correspond therefore to players silhouettes.

We must now use this accumulator array to divide D_i into groups of detections aligned with respect to the camera. This is done by repeating the following process for each non empty bin in the accumulator array and starting with the highest bin values. Given a bin b, we create a group G_b with all detections having voted for bin $b-1$, b and $b+1$. We need to include the close neighbors $b-1$ and $b+1$ of b in the group because, as the accumulator is a discrete representation of a continuous parameter space, two aligned detections may have voted for two different, but close, bins. We then set the values of $b-1$, $b+1$ and b to 0 in the accumulator array. Once a detection has been included in a group, it cannot be included in another one in a subsequent iteration.

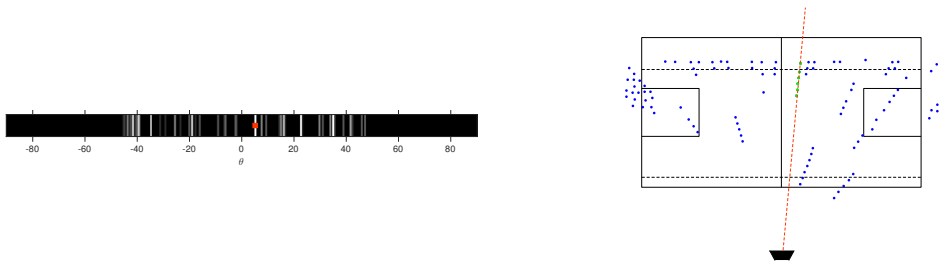
An example of this process is depicted in Figure 2.13. Figure 2.13a shows the one dimensional accumulator array build with the detections from Figure 2.13b. The red point denotes a peak in the array. The line corresponding to this peak is drawn in red on the court. The detections in green are the detections which have voted for this line in the accumulator array and the detections which have voted for the close neighbors. At the end of the process, the green detections are therefore gathered in the same group.

Figure 2.14 shows 26 groups of detections which have been created using the Hough Transform. Each group is emphasized with a different color.

2.5.2 Detections clustering

As you can see on Figure 2.14, we have created groups of detections aligned with respect to the camera. However, we cannot interpret each of these groups as a unique player, because it's possible to have some players aligned on the same straight line passing through the camera. An example of such a situation is shown in Figure 2.15. We must

[14]The camera position for each stadium was provided by Keemotion.



(a) A peak in the Hough space.

(b) The chosen line and the corresponding detections.

Fig. 2.13 Hough transform inspired algorithm applied on a set of detections.

therefore find a way to split these groups into subgroups of detections belonging to the same player.

Given a group G_b of aligned detections obtained with the Hough Transform, we can first start to split it into subgroups such that the minimal distance between any pair of detections from two different subgroups is greater than a specific threshold. This method is depicted in Figure 2.16, where a group of detections (green) has been split into two subgroups (orange and red). A threshold of 1.5 meter was used here.

However, this first split is not sufficient for situations where two players are aligned and close to each other. Such a situation is shown in Figure 2.15. We will therefore do a second splitting of these groups based on a prediction of the number of detections that a player should normally produce. The first fact we must notice is that this number depends on the distance of the player to the camera: A player which is at the back of the court appears smaller in the captured image and produce therefore less detections than a player closer to the camera. Actually, it would be possible to compute directly and precisely the number of detections that a player must produce with respect to its distance to the camera, but we will use instead a simpler strategy which works quite well in practice. This strategy is based on a simple interpolation: using an estimation of the number of detections per player at two distinct distances of the camera and assuming that this number evolves linearly relative to the distance to the camera, we can create a linear function $y = f(x)$ taking as input a distance and returning an estimated number of detections.

We can then use this function $f(x)$ as follows: given a group of n detections $G = \{d_1,\ d_2,\ \ldots,\ d_n\}$, arranged in ascending order with respect to their distance to the camera, we try to split G into s subgroups of roughly the same size, with :

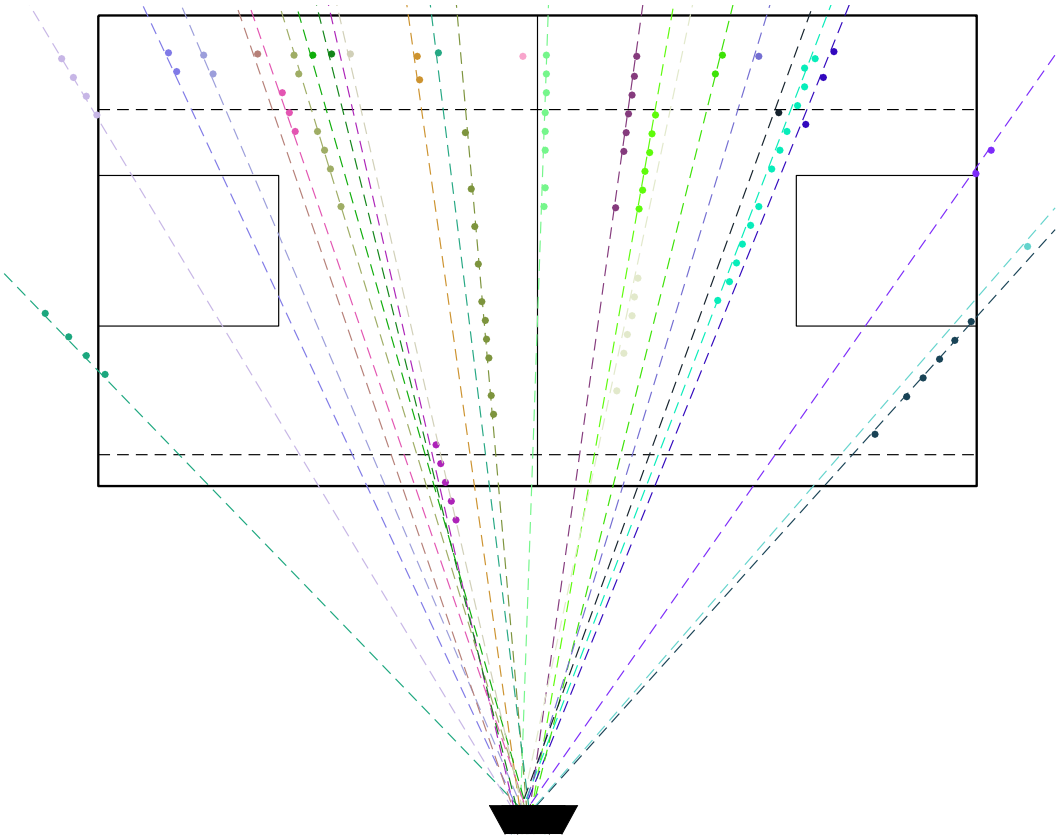


Fig. 2.14 An example of detections clustered with respect to their alignments with the camera.

$$s = \frac{n}{f(\sqrt[2]{(x_1 - x_{cam})^2 + (y_1 - y_{cam})^2})} \tag{2.10}$$

Where $(x_1,\ y_1)$ is d_1 position and $(x_{cam},\ y_{cam})$ is the camera position.

