The Visual Computer (2024) 40:2695–2711
https://doi.org/10.1007/s00371-023-02972-1

ORIGINAL ARTICLE

PlayNet: real-time handball play classification with Kalman embeddings and neural networks

Omar A. Mures[1,2] · Javier Taibo[1] · Emilio J. Padrón[1,3] · Jose A. Iglesias-Guitian[1,3]

Accepted: 13 June 2023 / Published online: 7 August 2023

Abstract
Real-time play recognition and classification algorithms are crucial for automating video production and live broadcasts of sporting events. However, current methods relying on human pose estimation and deep neural networks introduce high latency on commodity hardware, limiting their usability in low-cost real-time applications. We present PlayNet, a novel approach to real-time handball play classification. Our method is based on Kalman embeddings, a new low-dimensional representation for game states that enables efficient operation on commodity hardware and customized camera layouts. Firstly, we leverage Kalman filtering to detect and track the main agents in the playing field, allowing us to represent them in a single normalized coordinate space. Secondly, we utilize a neural network trained in nonlinear dimensionality reduction through fuzzy topological data structure analysis. As a result, PlayNet achieves real-time play classification with under 55 ms of latency on commodity hardware, making it a promising addition to automated live broadcasting and game analysis pipelines.

Keywords Handball play classification · Real-time multimedia · Neural networks · Kalman filtering · Dimensionality reduction

1 Introduction

Small sports teams and organizations often struggle with broadcasting their games and analyzing game data due to the high costs associated with the necessary manpower, logistics, and hardware. Automatic live video production with commodity hardware and no human intervention offers a low-cost solution to this problem. However, to accomplish it, real-time play recognition is a crucial task that needs to be addressed before the automatic broadcasting system can operate reliably.

✉ Omar A. Mures
omar.alvarez@udc.es

Javier Taibo
javier.taibo@udc.es

Emilio J. Padrón
emilio.padron@udc.es

Jose A. Iglesias-Guitian
j.iglesias.guitian@udc.es

[1] Universidade da Coruña, A Coruña, Spain

[2] CINFO, A Coruña, Spain

[3] CITIC - Centre for ICT Research, A Coruña, Spain

The highly unpredictable nature of player movements in invasive sports such as handball and the variability of the background scenarios present many different issues when applying classical computer vision algorithms. In this context, learning-based approaches have emerged as promising alternatives to enhance automatic video production and game data analysis robustness. However, state-of-the-art methods often address produced video content instead of raw video footage, assume a broadcasting signal delay, use specialized tracking hardware, or operate in fully offline regimes.

This paper presents PlayNet, a novel approach for handball play classification that can work with only raw video footage and operate in real time using GPU commodity hardware. Consequently, PlayNet can guide physical cameras in real-time, such as motorized PTZs, or switch views from multiple fixed cameras, as required for the automatic live broadcasting of sporting events. In summary, the main contributions of this work are:

– We propose Kalman embeddings, a low-dimensional representation for a game frame that leverages Kalman filtering and a neural network trained to perform nonlinear dimensionality reduction. This representation has

demonstrated its effectiveness in real-time play classification throughout our experiments.
- We introduce the ONTV (Orthographic Normalized Top-View) space, a normalized representation of the game area. This space effectively decouples the game state from camera configurations, enabling PlayNet to accommodate various camera setups and courts.
- We have developed PlayNet, a practical handball play classifier implemented using the abstractions above. PlayNet has been successfully deployed in a production environment, enabling automatic live broadcasting.

The rest of the paper is organized as follows: Sect. 2 reviews the state-of-the-art in the automatic classification of game states in sports videos. Section 3 introduces the domain and the fundamentals of the problem at hand. The PlayNet architecture is first outlined in Sect. 4, whereas its main components are described in Sects. 5 and 6. Implementation details are shown in Sect. 7, and performance and accuracy are evaluated and discussed in Sect. 8, comparing them to other modern approaches on a sample dataset and discussing system limitations and future work. Finally, in Sect. 9, we provide our conclusions.

2 Related work

Shih et al. [35] have published a survey that provides a comprehensive and in-depth interpretation of the potential insights gained through content-aware sports video analysis. Another survey by Cuevas et al. [11] discusses techniques and applications for analyzing sports video sequences. They include an in-depth discussion about event detection and game analysis. Despite being focused on soccer, it remains an interesting read related to our work. Other approaches [28, 30, 40] rely on specialized tracking hardware, such as inertial measurement units (IMUs), to detect semantic events and improve tracking. This section restricts our discussion to closely related works with similar goals and avoids expensive capture setups or tracking hardware. Next, we focus our effort on contextualizing the principal contributions of our work within the current state-of-the-art.

Semantic event detection in sports. Semantic event detection research focuses on automated processes for identifying and classifying relevant events that may occur during a sporting event. This information provides a better understanding of the game, and can be used for various purposes such as game analysis, player evaluation, and generating highlights and summaries of the game for broadcasting or replay. Most of the proposed automated techniques for detecting semantic events in sports rely on specific cues in the video footage [11, 12, 29, 39]. These cues can include captions, banners, scene cuts, and various visual elements added during the live broadcast or in post-production. However, it is important to note that this contextual information is typically absent from the raw video footage. As a result, automated sports broadcasting systems like ours cannot depend on these cues to detect and classify events.

Other works address raw video footage by performing offline analysis after the end of the game or a fragment to be produced [14, 17, 26, 27, 36], but none of them accomplished real-time game state classification. In contrast, Schlipsing et al. [34] analyze raw video footage of soccer matches in real time, but their focus was player classification rather than semantic event detection. Our work introduces a handball play classifier working in real-time on raw video footage that can detect meaningful semantic events for automatic broadcasting.

Play state recognition for automatic video production. Play state recognition is a prerequisite for automatic broadcasting systems and video production [6, 11, 12, 39]. The connection between semantic event detection and play state recognition is that the detection of specific events can provide valuable information that can be used to determine the current play state of the game. For example, Quiroga et al. [32] achieve automatic video production of basketball games using neural networks. However, authors report up to 10 s. of latency since their system operates in the cloud, which may be acceptable for deferred live broadcasting, but not for live production using PTZ cameras [5]. Our work aims to fill the real-time game state classification gap for automatic live broadcasting systems.

Supervised dimensionality reduction. Our method detects and tracks game agents in the court, representing their movement patterns in a normalized coordinate space. However, the high dimensionality of this representation poses computational challenges. To address this, we employ dimensionality reduction techniques for faster and more accurate results. In previous studies such as [21, 22] this problem was addressed using a two-dimensional extension of random projection for feature extraction, and a two-dimensional discrete cosine transform for feature fusion. While exploring more recent relevant literature, promising techniques such as [1, 2, 25, 31] were studied, but they required processing all data points for embedding new ones, rendering them unsuitable for real-time applications. Another noteworthy approach is [2], which showcases an interactive visual analytics system capable of visualizing time-series data utilizing several dimensionality reduction techniques. However, this approach was unsuitable for our problem as it was primarily developed for visualization rather than classification. Among the options available, IVIS [37] appears to be the most suitable approximation for our specific use case. It employs a Siamese network architecture to embed data points and, following testing, has demonstrated its capability to perform in real time. Notably,

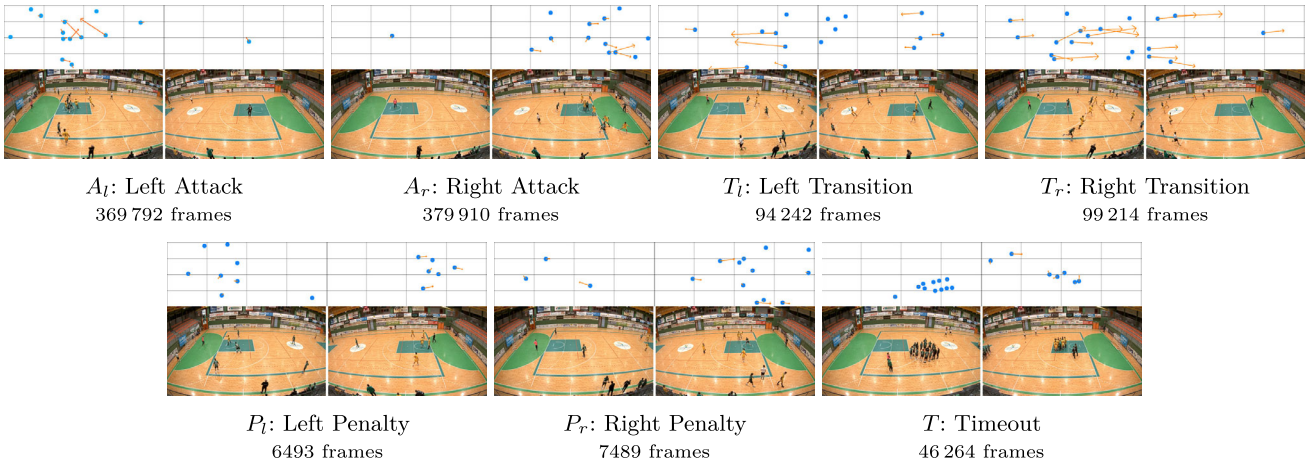


Fig. 1 Example frames showcasing the seven classes considered in PlayNet, along with the number of frames of each class included in the dataset. The corresponding ONTV representations are shown above the video frames

none of the evaluated approaches have surpassed our method in terms of performance and accuracy in our use case.