2.5.3 Players positions

We now have clustered our detections in smalls groups, where each group depicts a different player (or another salient entity such as the referee, a coach, a billboard etc). The final step of our algorithm is to infer the corresponding player position. We will do that simply by taking the mean of these detections. This strategy seams not optimal because we could have taken the lowest detection as the player's position, because this detection must be the closest to the player's feets and therefore to his position on the ground. But in practice these two possible positions are really close to each other

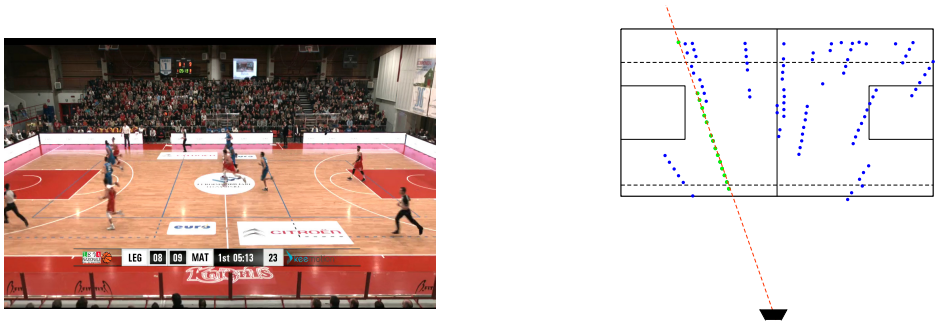
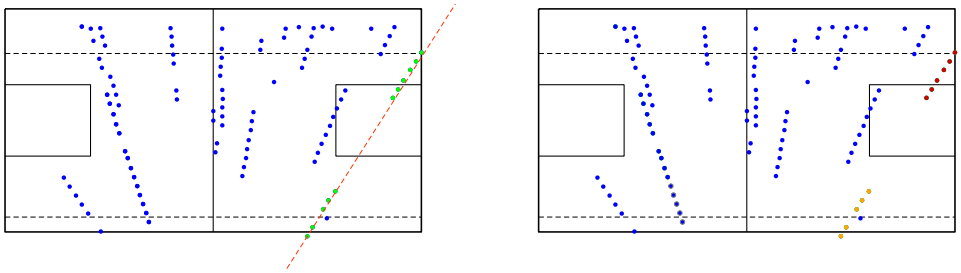


Fig. 2.15 A situation where some players are aligned with the camera.



(a) Detections clustered with Hough transform.

(b) Previous group of detections split into two subgroups.

Fig. 2.16 An example of the method used to split groups of distant detections.

and the one we have chosen gives a much more stable estimation when we compare the results over successive frames. As the algorithm to infer players positions will be applied before the computation of the speeds[15], the results of the Hungarian Algorithm will therefore be also much more stable.

Figure 2.17 shows two examples of players position inferred using the process described in this Section. The original detections are shown in gray and the inferred players positions in blue. Players positions corresponding to groups of less than 3 detections and outside the filtering area defined in 2.2 have been removed.

When applying this whole Hough transform process to a frame D_i, all detections in D_i are replaced by the returned players positions.

If the average position of a group of detection is outside of the filtering area described in Section 2.2, we consider this detection as a player only if the group contains more than 3 detections. This process allows to add another layer when filtering false detections and it works generally well because false detections are generally no aligned like players detections.

[15]Described in Section 2.4.

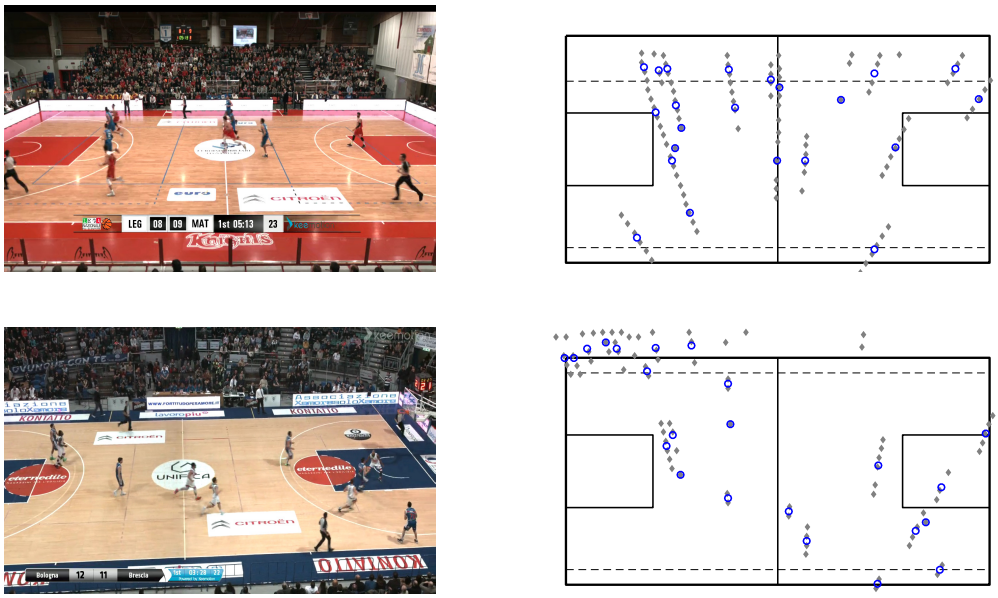


Fig. 2.17 Example of players positions inferred using Hough transform.

2.6 The Feature Map

In the previous Sections, we have seen two preprocessing methods and two feature extraction strategies. We will use them in the two following routines to create the feature vectors composing the final Feature Dataset. Firstly, we start to preprocess each set of detections in the Detection Dataset using the Hough Transform described in Section 2.5, in order to replace the detections by unique players positions. We then use the filtering strategy explained in Section 2.2 to remove false detections around the court. Secondly, we launch the feature extraction routine, which is applied on a set of detections only if it's labeled in the Annotation Dataset[16]. The goal of this second routine is to create the Activity Map and the Velocity Map, as explained in Section 2.3 and 2.4. As a reminder, the Activity Map is a set of cells $c_j = (x_j, y_j, a_j)$, with a_j the activity recorded in the cell and the Velocity Map is also a set of cells $c_j = (x_j, y_j, sx_j, sy_j)$. As we choose to use the same cell grid for the two Maps, we can combine them into a unique map, named the Feature Map. However, we had here a new index in the mathematical representation, to depict explicitly the frame in which cells values were recorded.

Definition 2.6.1. The *Feature Map* is a set of $M \times N$ cells scattered on the basketball court. These cells are placed at the center of the squares formed by a $M \times N$ uniform grid. Each cell $c_{i,j}$, with $j \in [1, M \times N]$, is a tuple $c_{i,j} = (x_j, y_j, a_{i,j}, sx_{i,j}, sy_{i,j})$, where

[16] Because each example in the Feature Dataset must be labeled.

(x_j, y_j) is the position of the cell in the court coordinates, $a_{i,j}$ is a number recording the activity around the cell at frame f_i and $(sx_{i,j},\ sy_{i,j})$ are the x and y components of the velocity recorded around the cell at frame f_i.

The final Feature Dataset FD is simply the combination of these l Feature Maps:

$$FD = \begin{bmatrix} a_{1,1} & \dots & a_{1,M\times N} & sx_{1,1} & \dots & sx_{1,M\times N} & sy_{1,1} & \dots & sy_{1,M\times N} \\ a_{2,1} & \dots & a_{2,M\times N} & sx_{2,1} & \dots & sx_{2,M\times N} & sy_{2,1} & \dots & sy_{2,M\times N} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ a_{l,1} & \dots & a_{l,M\times N} & sx_{l,1} & \dots & sx_{l,M\times N} & sy_{l,1} & \dots & sy_{l,M\times N} \end{bmatrix} \tag{2.11}$$

Figure 2.18 shows this whole process applied on a real example.