Handball datasets. Most existing sports event detection datasets predominantly use video data and cater to specific sports, leaving handball with a paucity of publicly available datasets. In addition, although some datasets annotate player actions [17], game state annotation is generally absent. A notable exception is EIGD-H [3], a multimodal benchmark dataset containing synchronized video, audio, and positional data for handball games. Their authors propose a unified taxonomy of high-level ball-centered events in invasion games like handball. In addition, their dataset contains handcrafted frame annotations by domain experts, adding up to a total of 125 min. of playing time. Our work introduces an open dataset for handball play classification released under the CC-BY-NC license at Zenodo.[1] Our dataset comprises 1 million labeled game states and their associated positional data, corresponding to approximately 695 min of playing time. However, we cannot release the raw video frames for commercial and legal reasons. Additional information about our dataset is disclosed in Sect. 8.

3 Problem statement

Our primary focus is the real-time classification of meaningful states in handball games, using only raw video footage and commodity hardware. This task is essential for the low-cost, human-free seamless operation of production video cameras while ensuring minimal latency in the live broadcast.

Handball play classification. From a live broadcast production perspective, we consider the following possible states in a handball game (see Fig. 1): left/right attack (A_l/A_r), left/right transition (T_l/T_r, counter-attacks are included here), left/right penalty (P_l/P_r) and timeout (T). Thus, the problem could be summarized as classifying the current state of the game into one of the seven aforementioned classes. The algorithm takes as input a full-court video stream that could be obtained by just a single camera or an array of cameras arranged in any layout that fully covers the court.

Camera layout abstraction. Production camera setups for sports broadcasting can change due to multiple factors, such as the number of cameras, budget constraints, and unique characteristics of each indoor arena. As a result, even when an agent occupies the same position on the court, the obtained agent positions can differ. Variations in camera location, perspective, lens configuration, sensor resolution, and other factors cause these discrepancies. The problem becomes even more challenging when using multiple cameras with a custom layout to cover the entire playfield. In addition, creating a robust algorithm for game state classification faces a generalization problem due to the variability in data resulting from tasks like object detection and tracking. These properties can significantly differ across camera sources, mosaic layout setups, or camera parameter configurations.

Working hypothesis. In this work, we propose that tracking the ball and analyzing the movement patterns of individuals on the court might be adequate for classifying the various states of a handball game. Our system considers 16 game agents and the ball, including 14 players divided into two teams, and two referees. We do not explicitly differentiate between players of the two teams or players and referees, except for the ball, which is distinguished from other agents due to its particular significance.

[1] https://doi.org/10.5281/zenodo.7180366.

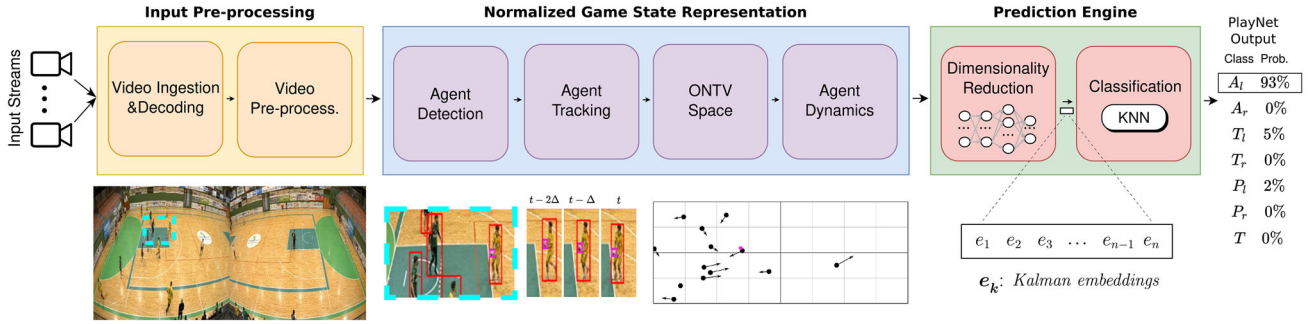


Fig. 2 System overview of PlayNet. The upper diagram shows the data flow in the system, starting with the video ingestion from an arbitrary number of cameras and finishing with PlayNet's output, the probability distribution for the current frame over the seven classes considered by the system. Below the diagram, examples of the data traversing the pipeline are shown: first, a frame obtained from a 2-camera setup; next, bounding boxes from agent detection and details from agent tracking across the last three frames; then, the ONTV representation for the current frame; and finally, the Kalman embeddings ($e_1 \dots e_n$) used to classify the frame and produce the output

4 System overview

This section provides a general overview of the proposed solution for real-time handball game state classification (Fig. 2). Our method, named PlayNet, consists of the following main stages.

Calibration. The system requires an initial calibration step for all cameras in the mosaic layout to ensure the proper operation of PlayNet. Using in-house software, camera operators manually identify the field borders and the region of interest within each camera image. This step significantly enhances agent tracking and effectively mitigates most detection issues.

Input pre-processing. In this stage, the system decodes and optimizes the data from each input video stream for efficient processing in subsequent phases. Standard image processing operations are applied to each frame, including pixel format conversion, resizing, cropping, and color normalization. These operations generate batches of downsampled images, maximizing the object detection and tracking throughput.

Normalized Game State Representation. During this stage, the system identifies the relevant handball game agents in the frame and utilizes their spatiotemporal information to create a unified game state representation within a common coordinate space. Our proposed approach employs an Orthographic Normalized Top-View (ONTV) space, described in detail in Sect. 5. In addition, to manage the positions and velocities of game agents in the ONTV space, we implemented a custom Kalman filter.

Prediction engine. In the final stage, our system performs nonlinear dimensionality reduction based on manifold learning techniques and fuzzy topological data analysis. This step results in a condensed version of the game state representation, referred to as Kalman embeddings (described in Sect. 6), ultimately serving as input for a classifier responsible for the final decision in play classification.

In Sect. 5 and 6, we detail the two main stages of the PlayNet architecture. In Sect. 7 we provide some implementation details required to replicate the inner-workings of the PlayNet system.

5 Normalized game state representation

In this stage, according to our working hypothesis, our system aims to obtain a normalized and unified representation of the game state, summarizing its dynamics from the perspective of a handball play classifier. This module receives one or multiple streams of images from cameras arranged in a mosaic layout, ensuring comprehensive coverage of the play area. This work primarily focuses on a setup with two wide-angle cameras horizontally shifted to cover the entire playfield (see Fig. 3).

We propose utilizing an object detector followed by an object tracker to identify and track each game agent across frames. As a result, the location coordinates of the game agents (including the game ball) are obtained in separate coordinate spaces corresponding to each camera in the mosaic layout. To create a unified and normalized representation of the game state, PlayNet projects all positional information into a consistent coordinate space.

To further enhance the game state representation, a Kalman filter is employed. This filter corrects precision errors in agent locations detected by the system and incorporates velocity vectors, capturing movement speed and direction. In the following subsections, we provide detailed explanations for our object detection and tracking methods, the coordinate space utilized in PlayNet, and the Kalman filter implementation.

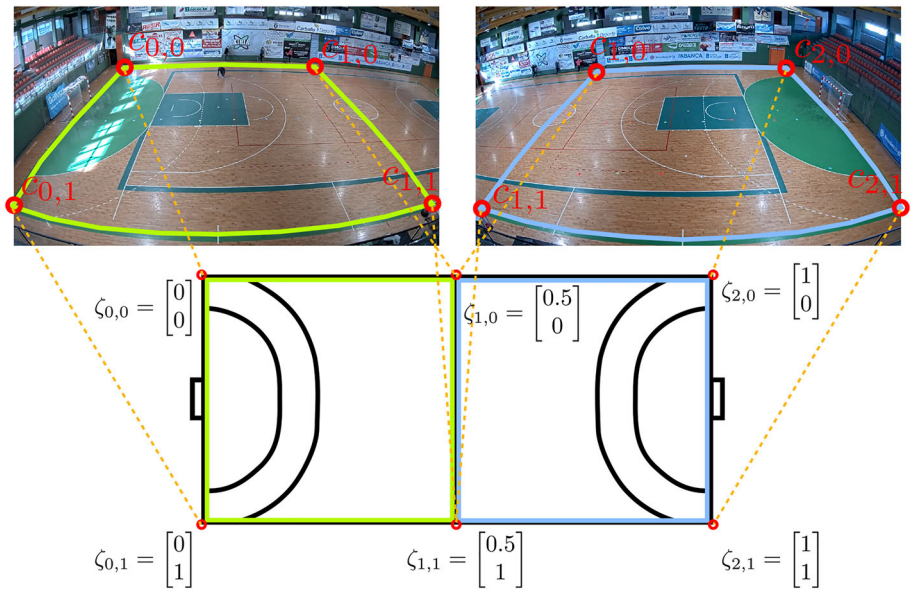


Fig. 3 Four-point transform used to convert coordinates from per-camera local image spaces into ONTV space in our experimental setup. Camera frames are shown at the top of the figure and the corresponding ONTV space is depicted at the bottom. The corners of the court are drawn in red and its limits in green and cyan. Dashed lines show the mapping between image space and normalized corners: $c_{x,y} \Rightarrow \zeta_{x,y}$. Points in ONTV space are represented as column vectors

5.1 Agent detection

The agent detection module is a fundamental piece for the proper operation of PlayNet. Having accurate estimates of game agent positions is critical for our game state representation. The main requirement imposed by our system is that a good balance between detection accuracy and speed is necessary. As long as this requirement is fulfilled, we could replace the object detector module with a better one, and the whole system should benefit from it. When starting the design of PlayNet, we found that YOLOv4 [4] was a reasonable choice as it fulfills the need to ascertain all game agent locations at every frame with noticeable robustness and speed.