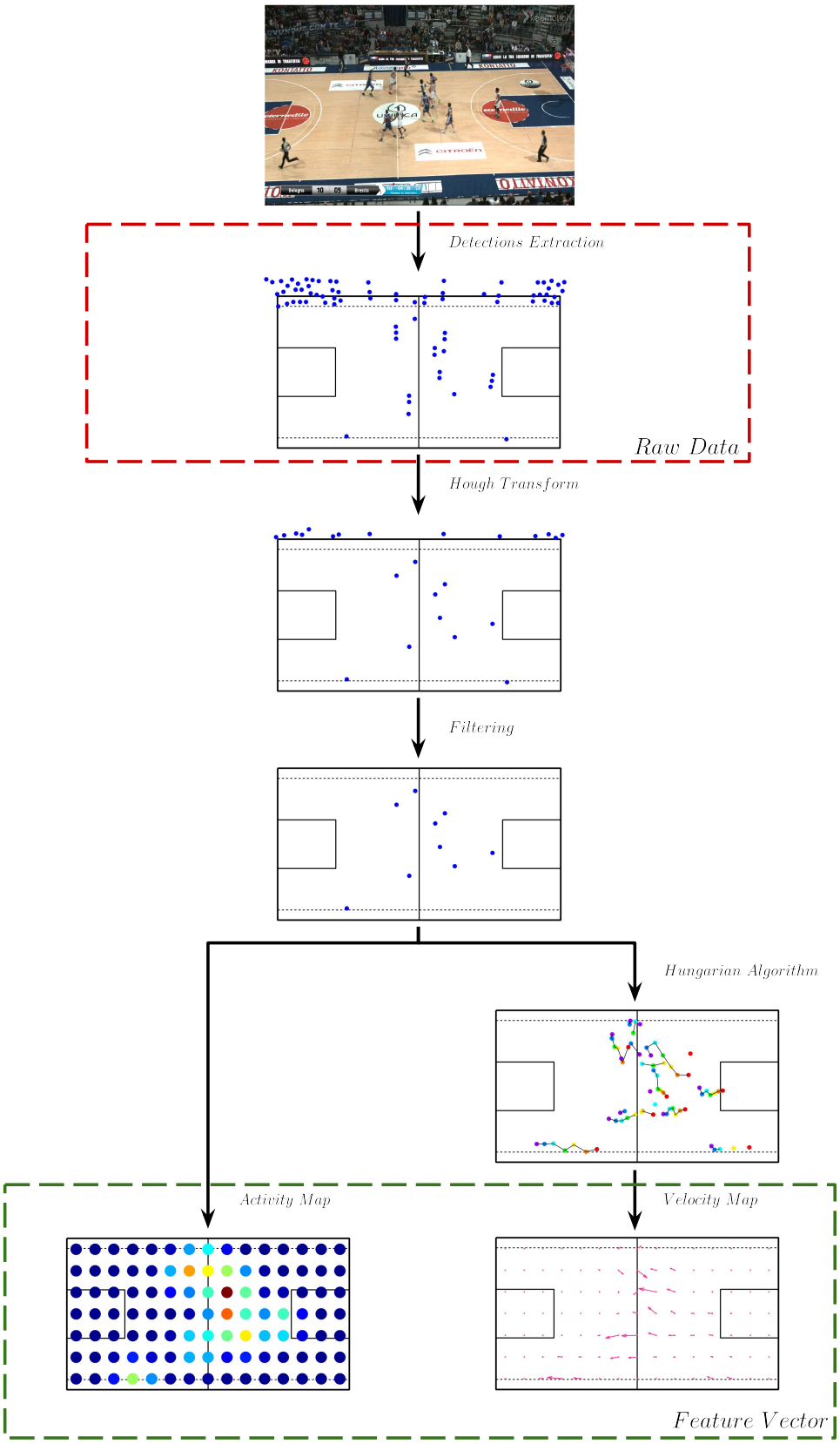


Fig. 2.18 Complete Feature Extraction Process.

Chapter 3

Learning

As explained in previous chapters, we will use some famous regression algorithm to predict the position of the COA. As a reminder of Section 1.2.2, this COA is defined as a two dimensional coordinate on the basketball court. We will however only try to predict the x coordinate of the COA, leaving the 2 dimensional case as a future improvement. The output of the regression is therefore a simple scalar value, which can be interpreted as a coordinate on a straight line cutting lengthwise the basketball court.

The problem is therefore mathematically modeled as:

$$x_i^{COA} = f(\mathbf{v}_i) \tag{3.1}$$

, where f is the regressor to be learned, x_i^{COA} is the x-component of the Center Of Attention at a frame f_i and $\mathbf{v}_i$ is the feature vector built at f_i.

We could also have modeled the problem as a classification, with a discrete number of COA, for example $M \times N$ COA corresponding to the positions of the $M \times N$ cells of the cell map[1]. However, this approach is less interesting than a regression because of the way classification algorithm are optimized. In fact, the objective function that is usually minimized in classification penalizes equally each misclassification, which is not a good criterion for our problem because, as we want the predicted COA to be as close as possible in court space to the real COA, some misclassifications are much more painful than others.

A common important criterion we must take into account while choosing a regressor is a good generalization property. As explained in Section 1.3.4, the observations labeling process is very slow and the dataset used to train the regressor is therefore

[1] See Section 2.3.

small and maybe not representative of the whole real feature space. Using a small dataset means therefore that the training process may be sensitive to overfitting. The regressor must therefore have good generalization properties in order to predict correctly the COA in any situation given a small dataset for training.

In this Chapter, we will review briefly the three regression algorithms used, explain why they were chosen and describe the parameters to be tuned. We will then explain the *10-fold cross validation*, which is the process used to assess the performance of each algorithm. We will finally explain the preprocessing step performed on the Feature Dataset before training. The practical results on the real dataset will be described in Chapter 4.

3.1 Support Vector Regression

Support Vector Regression is the regression version of Support Vector Machine, a very popular kernel based model used in supervised learning for classification and regression. The basic idea behind the SVM approach is to map the input space to a high-dimensional space using a kernel function and then linearly fit the input values $\mathbf{x}$ to the desired responses $\mathbf{y}$. A more detailed description of the model can be found in [17], [4] and [2]. We will use a particular instance on SVR called ϵ-SVR, where ϵ is a parameter used as a threshold on the maximal tolerated (non-penalized) error on the training target values. We will use the Radial Basis Function RBF as kernel, because it has only two hyperparameters to tune and can handle non linear relations between inputs and outputs. The RBF is as follows:

$$K(\mathbf{x}_i, \mathbf{x}_j) = e^{-\gamma \left\| \mathbf{x}_i - \mathbf{x}_j \right\|^2}, \ \gamma > 0 \tag{3.2}$$

The two parameters we will have to tune in SVR are C and γ. The Cost[2] C is a trade-off between training error and flatness of the solution and allows therefore to control the generalization property of the model and avoid overfitting. The other parameter γ is the size of the radial basis functions and control therefore the radius of influence of the support vectors.

[2]https://www.researchgate.net/post/In_support_vector_machinesSVM_how_we_adjust_the_parameter_C_why_we_use_this_parameter

3.2 Multilayer Perceptron

A Multilayer Perceptron[3] is a particular instance of a feedforward neural network model based on three layers of neurons. The network architecture is illustrated in Figure 3.1, found in [2]. The first layer is called the input layer and is composed of one neuron per entry of the feature vector to be processed. The second layer is composed of a varying number of neurons, called the hidden units, and is fully connected to the first and the third layer. The third layer is called the output layer and is composed of one neuron per desired output. As the only value we want to predict is the x component of the COA, there is only one neuron in the last layer of our model. A MLP can be seen as a directed graph, where each neuron's output, except for the input layer, is a weighted sum of it's inputs on which an activation function in applied. It can be mathematically modeled[4] as follows:

$$y_k(\mathbf{x},\ \mathbf{w}) = \sum_{j=1}^{M} w_{kj}^{(2)}\, h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_{k0}^{(2)} \tag{3.3}$$

, where $\mathbf{x}$ is the input vector and D its size, h is a differentiable and nonlinear activation function[5], $w_{ji}^{(1)}$ are the weights to be optimized during training, $w_{j0}^{(1)}$ are the biases parameters and M is the number of hidden units. The surpervised learning technique used to train the network is called *backpropagation*. The main parameters we will need to tune are M and the maximal number of iterations during training. We will try to use a MLP to infer the COA for various reasons. Firstly, it's a well known algorithm, widely used for machine learning tasks, which can distinguish non linearly separable data and is said to be a universal approximator. It's also much more compact and faster to evaluate than SVR, which can be advantageous for a real-time system such as Keemotion framing algorithm. The model is also very user-friendly as there are very few parameters to tune. A MLP can also be easily extended to multiple

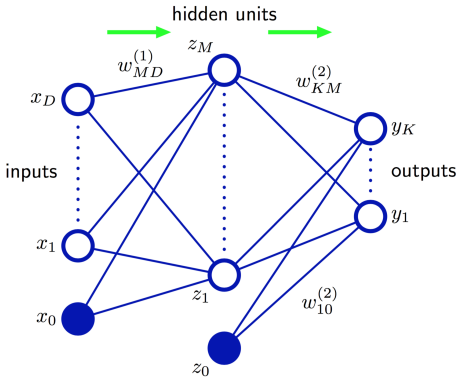


Fig. 3.1 A Multilayer Perceptron.