At this stage, we identify the ball as a separate case of an agent that requires special treatment, e.g., during the tracking stage. In terms of considering wrong external game agents, i.e., coaches or players on the bench, as actual in-game agents, which would introduce additional noise in our game representations, we opted to prefilter them by simply using a strict per-camera region of interest approach.

In our system, the object detector receives per-frame image data that should cover the whole playfield and returns a vector of bounding boxes, one for each game agent detected, determined by their lower-left corner position x, y, and their dimensions $w \times h$.

5.2 Agent tracking

To provide a comprehensive representation of the game agent dynamics that accurately reflects the state of a handball game, PlayNet incorporates estimations of both the location and instant dynamics of the game agents, including movement direction and speed. Therefore, it is crucial to establish the correspondence between game agents across consecutive frames, precisely the objective of the agent tracking module in PlayNet. This module analyzes the position of in-game agents across frames of an input video stream estimating their trajectories and speeds. Specifically, we perform matching operations on consecutive frames for all the detected agents.

The Euclidean-Jaccard index. We adapt the Intersection over Union (IoU), also known as the Jaccard index [33], a widely-used distance metric for object detection, to incorporate the Euclidean distance between the involved bounding boxes (BBs). The Jaccard index quantifies the similarity between two BBs (A and B) by comparing the ratio of their overlapped areas to the area of their union, expressed as $J(A, B) = \frac{A \cap B}{A \cup B}$. However, in fast-paced games like handball, where there may be no overlap between consecutive frame BBs, the Jaccard index fails to differentiate between closer and farther BBs. To address this limitation, we propose extending the Jaccard index as a dissimilarity metric by considering the Euclidean distance between the centroids of consecutive frame BBs:

$$J_\delta(A, B) = \delta(c_A, c_B)(1 - J(A, B)), \tag{1}$$

Here, $\delta(c_A, c_B)$ represents the Euclidean distance between the two BB centroids c_A and c_B. In our scenario, to identify whether two BBs correspond to the same game agent, we rely on two user-defined parameters: a minimum IoU (J_{min}) and a maximum distance between centroids (δ_{max}), which should be tailored for each specific application.

Caching scheme for agent tracking. Our agent tracking module maintains a cache C with relevant information for the detected agents. Initially, a new agent A_0 is detected, and a new entry is created in the cache. In subsequent frames,

when a new agent A_i is detected, the system searches for a matching entry by selecting the entry A_j that minimizes $J_\delta(A_i, A_j)\ \forall j \in C$, satisfying $\delta(c_i, c_j) \leq \delta_{max}$ and $J(A_i, A_j) \geq J_{min}$. If A_j is a valid match, A_i and A_j are considered the same agent, and the cache entry is updated accordingly (replacing the position and BB information of A_j with the corresponding data from A_i). If no match is found, a new entry is created in C for A_i.

Cache entries that are not matched can persist for several frames, up to a maximum of f_{max}, to mitigate the effects of missing agent detections over consecutive frames. It is important to note that legitimate agents may not be properly detected for various reasons, such as the potential loss of agents during transitions from one part of the field covered by one camera to another or due to occlusions. In such cases, PlayNet leverages this cache and the agent dynamics computed by the Kalman filter (described in Sect. 5.4) to estimate the position of the agent in the playing field. By updating agent positions with an estimate, PlayNet effectively mitigates the problem of missed detections.

Considering the unreliable nature of network connections and possible variations in processing time, which may result in skipped frames, the information passed to the following stages of the PlayNet pipeline always includes an attribute Δt. This attribute accounts for possible frame skips and represents the elapsed time between the current and the last processed frame, enabling accurate computation of agent dynamics and predictions.

5.3 Orthographic normalized top-view space

We introduce the Orthographic Normalized Top-View (ONTV) space, a decoupled representation of the handball game state from a specific camera setup. ONTV resembles a bird's-eye view perspective of the entire field (Fig. 3). This 2-D coordinate space ranges from 0.0 to 1.0 in each dimension, providing a normalized abstraction of agent positions across different camera sources. By utilizing ONTV, we can abstract agent positions from the specific image space coordinates of each camera, making our solution adaptable to various arena types and camera layout configurations. Additionally, employing a unified representation eliminates the need for retraining or creating different versions of the play classifier algorithm for different camera layouts.

Pre-requisites to operate in ONTV space. The detected agent positions are initially expressed in local pixel coordinates, i.e. the image space of a given camera source stream which may belong to a more complex layout with an arbitrary number of cameras N. These pixel coordinates correspond to the lower-center point of the agent BBs obtained during the detection stage (Sect. 5.1). To ensure accurate operation in ONTV space, our calibration stage requires the camera setup to meet two criteria: i) comprehensive coverage of the entire play field across all camera sources, and ii) clear limits for each camera to avoid significant overlap with other sources (as depicted in Fig. 3 for our two-camera setup). Any agents detected outside of this enclosure are disregarded during the detection stage.

ONTV perspective transformation. To transform coordinates from the image space of a camera source into ONTV space, our system employs perspective transformations [10]. The perspective transformation for each camera is described by a 3×3 homography matrix, $\mathbf{H}_c$, and a scaling factor, u, that accounts for the projection distance.

The region of interest covered by camera source c is defined by its four corner points within the court: $(c_{tl}, c_{tr}, c_{bl}, c_{br})$, expressed in the image space of that camera. We define a four-point perspective transformation to convert these corner points into ONTV space. By solving a system of linear equations for each camera source, we obtain the corresponding $\mathbf{H}_c$ and u, enabling us to transform any point in the image space of a camera into ONTV space (see A.1 for the complete formulation).

Once we compute $\mathbf{H}_c$ and u for each camera source, we can convert the coordinates (p_x, p_y) of an agent in its corresponding camera image space to the final ONTV coordinates (o_x, o_y) using the equation:

$$\begin{bmatrix} o_x \\ o_y \\ 1 \end{bmatrix} = \frac{1}{u} \cdot \mathbf{H}_c \cdot \begin{bmatrix} p_x \\ p_y \\ 1 \end{bmatrix} \tag{2}$$

From this point on, the system utilizes the converted ONTV coordinates for all agents in the field. For each processed frame, k, the output from this step is a collection of vectors containing the ONTV space position for each agent, as well as the elapsed time since the previously processed frame.

5.4 Agent dynamics

The last component of the Normalized Game State Representation stage is a Kalman filter that estimates the agent information needed to characterize the game state (position, direction, and speed).

Kalman filtering [18, 45] is a well-known solution to estimate the current state of a linear dynamic system based on inaccurate and uncertain measurements, as well as a widely adopted technique to predict future system states based on past estimations. The Kalman filter component in PlayNet is applied to each agent. It receives the output vectors from Sect. 5.3 as observations, i.e., the vectors with agent positions in ONTV space for the current frame, k. As output, the filter produces the core of our embeddings, estimated positions

(updating/correcting the received measurements), and velocities (direction and speed) in ONTV space for each detected agent in the current frame.

The carried-out process is based on the Kalman filter equations explained in A.2, which describe a discrete-time dynamic system in state-space model form. We consider a state for each agent, consisting of its position and velocity. Applying the Kalman equations, each timestep, k, produces first a new state prediction, $\hat{\boldsymbol{x}}_{\boldsymbol{k}}^{-}$, purely based on the dynamic physical process modeled in the filter on the previous timestep estimate; and finally, a new estimation, $\hat{\boldsymbol{x}}_{\boldsymbol{k}}$, after considering the updated observations.

The output of the Agent Dynamics component is a vector with current estimated states for all agents, $\hat{\boldsymbol{a}}_{\boldsymbol{k}}$. This vector contains estimated agent positions in ONTV space, as well as their velocity vectors.

6 Prediction engine

The prediction engine is the core stage of PlayNet and the main contribution of this work: a low-dimensional representation of the current video frame based on agent movement patterns, subsequently classified as one of the predefined seven game state classes. This low dimensional representation is what we call Kalman embeddings. It is computed by performing a dimensionality reduction operation on an array ($\hat{\boldsymbol{a}}_{\boldsymbol{k}}$) containing the position and velocity for each agent. As stated in Sect. 3, our hypothesis in this work is that Kalman embeddings capture defining features of the different types of plays in a handball game. Hence, we can use a simpler classifier instead of depending on expensive convolutional features or LSTM networks. The model flow diagram in Fig. 4 represents the two components that constitute this module: the proposed neural network, to reduce data dimensionality, and the classifier, that provides the final answer that PlayNet returns for the current frame.

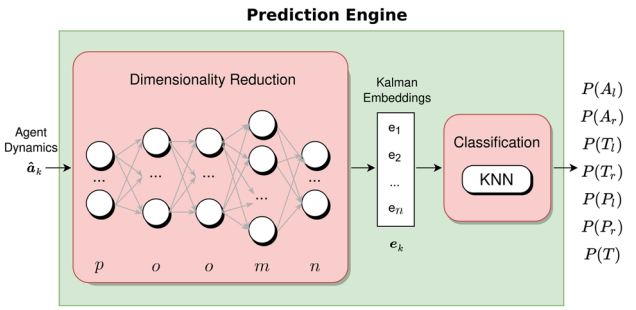


Fig. 4 The PlayNet prediction engine utilizes a neural network to perform dimensionality reduction of the normalized game state representation, $\hat{\boldsymbol{a}}_{\boldsymbol{k}}$. The resulting Kalman embeddings, $\boldsymbol{e}_{\boldsymbol{k}}$, represented as an n-D vector in the diagram, are then classified to obtain the game state probability distribution

6.1 Dimensionality reduction

A fully connected neural network with four layers (as shown in Fig. 4) was chosen to efficiently perform dimensionality reduction of unseen vectors in PlayNet.

The Kalman filter provides estimated positions and velocities for every agent, yielding p values for each frame (vector $\hat{\boldsymbol{a}}_{\boldsymbol{k}}$). We use variability analysis with PCA [42] for studying the variance that is explained by each set of components (for all sets of principal components, we add up their variances and divide them by the total variance). Then, we choose the output dimension n based on the explained variability distribution. For optimal parameters for handball classification and more details, please refer to Sect. 7.

Once the optimal output dimension is chosen, looking for effective ways to decrease the size of the data to be classified, we analyzed UMAP and other alternative approaches. Uniform Manifold Approximation and Projection (UMAP) [25] is a dimensionality reduction technique that can be used for visualization and clustering similarly to t-SNE [24]. Although UMAP is commonly used for unsupervised dimension reduction, it offers excellent flexibility and can be extended to perform other tasks, such as using categorical label information to conduct supervised dimensionality reduction. We found that UMAP, similarly to other methods, is not a suitable technique for our use case due to the increased processing overhead as the dataset gets bigger: the complete dataset must be used by UMAP to embed unseen data. Since PlayNet currently has more than 1 million input frames in its dataset, which will grow over time, we need a solution with better scalability with problem size to guarantee high-performance operation (refer to Table 1 for the comprehensive results obtained from each tested method).

We chose an architecture inspired by Auto Encoders (AEs), which have proven effective as a dimensionality reduction technique and could be valuable for better capturing latent structure in conjunction with UMAP. Our implementation uses a fully connected neural network with p units in its input layer (the input vector, $\hat{\boldsymbol{a}}_{\boldsymbol{k}}$, is p-dimensional), o units in the first and second hidden layers, and m in the third (being $m > o$), with the output layer containing n elements, which represent the desired embedding output size (for more implementation details, please refer to Sect. 7).

Our model was trained using supervised learning with a training set generated by UMAP, searching for a result as close as possible in terms of effectiveness but without the UMAP performance penalty. Mean squared error (MSE) was used to evaluate the correctness of the predicted embeddings.

The trained neural network computes the Kalman embeddings, $\boldsymbol{e}_{\boldsymbol{k}}$, used as input by the last element in our system for classifying handball plays.

Table 1 Accuracy, precision, recall, F-Score, model parameter count, and inference time for various combinations of embeddings and classifiers. To provide a clear overview for the reader, we have categorized the approaches based on their suitability for online or offline processing (we deem runtimes over 60 ms unsuitable for real-time camera control)

Perf.	Embed.	Classifier	Params.	Acc. (%)	Precision		Recall		F-Score		Time (ms)
					Weighted (%)	Macro (%)	Weighted (%)	Macro (%)	Weighted (%)	Macro (%)	
Online	**Ours**	**KNN** [9]	1.35M	91.7	91.9	73.4	91.7	74.3	**91.8**	**73.8**	**17.74**
		LightGBM [20]		91.6	91.7	72.9	91.6	72.1	91.6	72.5	
		RandomForest [15]		91.4	91.9	71.9	91.4	73.5	91.6	72.7	
		ExtraTress [13]		91.7	91.9	72.9	91.7	73.9	**91.8**	73.4	
		MLP [38]		91.6	**92.0**	72.1	91.6	**74.8**	**91.8**	73.4	
		Ensemble [43]		91.7	**92.0**	72.9	91.7	73.8	**91.8**	73.3	
	IVIS [37]	KNN [9]	0.05M	90.5	90.1	73.8	90.5	69.1	90.2	71.1	18.13
		LigthGBM [20]		90.7	90.4	78.1	90.7	66.8	90.0	70.9	
		RandomForest [15]		91.6	91.1	78.4	91.6	68.4	91.1	71.6	
		ExtraTrees [13]		89.3	89.3	**85.4**	89.3	58.4	88.1	61.9	
		MLP [38]		90.2	89.0	61.7	90.2	62.5	89.6	62.0	
		Ensemble [43]		91.2	90.7	76.5	91.2	69.0	90.8	71.7	
	None	MLP [38]	1.53M	89.0	88.7	62.0	89.0	56.2	88.3	57.6	18.60
Offline	UMAP [25]	KNN [9]		89.1	88.4	70.8	89.1	61.4	88.8	65.8	> 1000.00
		LightGBM [20]		89.2	88.6	71.5	89.3	61.3	89.0	66.0	
		RandomForest [15]		89.2	88.6	69.6	89.2	61.9	88.9	65.5	
		ExtraTrees [13]		89.1	88.6	69.9	89.2	61.5	88.9	65.4	
		MLP [38]		89.5	88.8	71.6	89.5	63.0	89.1	67.0	
		Ensemble [43]		89.2	88.6	69.8	89.2	61.5	88.9	65.4	
	openTSNE [31]	KNN [9]		85.9	84.6	63.1	85.9	56.2	84.8	58.1	> 1000.00
		LightGBM [20]		83.9	83.4	65.5	83.9	48.2	81.7	52.2	
		RandomForest [15]		85.9	85.0	71.0	85.9	53.1	84.5	55.8	
		ExtraTrees [13]		84.9	83.8	61.4	84.9	50.5	83.2	54.0	
		MLP [38]		84.6	83.2	59.9	84.6	50.8	83.1	53.9	
		Ensemble [43]		86.5	85.3	66.4	86.5	55.9	85.2	58.0	
	MDE [1]	KNN [9]		89.0	88.2	71.4	89.0	62.4	88.4	65.4	> 1000.00
		LigthGBM [20]		87.1	86.6	70.3	87.1	55.2	85.5	59.2	
		RandomForest [15]		88.7	87.9	73.6	88.7	58.5	87.6	60.2	
		ExtraTrees [13]		85.5	84.5	61.8	85.5	51.6	83.9	55.0	
		MLP [38]		87.8	86.2	60.2	87.8	58.6	86.9	59.2	
		Ensemble [43]		89.3	88.5	72.9	89.3	61.3	88.5	64.2	
	None	CNN [44]	1.44M	85.8	90.2	64.8	85.8	57.0	87.5	59.2	60.09
		LSTM [16]	1.39M	91.5	90.5	68.9	91.5	63.4	90.8	64.7	234.51
		CNN+LSTM [19]	1.47M	**92.0**	91.6	80.2	**92.0**	70.5	91.6	73.5	104.36

Bold highlights the best results obtained in terms of time and metrics. It also highlights the best classifier and embedding methods

6.2 Kalman embedding classification

The final component in the pipeline classifies the Kalman embeddings, which we hypothesize to be a low-dimensional movement pattern representing the ongoing events on the handball court. Consequently, this classification step effectively determines the game state for the current frame.

Our approach for this last step is treating it as a supervised classification problem: we have a dataset with several hours of handball videos annotated that we can use to train a machine learning model, i.e., a set of Kalman embeddings with their corresponding ground truth classes (more information about the dataset in Sect. 8). After testing multiple alternatives for this classification task, we concluded that a k-nearest neighbors classifier (KNN) [9] would be the most suitable option, considering the trade-off between accuracy and performance and the nature of our embeddings. Table 1 provides a comparison of accuracy and precision for the evaluated solutions.