[3] https://en.wikipedia.org/wiki/Multilayer_perceptron
[4] See 3.1.
[5] Generally the sigmoid function.

outputs. However, a MLP is generally very slow to train because it is a non-linear and non-convex optimization problem.

3.3 Random Forest

As described in [12], Random Forest[6][7][8] is a type of ensemble learning method, used in classification, regression and even unsurpervised learning, based on the combination of a group of weak tree models into a powerful model. Each tree of the Random Forest is built independently using a bootstrap sample of the training set. A bootstrap sample is obtained by randomly sampling, with replacement, the training set. Randomness is also added while growing the trees: each node is split using the best[9] split among a random subset of the features. The size of this subset of features is a fixed constant called *mtry*. The trees are grown without pruning but with a minimum of 3 examples per node. Once the forest is grown, a prediction for a new example is made by taking the average predictions of all trees. The are three key advantages of using Random Forest. Firstly, bagging and random feature selection at each node reduce chances of overfitting and provide therefore good generalization property to the model. Secondly, aggregation over several independently grown trees gives a higher model accuracy than a single tree. Finally, Random Forest is user friendly because it has only two main parameters to tune. The most important one is the total number of trees *ntree* and the second one is the size *mtry* of the random subset of features used for splitting each node.

3.4 Variable scaling

As explained in [9], it's generally a good practice to linearly scale the dataset to the range $[1, +1]$ or $[0, 1]$ before the learning process is launched. The x-component of the COA and the values of the cells in the Occupancy Map have therefore been scaled to the $[0, 1]$ interval. However, the values of the cells in the Velocity Map have been scaled to the $[-1, 1]$ range, as the speeds can be positive ($+x$ or $+y$ direction) or negative ($-x$ or $-y$ direction). The Occupancy Map and the Velocity Map have of course been scaled with respect to their respective global maxima. Features are therefore not being scaled independently.

[6] See [12]
[7] https://en.wikipedia.org/wiki/Random_forest
[8] http://www.analyticsvidhya.com/blog/2015/09/random-forest-algorithm-multiple-challenges/
[9] In terms of Gini Impurity measure.

3.5 10-fold cross validation

10-fold cross validation is a model validation technique we will use to assess the performance of the three machine learning algorithms. It works in two steps. Firstly, the original dataset is randomly partitioned into 10 equal sized subsamples. Secondly, each of these subsamples is used separately as a test set on a model trained with the nine other subsamples. 10-fold cross validation is used to give a much more accurate estimation of the prediction performance of an algorithm and avoid problems such as overfitting. The performance obtained are therefore much more representative of how the algorithm will generalize to totally new observations.

In our case, we will use a custom method to create the 10 folds. The reason is the following. Two successive frames, even with a half-second time step, are generally very similar, because players distribution is almost the same and the patterns formed by the false detections are also almost identical. If we create the ten folds with a random sampling of all examples, there are many chances that, at a given iteration, most frames in the test set have their neighboring frames in the training set. In this case, the algorithm can easily predict the COA for this test set thanks to similarities of these neighbors. This situation corresponds to overfitting and the performance obtained are therefore much more better than the one we would obtain on totally new examples. We must therefore adapt the 10-fold creation process and use a random sampling of the sequences and not of the examples. All frames of the same sequence are therefore put in the same fold. A sequence is defined as a set of successive frames, corresponding generally to about 10 seconds of video and depicting a particular action during a basketball game. We will explain in much more details how the dataset is divided into sequences in Section 4.1.

Chapter 4

Results and Validation

In the previous Chapters, we have seen how to build the basic raw dataset, extract some meaningful features[1] and annotate a subset of the collected examples. We have also seen how to express our problem of finding the Center Of Attention as a supervised learning task using a trained regressor. We have finally seen how to model and train this regression using some state-of-the-art machine learning algorithms. In this chapter, we will provide much more practical details on the data collected and the annotated examples. We will also provide some qualitative assessment of the feature extraction strategy. We will then see how to tune the parameters of the three machine learning algorithms. We will finally compare their prediction accuracy.

4.1 Video sequences selection

In order to build the two datasets described in Section 1.3.3 and 1.3.4, the annotated games sequences were chosen wisely, because the final dataset has to contain a significant portion of tricky game phases[2]. The reason is the following. As we have seen in Section 1.2, the Center of Mass of the group of players is in most situations a good approximation of the COA. If we create the dataset without paying attention to the number of tricky game phases included, we could expect our regression to learn a COA position near the Center of Mass, because this strategy would work correctly for most situations.

The regression would therefore not work correctly in the tricky situations whereas these are the game phases we are principaly interested in. However, it is also important to include "normal game phases" in the dataset because, in the end, the algorithm must be able to predict the COA for both normal and tricky situations.

[1] This process is summarized in Figure 2.18.

[2] These tricky game phases were described in Section 1.2.1

As a reminder, we defined 7 tricky game phases categories[3]. These categories, together with the label used to refer to them are listed in Table 4.1.

The final Annotation Dataset is composed of 74 sequences, for a total of 1351 annotated frames. Each sequence correspond to a tricky game phase and is therefore assigned one of the eight categories. The annotations were made on 3 basketball games recorded in three different stadiums: Wels, Legnano and Bologna. There are no sequences explicitly labeled as a "normal phase[4]". The dataset contains however such examples because the annotated sequences are quite long and the beginning and the end of these sequences correspond generally to "normal phases". However, it would be interesting for future work to explicitly isolate these normal situations from tricky situations.

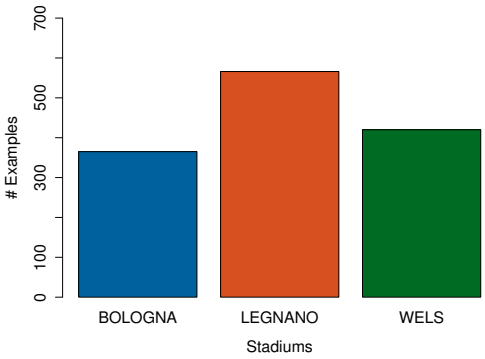


Fig. 4.1 Number of annotated frames per stadium.

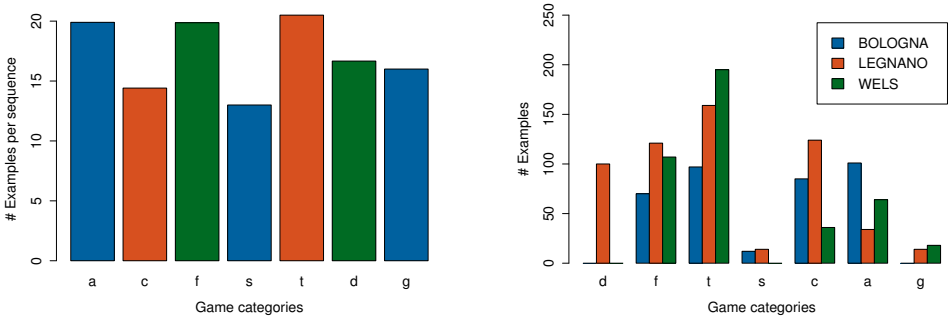
Table 4.1 Game phases categories

Categories	Label
False detections	d
Free-Throw	f
Transition	t
Jump ball	s
Counter attack	c
Throw-in	a
Attack phase	g

Figure 4.1 shows how the 1351 examples are distributed among the three stadiums. Figure 4.2a show the average number of frames per sequence for each game category. We can see in this figure that counter attack and jump ball sequences are on average shorter than the others. Figure 4.2 shows the number of examples in each game category. We can clearly see here that some categories are much more represented than other. The dataset should be improved in future work to solve this unbalanced category distribution.