The input to the classifier is the n-D vector of Kalman embeddings ($\boldsymbol{e_k}$ in Fig. 4), and the output is one of the possible game states: $\{A_l, A_r, T_l, T_r, P_l, P_r, T\}$. A vector is classified depending on the most common class among its nearest neighbors (in the training set). In turn, this means that to classify our plays; we need to find the K most similar neighbors (k-nearest neighbors). Since due to the supervised dimensionality reduction, our input space has data points that have been pulled apart and separated per class, we presume that a KNN classifier which relies on its nearest neighbors to classify unseen data can perform well. The experimental results presented in Table 1, comparing the KNN and other classification approaches, support our hypothesis and demonstrate the effectiveness of the KNN classifier.

The brute force approach for finding k-nearest neighbors involves examining every vector in the training set, resulting in a complexity of $O(ns)$, where n is the number of dimensions and s is the number of samples in the training set. To achieve real-time classification and accommodate a growing dataset, we employ a Ball-tree [23] acceleration structure, which significantly reduces both training and inference times. Compared to a brute force approach, the Ball-tree offers improved efficiency with a query time complexity of approximately $O(n \log s)$.

As a result of this stage, PlayNet classifies the current frame into one of the seven game states under consideration.

7 Implementation details

This section provides an in-depth look into the low-level implementation details of Playnet.

Pre-processing. The input video streams from the cameras, two streams in our experimental setup (as shown in Figs. 2 and 3), are pre-processed entirely on the GPU. First, the images from each *h.264* stream are decoded using NVDEC, leveraging hardware decoding. Next, the decoded images are processed using NPP to perform batch pixel format conversions (NV12 to RGB), cropping (area of interest), re-sizing (4K crop to 832×832) and color remapping (from the range [0, 255] to [0, 1]). All these operations are conducted on the GPU without requiring data transfers between the CPU and GPU.

Agent detection. For object detection in the pre-processed images, our experiments with YOLOv4 revealed that utilizing batches of $4 \times 832 \times 832$ optimizes GPU resource utilization and minimizes memory transfers. The area of interest in each input video stream is cropped, divided in half, and resized to generate a set of four images. The chosen resolution and batch size consider VRAM limitations and aim to maximize compute resource usage.

For optimal performance, we employed the tkdnn [41] framework, which leverages TensorRT. At the time of our experiments, this combination provided the fastest solution. After optimizing and quantizing YOLOv4 with 1000 images for calibration and using 8-bit integer precision for inference (similar to [8]), we achieved a player detection precision of 94 mAP for players and 85 mAP for balls. mAP stands for *mean Average Precision*, a commonly used evaluation metric in computer vision and object detection. The use of 8-bit integer precision at inference time resulted in a speedup compared to floating-point precision using either 32 or 16 bits.

Agent tracking & dynamics. In terms of the tracking algorithm, the parameters that produced the best results for our setup were $J_{min} = 0.5$, $\delta_{max} = 0.02$ and $f_{max} = 5$ (these parameters are described in Sect. 5.2). Regarding the Kalman filter component, $\sigma_{px} = 10$, $n_{acc} = 5$ and $\mu_{kin} = 0.99$ (all detailed in A.2) proved to be the most effective choices, after multiple experiments with our test setup.

Dimensionality reduction. Moving on to the prediction engine, a frame is represented by a 68-D vector which contains information concerning the 17 agents on the court ($p = 68$). If there are more detected agents than this number, we favor agents that are moving faster and are far away from the court limits when selecting agents for higher robustness. If less agents than needed are present, the vector is padded with zeros.

After conducting a variability analysis with PCA, we concluded that 30 components explain 92% of the variance. The explained variance remained almost constant using more features, so that was the chosen output dimension ($n = 30$). Consequently, we have used UMAP with default parameters

$n\_neighbors = 15$ and $metric =$ 'euclidean' while changing $n\_components = 30$.

Regarding the neural network hidden layer size in Fig. 4, we found that an $o = 500$ and an $m = 2000$ worked well for this dataset. In all training tests, the Adam optimizer was used with an initial learning rate $\eta = 0.001$, $batch\_size = 5000$ and default parameters $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$ for 600 epochs. The modified parameters as well as the architecture for the neural net were obtained with a manually fine-tuned hyperparameter search.

Classification. Finally, delving into the parameters used for the chosen classifier, the KNN, we conducted another manual fine-tuning process following a hyperparameter search, which led us to the following parameter values: $n\_neighbors = 5$, $weights =$ 'distance', $algorithm =$ 'ball_tree', $leaf\_size = 30$, $p = 2$, and $metric =$ 'minkoswki'. These parameters were selected based on their performance, resulting in the best outcome.

8 Results

To address the lack of public datasets for the proposed problem, we created a new dataset comprising approximately 11 h of footage from five handball games held in two different arenas. The dataset consists of one million frames captured using a static two-camera setup (see Fig. 3), with video frames synchronized between the two cameras to ensure coherent results. Each frame was meticulously labeled with the seven classes that define the game state. Figure 1 provides illustrative examples for each of the seven classes and the corresponding number of frames in the dataset.

In order to test our method, the dataset has been divided into two parts: a training split, which comprises 70% of the dataset, and a testing split, which constitutes the remaining 30%. For the training split, we utilized three handball matches in one arena, while the testing split involved two unseen matches in a different arena. This allocation aligns precisely with the chosen percentages. Testing unseen matches and arenas provides a robust evaluation of both the effectiveness of our camera setup and the validity of our agent dynamics abstraction scheme.

All the experiments were performed using an i7-8700K CPU, RTX 2070 GPU, 32 GB DDR4 RAM and a 256GB SATA SSD.

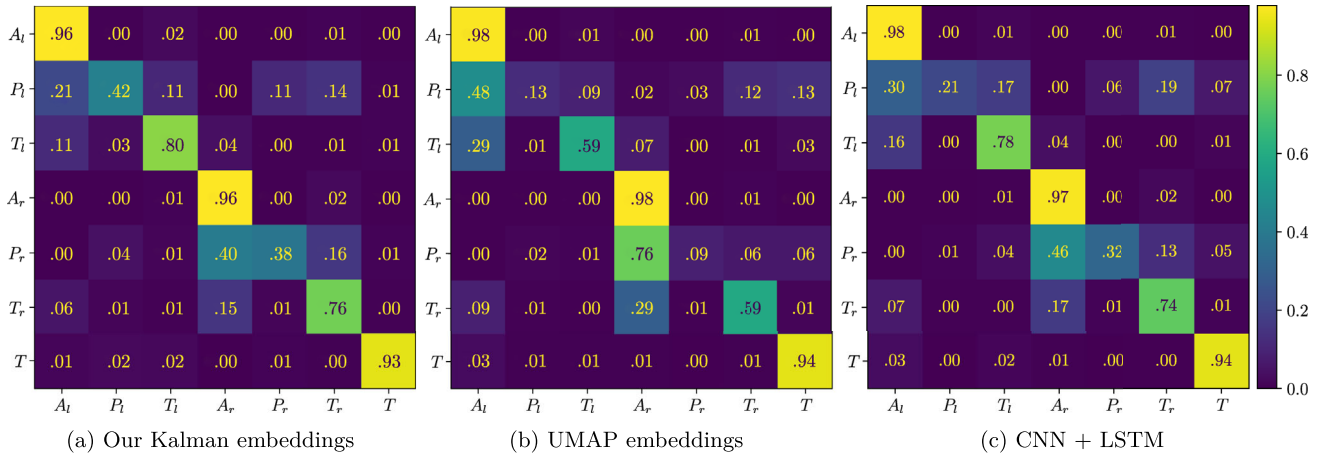
8.1 Accuracy, precision, recall and F-Score

Four well-known metrics are used to assess the effectiveness of PlayNet: accuracy, which represents the overall correctness of predictions; precision, which quantifies the exactness of PlayNet when predicting a specific game state; recall, denoting the percentage of total occurrences of a class that PlayNet can detect; and F-Score, which is the harmonic mean of precision and recall. In addition, we provide both simple (macro) and weighted averages for precision, recall, and F-Score to account for class imbalance within a handball game, as depicted in Fig. 1.

The results achieved for the selected metrics on the testing split are presented in Table 1. The table includes outcomes from various classifiers, including the KNN model that was ultimately selected for PlayNet. Other models tested include LightGBM [20], a gradient boosting library that leverages tree-based learning algorithms. We chose LightGBM over XGBoost [7] due to its superior compute and memory efficiency. Additionally, classical tree decision methods such as Random Forests [15] and Extra Trees [13] were evaluated. Furthermore, a fully connected neural network architecture was also tested to explore a different classification approach. Finally an ensemble of all the aforementioned classifiers was also assessed using stacking [43], which trains a model that combines all of their predictions in an optimal manner.