[3]See Section 1.2.1.

[4]We call a "normal phase" a phase which is correctly framed by Keemotion's algorithm.



(a) Average number of frames per sequence for each game category.

(b) Number of annotated frames per game phase category.

Fig. 4.2

4.2 Feature extraction process performance

In this Section, we will use a qualitative approach to asses the performance of the different steps of the feature extraction process. We will try to analyze the results obtained on some randomly chosen frames and show situations in which these methods work and don't work.

4.2.1 Hough Transform and Filtering

The validation of the Hough Transform and the Filtering strategy[5] was made by analyzing the performance over a few randomly chosen frames. Figure 4.3 shows the resulting filtered players positions for two frames in each stadium. The original detections are shown in gray. The players positions obtained with the Hough Transform are depicted by colored circles: red for the positions outside the filtering area and blue for the others.

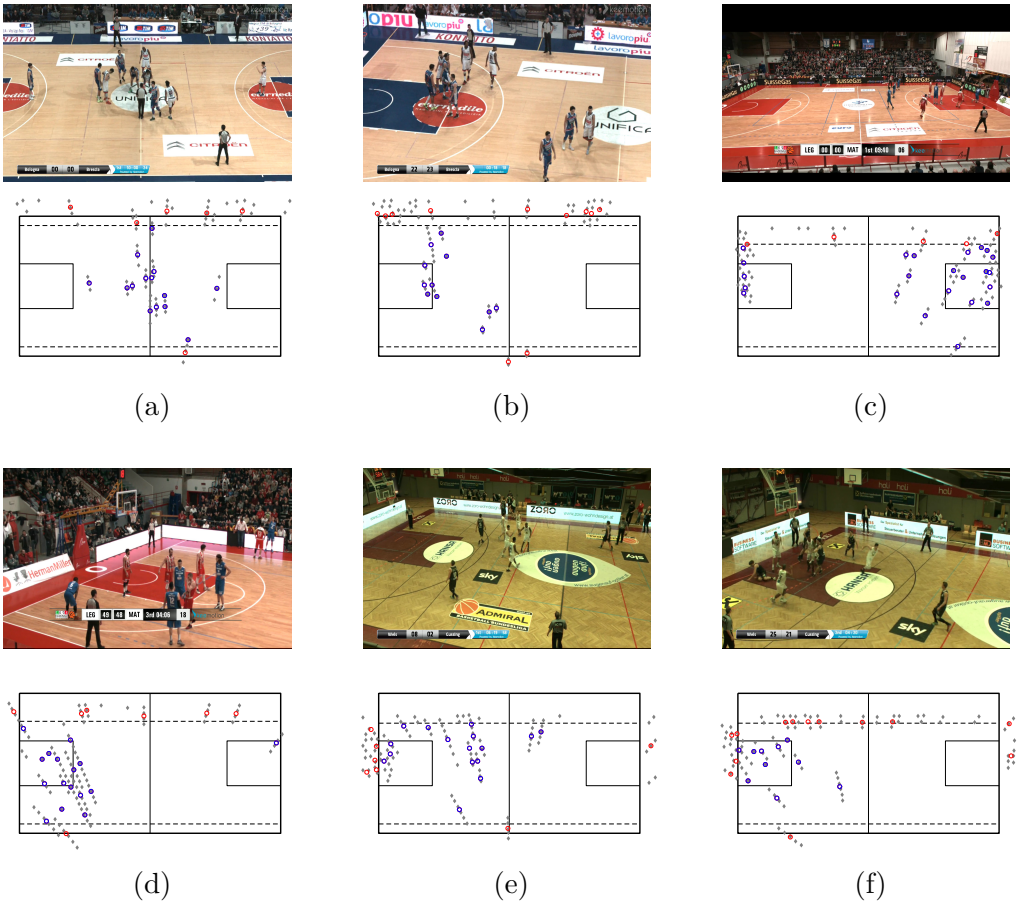
The performance of the Hough Transform is generally good. However, we can see that sometimes a set of aligned detections corresponding to a unique player silhouette is replaced by two detections, such as the referee at the bottom of Figure 4.3a or the two silhouettes near the Free-Throw area in Figure 4.3b. This happens because the detections vote for enough distant bins in the accumulator array to be considered as belonging to separate players.

Regarding the Filtering strategy, we can see that it gives good results: most of the noisy detections are removed at each frame. Even the referees are often removed

[5]These strategy were explained in Section 2.5 and 2.2.

from the set of detections, such as in Figure 4.3a, 4.3d and 4.3e, which is quite good. However there are some particular situations where players are removed from the set of detections and other situations where noisy detections are kept. For example, the noisy detections causing the bad framing in Figure 4.3c and even in Figure 4.3e are not removed. The filtering is also quite ambiguous on the left court edge in Figure 4.3f. On Figure 4.3b, we can see that a very important player is removed from the set of detections: the player responsible for the Throw-in. A improvement that could be done is to only filter detection in the upper part of the court.

From these examples, we can clearly see that the filtering is a trade-off between a complete but noisy information and a less noisy but possibly incomplete information. In Section 4.5, we will try to predict the COA with and without the Hough Transform and the Filtering in the feature extraction process and compare results.



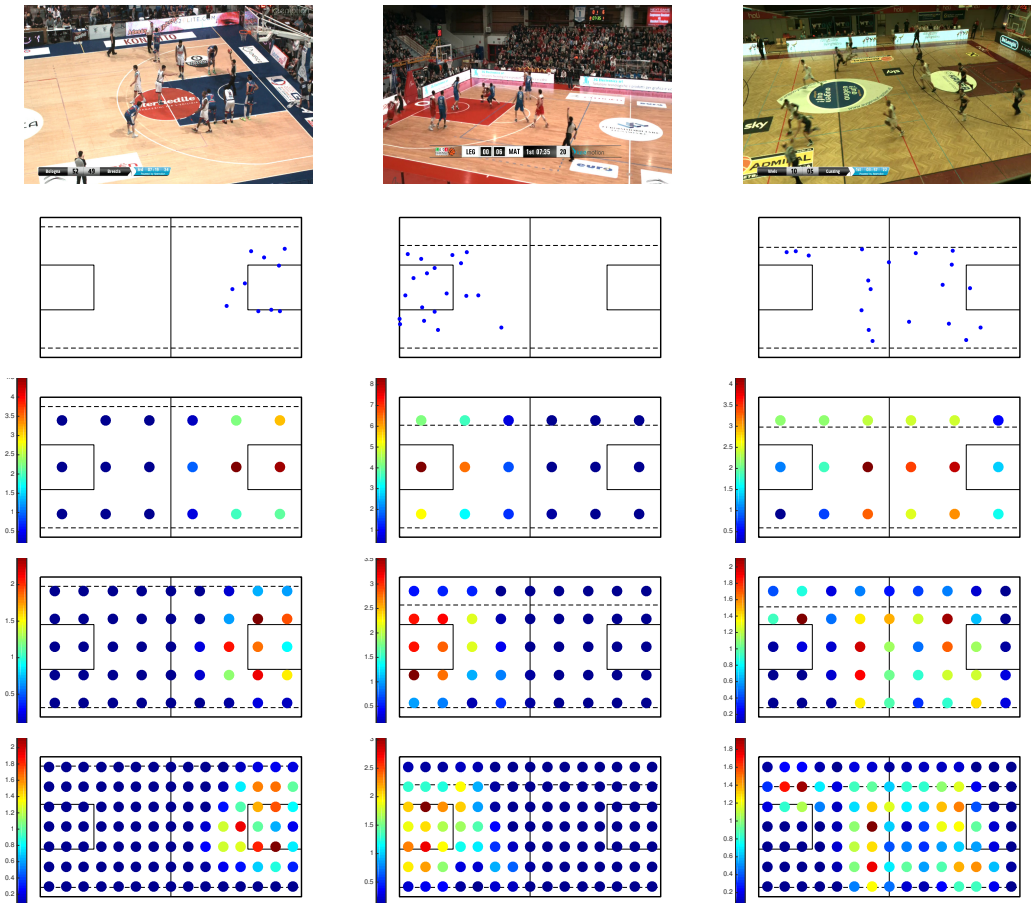
(a) (b) (c)

(d) (e) (f)

Fig. 4.3 Hough Transform and Filtering applied on several randomly chosen frames.