In addition to showcasing the classification results obtained using our embeddings, we also present the outcomes attained by other state-of-the-art methods for embedding data into a lower-dimensional space previously discussed in Sect. 2, such as UMAP [25], IVIS [37], t-SNE [31], and MDE [1]. To explore all possible avenues, we also evaluated models that do not rely on Kalman embeddings but instead leverage the available temporal information by processing multiple frames using a sliding window approach. For this purpose, we selected a MLP [38], a 1D CNN [44], a LSTM [16], and a hybrid architecture that uses both a 1D CNN and a LSTM [19]. These models are specifically designed for time-series classification, but they offer an alternative approach to tackle our problem, which can also be framed within the context of time-series analysis. We used default parameters for all tested methods where developer recommendations tailored to our dataset size were unavailable.

The table shows how the neural network embedding methods ('Ours' and 'IVIS') obtained an average accuracy of around $\sim 92\%$ versus $\sim 89\%$ with the closest offline embedding methods ('UMAP' and 'MDE'). These results, coupled with the performance advantage when using our neural network embeddings, illustrate the benefits and effectiveness of our method. Furthermore, delving deeper into the results, we can see that UMAP, t-SNE and MDE embeddings underperform our embeddings and have particular trouble with underrepresented classes in the training dataset, we can see this in the macro F-Score ('Ours' $\sim 74\%$ versus 'UMAP' $\sim 67\%$), probably due to overfitting when using the offline methods which process again all of the dataset for every new data point, causing them to adapt their latent representation to the new data which can skew the final result. Furthermore, when considering neural network embeddings, our method



(a) Our Kalman embeddings (b) UMAP embeddings (c) CNN + LSTM

Fig. 5 Confusion matrices normalized over true labels (rows) of the top-performing representatives from online and offline dimensionality reduction (evaluated using the KNN classifier) and no dimensionality reduction. The seven game states considered in PlayNet are noted along the axes: true class on the y-axis and predicted class on the x-axis

exhibits a slight advantage over IVIS in terms of F-Score ('Ours' $\sim$ 74% compared to 'IVIS' $\sim$ 72%), further reinforcing our method as the preferred choice.

Lastly, we have not found any advantages regarding methods that do not require Kalman embeddings. In terms of accuracy, the best of them slightly outperforms our method ('Ours' 91.7% versus 'CNN+LSTM' 92%) but has slightly more problems with underrepresented classes; we can see this in the macro F-Score ('Ours' 73.8% versus 'CNN+LSTM' 73.5%), please refer to Fig. 5 for a detailed analysis of these results broken down by class. In summary, the obtained F-Score, coupled with the performance penalty due to the model overhead, presents our method as the right choice. Not only did the best of these models not perform in real-time, but they also failed to outperform our approach in terms of the macro F-Score. Considering both factors, our chosen approach demonstrated superior performance compared to all of the tested alternatives.

Regarding the chosen evaluation metrics, we considered the F-Score especially relevant for specific classes, such as penalties. For instance, if the main difference between a penalty and an attack in a handball match, from a production point of view, is that a penalty can activate a closeup or an extra camera shot covering only the penalty area, missing some penalty detections can be acceptable (i.e., a relatively low recall score for penalties) as the attack view will cover a superset of the penalty area. In contrast, false penalty detections (i.e., a low precision score for penalties) can lead to entirely wrong camera shots. They are two important classes but far less frequent than others during a match, the macro-average score deserves special attention as it exposes a more representative picture of the system performance for these scarce classes. Since striking a balance between precision and recall is essential, we consider the macro F-Score as the most relevant metric for our specific use case (the precision metric loses its significance when the model fails to capture the majority of penalties, as illustrated in Fig. 5). Both the macro F-Score metric and the execution time were determining factors for the selection of the final model.

8.2 Runtime performance

The average processing time of one thousand runs using our method is shown in Table 2. As observed in the breakdown of timings, the primary bottleneck in the PlayNet pipeline is the agent detection stage. Indeed, one of the main reasons for choosing a lightweight version of YOLOv4 in PlayNet is to optimize inference times as much as possible (for more details please refer to Sect. 7).

Decoding is performed in parallel for each frame in a batch to achieve maximum performance. These stages are computed in the GPU (decoding, pixel format conversion, cropping, resizing, and detection) without data transfers to the CPU, which to our knowledge, is the fastest way to perform these operations (avoiding latency constraints). On the other hand, the rest of the steps (Kalman filtering, ONTV transforms, embeddings, and KNN classifier prediction) use CPU processing since they are not well suited for the GPU. It is worth noting that due to having only a single element for inference at each timestep, we are unable to fully maximize the utilization of CPUs or GPUs. In our testing, we did not surpass 1 GFLOPs with any of the methods, even with the more complex models such as CNNs or LSTMs. This indicates that our performance is limited by latency rather than computation, which particularly affects the GPU. However it is important to mention that this limitation is specific to our scenario, as in offline processing scenarios where batch inference with thousands of elements can be leveraged, GPU processing is likely to outperform CPU processing.

Table 2 Pipeline execution time measurements

Stage	Time (ms)
Input Pre-processing	**7.50**
Ingestion & Decoding [GPU]	4.94
Video pre-processing [GPU]	2.56
Normalized Game State Repr.	**27.42**
Agent detection [GPU]	26.69
Agent tracking [CPU]	0.21
ONTV space [CPU]	0.45
Agent dynamics [CPU]	0.07
Prediction Engine	**17.74**
Dimensionality reduction [CPU]	16.85
Kalman embed. classification [CPU]	0.89
Total	**52.67**

Bold represents the different stages of the pipeline and their cumulative execution time

Finally, upon closer examination of the runtime performance achieved by our prediction engine, it is evident that our method surpasses all other approaches. As depicted in Table 1, the fastest methods, including Ours, IVIS, and the fully connected neural network, achieve a run-time of $\sim$ 18 ms. Notably, our method achieves the fastest runtime with 17.74 ms. On the other hand, the more computationally intensive models require a minimum of $\sim$ 60 ms for execution, rendering them unsuitable for our purpose. The slowest methods such as t-SNE, UMAP, and MDE, exceed a second of execution time.

8.3 System limitations and future work

Upon examination of the confusion matrices in Fig. 5, it is apparent that the classification of penalties (both left and right) presents a notable challenge. This issue persists across all the models evaluated in Table 1, as they consistently confuse penalties with transitions and attacks. Notably, this problem is more pronounced when utilizing UMAP embeddings, as indicated by the corresponding KNN classifier confusion matrices. It demonstrates that penalties are only correctly identified approximately $\sim$ 10% of the time when using UMAP embeddings, compared to around $\sim$ 40% when employing Kalman embeddings. Moreover, for models that do not incorporate dimensionality reduction techniques, the situation is a little better, with penalties being correctly identified around $\sim$ 26% of the time, but still not up to par with our method.

Although the results for penalty classification are not ideal, our method demonstrates a better ability to correctly identify penalties and when it does not, the correct attack side, resulting in less confusion between the two sides of the court. This mitigates the impact of the errors associated with penalty classification. Regarding handball match production, these errors imply that a more general view will occasionally be chosen over a zoom shot when a penalty occurs. While this is not ideal, it can still be considered an acceptable worst-case scenario. However, the same cannot be said for the CNN+LSTM or UMAP models. They tend to exhibit a higher probability of incorrectly classifying penalties, often even with plays on the opposite side of the court, which would lead to suboptimal camera shots.

Currently, PlayNet requires a manual calibration process to ensure proper operation. During this calibration, operators manually select the area of interest in each camera, aligning it with the boundaries of the court. Incorporating this step has proven to be straightforward and highly beneficial for agent tracking, as it effectively mitigates issues such as the erroneous detection of external individuals as players within the game. While it may not eliminate sporadic invasions onto the field, this calibration step minimizes their impact. Additionally, PlayNet leverages these defined regions of interest to focus on the correct agents to track and avoid double detections.

On the other hand, determining tracking parameters such as J_{min}, δ_{max} and f_{max} is more challenging and often requires a trial-and-error approach. They have to be as rigid as possible but allowing for agent matches to persist enough frames to deal with occlusion and flickering in the detector. Nonetheless, based on our extensive testing, the used values have demonstrated effectiveness across various arenas. Similarly, the Kalman filter component parameters, namely σ_{px}, n_{acc}, and μ_{kin}, are also determined through trial and error during the calibration process. While they seem to work well across different arenas, finding their optimal values may pose a challenge. They also need to be conservative enough to allow the Kalman filter to estimate more robust positions and velocities with the chosen dynamics model, but at the same time rely sufficiently on the measured data. These parameters try to account for errors in the detected positions, the balance between the dynamic model and observations and the fact that players need to exert an effort to keep on moving. Operators must conduct thorough testing to obtain the best parameter values if the default settings do not yield satisfactory results.

In terms of future developments, incorporating prior knowledge of the game could significantly ease the learning process of the neural network. For example, actively tracking pertinent zones within the game court where agents are likely to be positioned in certain situations and integrating this information into the game state representation. Additionally, exploring the use of data augmentation techniques could prove beneficial in addressing class imbalance issues. To further streamline operations, another potential improvement would be the automation of the calibration process, thus eliminating one of the remaining manual tasks and the most error prone part of the system. Lastly, accounting for lens

distortion during calibration, which is presently overlooked to simplify the process, could improve accuracy when estimating agent positions in ONTV space.

9 Conclusions

We have presented PlayNet, a novel real-time handball play classification approach capable of detecting game states on commodity hardware and customized camera layouts. Furthermore, the proposed system operates on raw video footage without relying on specialized tracking hardware like IMUs, filling the real-time game state classification gap for automatic low-cost live broadcasting systems in sports. Our approach represents the main agents in the playing field within a single normalized coordinate space: the ONTV space. This space decouples the inference engine from the input video streaming configuration, allowing PlayNet to work on a unified dataset space.

Our key contribution was transforming those high-dimensional game state representations in ONTV space into lower-dimensional instances tractable on commodity hardware for real-time play classification purposes. To accomplish this, we proposed Kalman embeddings, a lower-dimensional representation of the game state obtained with a neural network trained to reduce dimensionality. Using the Kalman embeddings presented a twofold advantage, first helping to reduce inference time and, secondly, reducing training time and training data size requirements for the different classifiers. In this work, we suggested using a KNN classifier and compared its performance with other commonly employed models.

Our experiments on production-ready environments supported our original idea: using movement patterns exclusively to classify handball plays. For instance, using a standard KNN classifier, we obtained over 91.7% accuracy and an F-Score of around 73.8% (about 91.8% by using a weighted average considering the existing class imbalance). Furthermore, the proposed system demonstrated real-time inference capabilities (below $\sim$ 55 ms on commodity hardware). Additionally, we released an open dataset for handball play classification, which we used for training and testing PlayNet.

Acknowledgements This work has been developed under the European Innovation Council Pilot No 954040. This work was supported also by ED431F 2021/11 and ED431G 2019/01 funded by Xunta de Galicia. Emilio J. Padrón's work was also partially supported through the research projects PID2019-104184RB-I00 funded by MCIN/AEI/10.13039/501100011033, and ED431C 2021/30. Jose A. Iglesias-Guitian also acknowledges the UDC-Inditex InTalent programme and the Spanish Ministry of Science and Innovation (AEI/RYC2018-025385-I).

Funding Open Access funding provided thanks to the CRUE-CSIC agreement with Springer Nature.

Data Availability The datasets generated during and/or analysed during the current study are available in the Zenodo repository, https://doi.org/10.5281/zenodo.7180366.

Declarations

Conflict of interest The author certifies that there is no conflict of interest with any individual/organization for the present work.

A Appendix

A.1 ONTV space formulation

A four-point perspective transformation is defined to convert the coordinates of a camera's four corner points $(c_{tl}, c_{tr}, c_{bl}, c_{br})$ into our common ONTV space. As a result, we can determine the following relationship among the homography matrix for each camera, $\mathbf{H}_c$, its corresponding scaling factor, u, and the coordinates of both spaces:

$$\begin{bmatrix} u\zeta_x \\ u\zeta_y \\ u \end{bmatrix} = \mathbf{H}_c \cdot \begin{bmatrix} c_x \\ c_y \\ 1 \end{bmatrix}, \tag{3}$$

where (c_x, c_y) represents the coordinates in the camera image space of one of the four corner points defining the region of interest, and (ζ_x, ζ_y) are the coordinates of the same corner point represented in our target ONTV space. Thus, to obtain $\mathbf{H}_c$ for a given camera source we just need to solve the system of linear equations defined by four equations like Eq. 3, one for each corner point:

$$\begin{bmatrix} u\zeta_x \\ u\zeta_y \\ u \end{bmatrix} = \mathbf{H}_c \cdot \begin{bmatrix} c_x \\ c_y \\ 1 \end{bmatrix} \Rightarrow \begin{bmatrix} u\zeta_x \\ u\zeta_y \\ u \end{bmatrix} = \begin{bmatrix} h_{11}\ h_{12}\ h_{13} \\ h_{21}\ h_{22}\ h_{23} \\ h_{31}\ h_{32}\ h_{33} \end{bmatrix} \cdot \begin{bmatrix} c_x \\ c_y \\ 1 \end{bmatrix} \tag{4}$$

Next, we just need to transform one of the four corner points using $\mathbf{H}_c$ and leverage the calibration data to dehomogenize the result and obtain the scaling factor u. By solving

the corresponding system of linear equations for each camera source present in the system, and following the previously described dehomogenization process, we can obtain both $\mathbf{H}_c$ and u for all the cameras in our system, allowing us to transform any given point in the field from the image space of one camera into ONTV space.

After computing $\mathbf{H}_c$ and u for each camera source, given an agent located at p_x, p_y coordinates in the corresponding image-space of its camera source, the final coordinates o_x, o_y in ONTV space are computed as follows:

$$\begin{bmatrix} o_x \\ o_y \end{bmatrix} = \frac{1}{u} \cdot \begin{bmatrix} \frac{h_{11}p_x + h_{12}p_y + h_{13}}{h_{31}p_x + h_{32}p_y + h_{33}} \\ \frac{h_{21}p_x + h_{22}p_y + h_{23}}{h_{31}p_x + h_{32}p_y + h_{33}} \end{bmatrix} \tag{5}$$

A.2 Kalman filter formulation

The following two equations, that describe a discrete-time dynamic system in state-space model form, are the base for our Kalman filter: the process equation (Eq. 6), that takes the state vector $\boldsymbol{x}$ from time $k-1$ to time k, and the measurement equation (Eq. 7), that associates a vector of observations $\boldsymbol{z}_k$ with a specific state $\boldsymbol{x}_k$:

$$\boldsymbol{x}_k = \mathbf{F_k}\boldsymbol{x}_{k-1} + \boldsymbol{w}_k \tag{6}$$

$$\boldsymbol{z}_k = \mathbf{H}\boldsymbol{x}_k + \boldsymbol{v}_k \tag{7}$$

$\mathbf{F_k}$ is the state transition matrix used to update the system state from timestep $k-1$ to k, $\boldsymbol{w}_k$ is the process noise, $\mathbf{H}$ is the measurement matrix, whose purpose is to convert system state into outputs, and $\boldsymbol{v}_k$ is the measurement noise. Both $\boldsymbol{w}_k$ and $\boldsymbol{v}_k$ are zero-mean, uncorrelated gaussian noise, and have known covariance matrices $\boldsymbol{Q}$ and $\boldsymbol{R}$, respectively.

From those two equations, Kalman obtains the five equations that are applied each iteration. First, in the prediction stage, the state estimate and the error covariance are propagated from the previous to the current state by the transition equation (Eq. 8) and the predictor covariance equation (Eq. 9):

$$\hat{\boldsymbol{x}}_k^- = \mathbf{F_k}\hat{\boldsymbol{x}}_{k-1} + \boldsymbol{w}_k \tag{8}$$

$$\mathbf{P}_k^- = \mathbf{F_k}\mathbf{P}_{k-1}{\mathbf{F_k}}^\top + \mathbf{Q} \tag{9}$$

The state transition matrix, $\mathbf{F_k}$, is used to predict the system state in the current frame (timestep k) by updating the state in the previously processed frame (timestep $k-1$). In our simplified physics model, this matrix depends on the time between both frames, Δt, and the kinetic friction, μ_{kin}, obtained through experimentation:

$$\mathbf{F_k} = \begin{bmatrix} 1 & 0 & \Delta t(1 - \mu_{kin}\Delta t/2) & 0 \\ 0 & 1 & 0 & \Delta t(1 - \mu_{kin}\Delta t/2) \\ 0 & 0 & 1 - \mu_{kin}\Delta t & 0 \\ 0 & 0 & 0 & 1 - \mu_{kin}\Delta t \end{bmatrix} \tag{10}$$

Table 3 Main variables considered in our Kalman filter model

Notation	Description
$\boldsymbol{x}_k$	*State vector*: agent state for timestep k, $\boldsymbol{x}_k = [x, y, \dot{x}, \dot{y}]^T$, being x, y the position and $\dot{x}$, $\dot{y}$ the velocity
$\hat{\boldsymbol{x}}_k$	*Estimated state vector*: Kalman estimated agent state for timestep k
$\hat{\boldsymbol{x}}_k^-$	*Predicted state vector*: agent state predicted by the physical model for timestep k (*a priori state estimation*, without considering observations for timestep k)
$\boldsymbol{z}_k$	*Observation vector*: agent position from the object detector in timestep k, plus velocity computed considering the previously estimated position, $\boldsymbol{z}_k = \left[o_{x_k}\ o_{y_k}\ \frac{o_{x_k} - o_{x_{k-1}}}{\Delta t}\ \frac{o_{y_k} - o_{y_{k-1}}}{\Delta t}\right]^T$
$\mathbf{F}_k$	*Transition matrix*: predicts the current frame state from the previous one
$\boldsymbol{w}_k$	*Process noise*: expected error in the prediction process
$\boldsymbol{v}_k$	*Measurement noise*: expected error in the measurement
$\mathbf{H}$	*Measurement matrix*: converts between measurement and state
$\mathbf{P}_k$	*State covariance matrix*: state variance-covariance matrix for frame k
$\mathbf{P}_k^-$	*Predicted state covariance matrix*: predicted state variance-covariance matrix for frame k
$\mathbf{Q}$	*Process noise covariance matrix*: variance-covariance matrix that models process error
$\mathbf{R}$	*Measurement noise covariance matrix*: variance-covariance matrix that models measurement error
$\mathbf{K}_k$	*Kalman gain*: weight distribution between predicted state and measured observations
Δt	*Frame time*: time interval between previous and current frame
σ_{px}	*Pixel Standard Deviation*: estimated error of a bounding box position observation
n_{acc}	*Acceleration Noise*: balance of trust in the dynamical model and observations
μ_{kin}	*Kinetic Friction*: friction proportional to velocity

Note that in our case the transition matrix may be different every step because Δt is not a fixed time interval, but dependent on the video streaming real-time response (Table 3).