4.2.2 Activity Map

Figure 4.4 shows the computation of the Activity Map for three randomly chosen frames from each stadium. The Activity Map is computed for three different cell map sizes $M \times N$: 6×3, 9×5 and 15×7. The detections on which this process is applied are obtained with the Hough Transform and the Filtering. We can clearly see that a larger map gives a much more accurate representation of players distribution. On Figure 4.4c, we can see that, despite the use of the Hough Transform and the Filtering strategy, there are still some noisy detections in the upper left part. The corresponding Activity Map is therefore not totally correct. This is why the accuracy of this map is strongly related to the quality of the detections.



(a) (b) (c)

Fig. 4.4 Activity Map computed for three cell map sizes.

4.2.3 Velocity Map

Figures 4.5 and 4.6 shows some examples of velocity maps. These maps were computed on detections preprocessed with the Hough Transform and the filtering strategy. A *gap*[6] of 3 was chosen. We can see that the performance is generally better when the players are moving, such as in Figure 4.5b, because the assignment process using the Hungarian Algorithm is less trustworthy when the players are grouped and static. For example, on Figure 4.5a, some movements are detected although the players are immobile. Even if a lot of detections are missing or left unassigned, such as in Figure 4.3c, 4.3e and 4.3f, the few remaining assigned detections can still give a representative Velocity Map. When looking at the assignments in Figure 4.3c, players may seem to be slower than in Figure 4.3b, althoug they are not. The reason is that the time gap between two frames is much more smaller in 4.3c. As you can see on the right part of the court in Figure 4.3d, some inaccuracies in players position may lead to disproportionate speeds and therefore to incorrect velocity in some cells.

4.3 The Feature Dataset in numbers

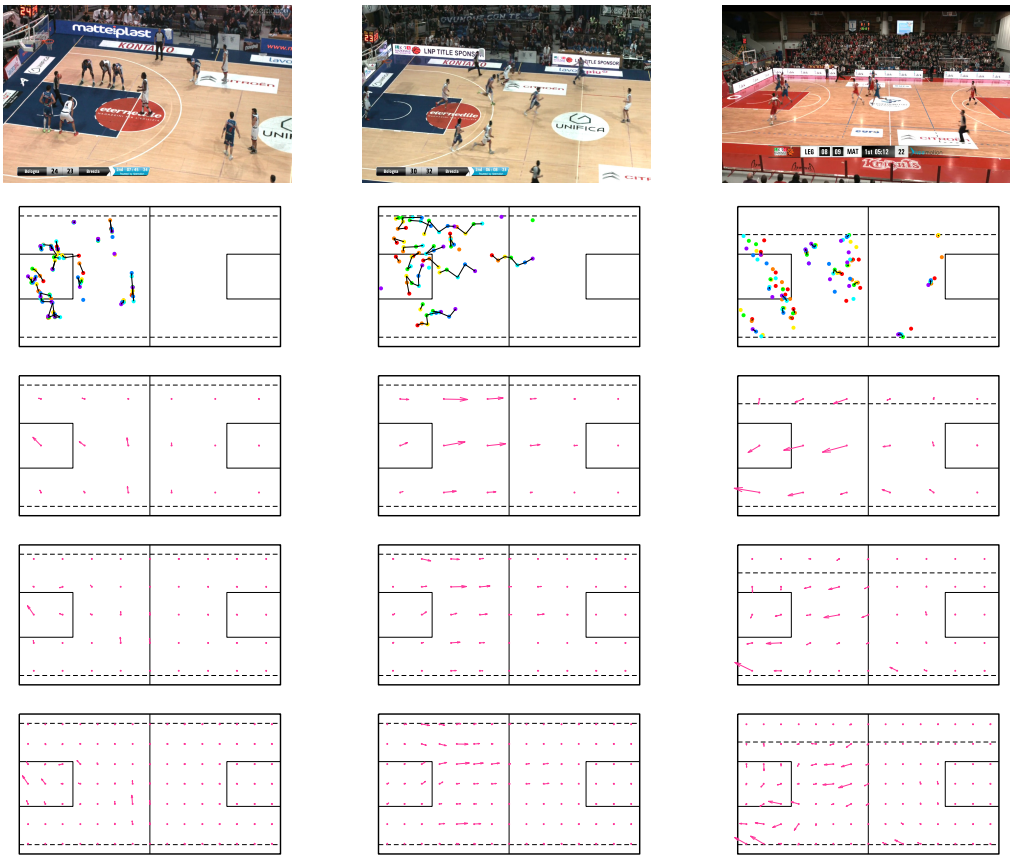
Three datasets with different cell map sizes were built using the feature extraction process. As a reminder, there are three parameters to set when we build a cell map: the size $M \times N$ of the grid and the size σ of the Gaussian. The three different cell maps we will use were built with the following parameters:

1. $M = 6$, $N = 3$, $\sigma = 290$.
2. $M = 9$, $N = 5$, $\sigma = 160$.
3. $M = 15$, $N = 7$, $\sigma = 130$.

These parameters were chosen arbitrarily. We will see in the next Section how to choose the cell map giving the best prediction accuracy.

As explained in Section 1.3.4, the annotation process is very slow and it's therefore very complicated to build a large and complete dataset. We will therefore try to artificially enlarge the Feature Dataset by generating, for each example, the three corresponding symmetric instances. This means that, given a cell map, we will generate a first symmetric instance along the x axis, a second symmetric instance along the y axis and finally a third symmetric instance by point symmetry with the center of the

[6]The Speeds are computed using the Hungarian Algorithm applied on $2 \times gap/1$ successive frames. For more details on the *gap* and the color code used, see Section 2.4.2.



(a) (b) (c)

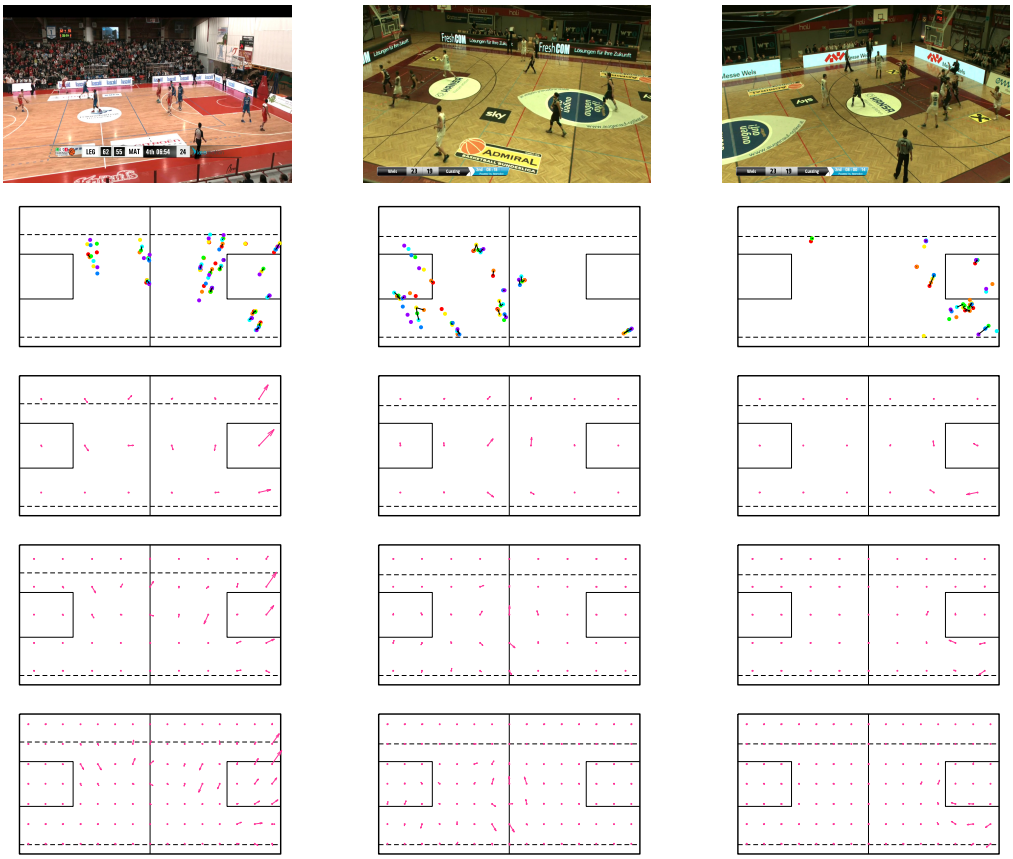
Fig. 4.5 Velocity Map computed for three cell map sizes.

court. With this process, we can generate four times more examples, for a total of 5404 instead of the 1351 original ones.

4.4 Parameters Tuning

In this Section, we will explain how the parameters of the three machine learning algorithms described in Section 3 were tuned and how we chose the size of the cell map[7]. The training was done using three R[8] packages: RSNNS[9] for MLP, *e*1071[10] for SVR and *randomForest*[11] for RF. As a reminder, the parameters to be optimized are *Gamma* and the *Cost* for SVR, the number of *hidden units* and the *maximal number*

[7] See Section 2.6.
[8] https://www.r-project.org
[9] https://cran.r-project.org/web/packages/RSNNS/index.html
[10] https://cran.r-project.org/web/packages/e1071/index.html
[11] https://cran.r-project.org/web/packages/randomForest/index.html



(a) (b) (c)

Fig. 4.6 Velocity Map computed for three cell map sizes.

of iterations for MLP, the *number of trees* in the forest and the *size of the subset of features* processed at each node for the RF.

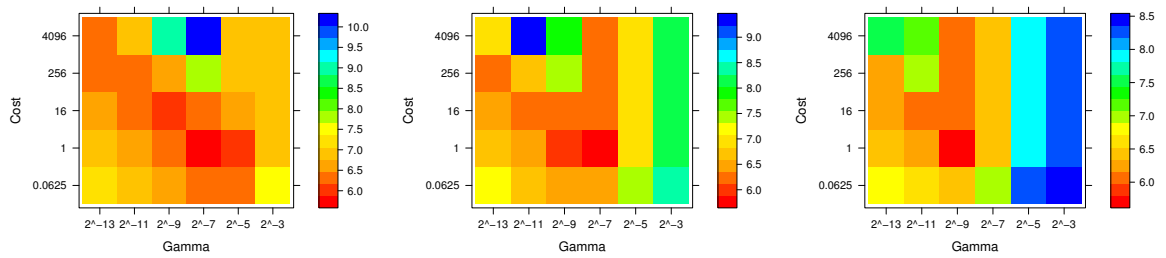
Figures 4.7 shows the tuning process for SVR on three different cell map sizes. The values indicates the Root Mean Square Error $RMSE$ in meters of the predictions made with the test set using ten fold cross validation[12]. The RMSE is defined as follows:

$$RMSE = \sqrt[2]{\frac{\sum_{t=1}^{n}(\hat{y}_t - y_t)^2}{n}} \tag{4.1}$$

, where $\hat{y}$ are the x-components of the predicted COA and y_t the corresponding annotations.

There are three main facts to notice with these results. Firstly, the best parameter combination (red square on the Figure) is roughly the same for each map size. Secondly, the best performance is also the same for each size. Similar tests were made for the

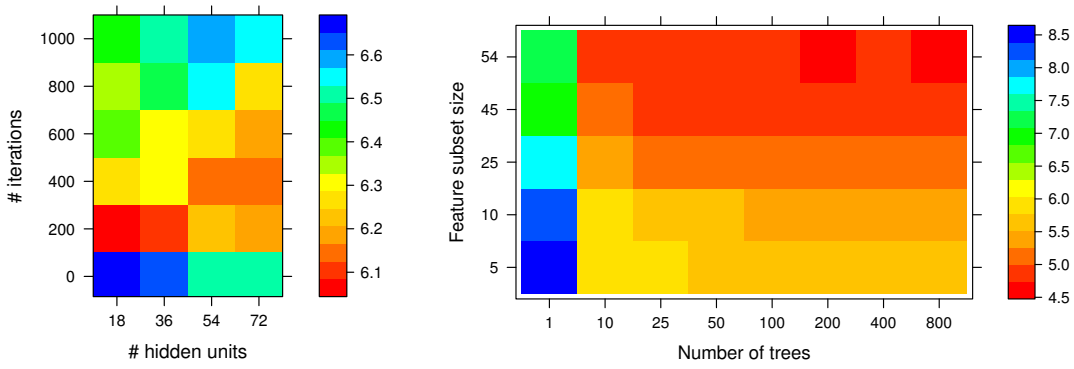
[12]See Section 3.5.



(a) 6×3 cells (b) 9×5 cells (c) 15×7 cells

Fig. 4.7 Tune gamma and cost parameters of SVR on different dataset. The criterion used is the RMSE expressed in meters.

Random Forest and the Multilayer Perceptron, and the conclusion is the same. We will therefore use the smaller map, because it allows faster training. Figure 4.8a and 4.8b shows the tuning process for the MLP and the RF on the 6×3 map. As a reminder, the size of the feature vector is three times the size of the cell map, there are therefore 54 features per examples for the 6×3 map.



(a) MLP (b) RF

Fig. 4.8 Parameters tuning of MLP and RF, with the RMSE in meters used as criterion, on the 6×3 cell map.

The list below summarizes the results of this process and shows the parameters combination we will use for each algorithm for the rest of this Chapter.

- **SVR**: $Gamma = 0.0078125$, $Cost = 1$.
- **MLP**: $\#\ Hidden\ units = 36$, $\#\ iterations = 200$.
- **RF**: $\#\ trees = 200$, $Featuresubsetsize = 54$.

Table 4.2 RMSE in meters for different algorithms and preprocessing combination.

Algorithm	No filtering		Filtering	
	No Hough	Hough	No Hough	Hough
SVR	5.96	5.94	6.21	5.91
MLP	6.25	6.21	6.38	6.05
RF	5.10	4.87	4.95	**4.84**

4.5 Results and Perspectives

Table 4.2 shows the RMSE in meters[13] for each combination of regression method and feature extraction process. These results show that the two preprocessing steps do not have a great impact on the performance, even if the best result for each algorithm is obtained with the Hough Transform and the Filtering activated.

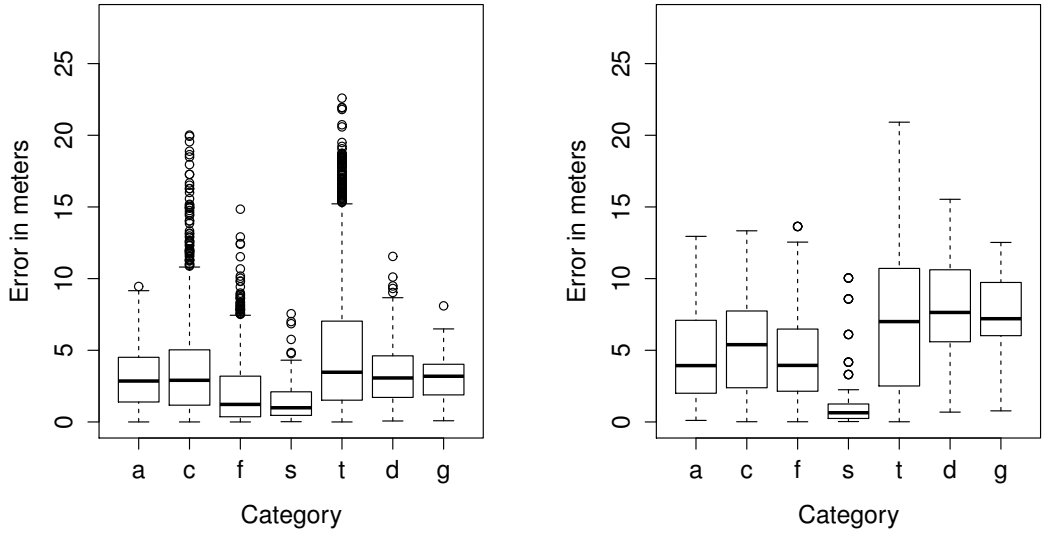
In order to asses the performance of the best algorithm, we will compare it to the results obtained with a simple strategy, which will therefore be used as a baseline. This strategy, called the *Centroid*:

Definition 4.5.1. The *Centroid* strategy consists in approximating the Center of Attention at a given frame as the players center of mass.

According to the results of Table 4.2, the best regression accuracy is reached with **Random Forest**, with a RMSE of $4.84m$ and a mean error of $3.47m$. The **Centroid** strategy gives a RMSE of 7.14 meters and a mean error of 5.85 meters. The RF gives therefore a much better prediction of the COA than the *Centroid*.

Figure 4.9 shows a comparison of the performance of the two approaches in each category. Each box indicates the error distribution in a specific category. The error expressed here is simply the absolute difference in meters between the prediction and the annotated Center of Attention. The points indicate outliers. We can clearly see that the trained Random Forest outperforms our baseline in every category[14], except for the Jump bal (s). The two strategies work pretty well in this category because the Center of Attention is easy to predict: it is generally exactly on the players center of mass. We can also notice that the worst results are obtained in counter attack (c) and transition (t) phases. This result was expected because these two phases are the one where the players are the most spread on the court, which makes the COA very difficult to predict. The results with Random Forest in attack phase (g) and false

[13] As a reminder, a basketball court is generally 28 meters long.
[14] As a reminder, the category corresponding to each label is indicated the Table 4.1.



(a) Random Forest (b) Centroid

Fig. 4.9 Error of the COA prediction per category for the best supervised learning strategy and the baseline.

detections situation (d) are also much more better than the Centroid strategy, because these are situation with a lot of noisy detection, which makes the centroid much more different than the real players center of mass and therefore an even worse estimation of the COA.

We can see on Figure that a lot of predictions are still 5 meters away or more from the real COA, which means that this method seems not precise and stable enough to be incorporated in a real system from now on. A more precise study should be made to investigate further the biggest error obtained and try to see if these errors are caused by bad features or simply by bad interpretation of these features by the regression. In the later case, further work should be made to understand which information is missing and to incorporate this information to the feature vector. We could also try to change the formulation of the problem. For example, we could try to predict the distance between the center of attention and the players center of mass, instead of predicting directly the COA. We could also try to classify the examples in one of the seven categories and then predict the Center of Attention using heuristics based on the centroid and the characteristics of the corresponding game phase. For example, in a counter attack phase, the COA is generally in front of the group of players, while in a transition phase, the COA is generally at the back of the group of players.

We would also try to compare the results obtained with different dataset sizes, to see if the performance can be improved using a larger dataset.

Conclusion

In this work, we proposed a supervised learning approach to predict the Center of Attention from noisy players detections on a basketball court. The main goal was to predict this Center of Attention for some particular tricky situations, where much more classical strategies failed to give a visually pleasing framing.

We first started to show how to built robust and meaningful feature vectors using the noisy collected players detections. These feature vector were composed of an Activity Map, representing players distribution on the court, and a Velocity Map, representing players motion on the court. The results showed that this feature extraction process was quite effective : the feature vector are a good representation of players distribution and motion on the court. However, we have seen that the performance of this process was strongly related to the quality of the detections. Most of the bad results obtained are due to noisy or missing detections.

We then saw how to predict the manually annotated Center of Attention using a dataset composed of the feature vectors together with state-of-the-art machine learning algorithms. We saw that the results obtained are much more accurate than the simple strategy consisting in approximating the Center of Attention with players centroid. However, there's still a non negligible portion of the predictions which are too far from the real COA, which means that the regression seems not precise enough to be used in a real system.

References

[1] Ariki, Y., Kubota, S., and Kumano, M. (2006). Automatic production system of soccer sports video by digital camera work based on situation recognition. In *Eigth IEEE International Symposium on Multimedia (ISM 2006), 11-13 December 2006, San Diego, CA, USA*, pages 851–860.

[2] Bishop, C. (2006). *Pattern Recognition and Machine Learning*. Information Science and Statistics. Springer.

[3] Bosch, A., Zisserman, A., and Muñoz, X. (2007). Image classification using random forests and ferns. In *ICCV*, pages 1–8. IEEE.

[4] Chang, C.-C. and Lin, C.-J. (2011). Libsvm: A library for support vector machines. *ACM Trans. Intell. Syst. Technol.*, 2(3):27:1–27:27.

[5] Chen, J. and Carr, P. (2014). Autonomous camera systems: A survey. *Twenty-Eighth AAAI Conference on Artificial Intelligence*.

[6] Chen, J. and Carr, P. (2015). Mimicking human camera operators. In *2015 IEEE Winter Conference on Applications of Computer Vision, WACV 2014, Waikoloa, HI, USA, January 5-9, 2015*, pages 215–222.

[7] F. Chen, C. D. V. (2010). Personalized production of basketball videos from multi-sensored data under limited display resolution. *Computer Vision and Image Understanding*, 114(6):667–680.

[8] Hoang M Le, Andrew Kang, Y. Y. P. C. (2016/6/3). Learning online smooth predictors for realtime camera planning using recurrent decision trees. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[9] Hsu, C.-W., Chang, C.-C., and Lin, C.-J. (2003). A practical guide to support vector classification. Technical report, Department of Computer Science, National Taiwan University.

[10] ISPGroup (2012). Apidis project. http://sites.uclouvain.be/ispgroup/index.php/Softwares/APIDIS.

[11] Kim, K., Lee, D., and Essa, I. (2012). Detecting regions of interest in dynamic scenes with camera motions. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society.

[12] Liaw, A. and Wiener, M. (2002). Classification and regression by randomforest. *R News*, 2(3):18–22.

[13] Liu, J. and Carr, P. (2014). *Detecting and Tracking Sports Players with Random Forests and Context-Conditioned Motion Models*, pages 113–132. Springer International Publishing, Cham.

[14] Parisot, P., Sevilmiş, B., and De Vleeschouwer, C. (2013). *Training with Corrupted Labels to Reinforce a Probably Correct Teamsport Player Detector*, pages 682–694. Springer International Publishing, Cham.

[15] Piraux, T. (2011). Contrôle automatique d'une caméra PTZ pour le suivi de cibles multiples: application au basket-ball. Master's thesis, Université Catholique de Louvain.

[16] SDL (2016). Simple DirectMedia Layer API. https://www.libsdl.org.

[17] Smola, A. J. and Schölkopf, B. (2004). A tutorial on support vector regression. *Statistics and Computing*, 14(3):199–222.

[18] Venables, W. N. and Ripley, B. D. (2002). *Modern Applied Statistics with S*. Springer, New York, fourth edition. ISBN 0-387-95457-0.

[19] Wang, X., Ablavsky, V., Shitrit, H. B., and Fua, P. (2014). Take your eyes off the ball: Improving ball-tracking by focusing on team play. *Computer Vision and Image Understanding*, 119:102–115.