Regarding the covariance matrix, $\mathbf{Q}$, we considered an acceleration noise, n_{acc}, to model the process noise and uncertainty:

$$\boldsymbol{n} = \begin{bmatrix} \frac{\Delta t^2 n_{acc}}{2} & \frac{\Delta t^2 n_{acc}}{2} & \Delta t n_{acc} & \Delta t n_{acc} \end{bmatrix} \tag{11}$$

$$\mathbf{Q} = \boldsymbol{n}^\intercal \boldsymbol{n} \tag{12}$$

In the state update stage, the Kalman filter corrects the prediction for the current frame, $\hat{\boldsymbol{x}}_k^-$, with the observed values, z_k, applying the state update equation (Eq. 13) to obtain the estimated state for the current frame, $\hat{\boldsymbol{x}}_k$, and updating the process covariance matrix (Eq. 14):

$$\hat{\boldsymbol{x}}_k = \hat{\boldsymbol{x}}_k^- + \mathbf{K}_k(z_k - \mathbf{H}\hat{\boldsymbol{x}}_k^-) \tag{13}$$

$$\mathbf{P}_k = (\mathbb{I} - \mathbf{K}_k\mathbf{H})\mathbf{P}_k^- \tag{14}$$

$\mathbf{K}_k$ is the Kalman gain, basically the importance we give to observations, z_k, i.e. $(1-\mathbf{K}_k)$ would be the weight we give to the previously predicted state, $\hat{\boldsymbol{x}}_k^-$. Of course, the Kalman gain is updated before applying Eq. 13 and Eq. 14:

$$\mathbf{K}_k = \frac{\mathbf{P}_k^- \mathbf{H}^\top}{\mathbf{H}\mathbf{P}_k^- \mathbf{H}^\top + \mathbf{R}} \tag{15}$$

Since we use agent position and velocity in both the system state and observations, our measurement matrix, $\mathbf{H}$, is a 4×4 identity matrix: $\mathbf{H} = \mathbb{I}$.

Finally, the covariance matrix we use to model the measurement error is this:

$$\mathbf{R} = \begin{bmatrix} 2\sigma_{px} & 0 & 0 & 0 \\ 0 & \sigma_{px} & 0 & 0 \\ 0 & 0 & 2\sigma_{px} & 0 \\ 0 & 0 & 0 & \sigma_{px} \end{bmatrix} \tag{16}$$

where σ_{px} is a parameter that accounts for the expected standard deviation in the measurement of the agent position by the object detector.

References

1. Agrawal, A., Ali, A., Boyd, S.: Minimum-distortion embedding. Found. Trends Mach. Learn. **14**(3), 211–378 (2021). https://doi.org/10.1561/2200000090
2. Ali, M., Jones, M.W., Xie, X., Williams, M.: Timecluster: dimension reduction applied to temporal data for visual analytics. Vis. Comput. **35**(6–8), 1013–1026 (2019). https://doi.org/10.1007/s00371-019-01673-y
3. Biermann, H., Theiner, J., Bassek, M., Raabe, D., Memmert, D., Ewerth, R.: A unified taxonomy and multimodal dataset for events in invasion games. In: Proceedings of the 4th International Workshop on Multimedia Content Analysis in Sports, pp. 1–10 (2021). https://doi.org/10.48550/arXiv.2108.11149
4. Bochkovskiy, A., Wang, C.Y., Liao, H.Y.M.: YOLOv4: Optimal speed and accuracy of object detection (2020). https://doi.org/10.48550/arXiv.2004.10934
5. Carr, P., Mistry, M., Matthews, I.: Hybrid robotic/virtual pan-tilt-zom cameras for autonomous event recording. In: Proceedings of the 21st ACM international conference on Multimedia, pp. 193–202 (2013). https://doi.org/10.1145/2502081.2502086
6. Carrillo, H., Quiroga, J., Zapata, L., Maldonado, E.: Automatic football video production system with edge processing. Mach. Vis. Appl. **33**(2), 32 (2022). https://doi.org/10.1007/s00138-022-01283-0
7. Chen, T., Guestrin, C.: Xgboost: A scalable tree boosting system 785–794 (2016). https://doi.org/10.1145/2939672.2939785
8. Choukroun, Y., Kravchik, E., Yang, F., Kisilev, P.: Low-bit quantization of neural networks for efficient inference. In: Proceedings of the 2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW), pp. 3009–3018. IEEE (2019). https://doi.org/10.1109/ICCVW.2019.00363
9. Cover, T., Hart, P.: Nearest neighbor pattern classification. IEEE Trans. Inf. Theory **13**(1), 21–27 (1967). https://doi.org/10.1109/TIT.1967.1053964
10. Criminisi, A., Reid, I., Zisserman, A.: A plane measuring device. Image Vis. Comput. **17**(8), 625–634 (1999). https://doi.org/10.1016/S0262-8856(98)00183-8
11. Cuevas, C., Quilon, D., García, N.: Techniques and applications for soccer video analysis: a survey. Multimed. Tools Appl. **79**(39), 29685–29721 (2020). https://doi.org/10.1007/s11042-020-09409-0
12. Deliege, A., Cioppa, A., Giancola, S., Seikavandi, M.J., Dueholm, J.V., Nasrollahi, K., Ghanem, B., Moeslund, T.B., Van Droogenbroeck, M.: Soccernet-v2: A dataset and benchmarks for holistic understanding of broadcast soccer videos. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 4508–4519 (2021). https://doi.org/10.48550/arXiv.2011.13367
13. Geurts, P., Ernst, D., Wehenkel, L.: Extremely randomized trees. Mach. Learn. **63**, 3–42 (2006). https://doi.org/10.1007/s10994-006-6226-1
14. Guntuboina, C., Porwal, A., Jain, P., Shingrakhia, H.: Deep learning based automated sports video summarization using YOLO. Electron. Lett. Comput. Vis. Image Anal. **20**(1), 99–116 (2021). https://doi.org/10.5565/rev/elcvia.1286
15. Ho, T.K.: Random decision forests. In: Proceedings of 3rd international conference on document analysis and recognition, vol. 1, pp. 278–282. IEEE (1995). https://doi.org/10.1109/ICDAR.1995.598994
16. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. **9**(8), 1735–1780 (1997). https://doi.org/10.1162/neco.1997.9.8.1735
17. Ivasic-Kos, M., Host, K., Pobar, M.: Application of deep learning methods for detection and tracking of players. In: P.L. Mazzeo, P. Spagnolo (eds.) Deep Learning Applications. IntechOpen (2021). https://doi.org/10.5772/intechopen.96308
18. Kalman, R.E., Bucy, R.S.: New results in linear filtering and prediction theory. J. Basic Eng. **83**(1), 95–108 (1961). https://doi.org/10.1115/1.3658902
19. Karim, F., Majumdar, S., Darabi, H., Chen, S.: Lstm fully convolutional networks for time series classification. IEEE Access **6**, 1662–1669 (2017). https://doi.org/10.1109/ACCESS.2017.2779939
20. Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., Liu, T.Y.: Lightgbm: a highly efficient gradient boosting decision tree. Adv. Neural Inf. Process. Syst. **30** (2017)
21. Leng, L., Li, M., Kim, C., Bi, X.: Dual-source discrimination power analysis for multi-instance contactless palmprint recognition. Multimed. Tools Appl. **76**, 333–354 (2017). https://doi.org/10.1007/s11042-015-3058-7

22. Leng, L., Zhang, J., Chen, G., Khan, M.K., Alghathbar, K.: Two-directional two-dimensional random projection and its variations for face and palmprint recognition. In: Computational Science and Its Applications - ICCSA 2011, Lecture Notes in Computer Science, vol. 6786, pp. 458–470. Springer (2011). https://doi.org/10.1007/978-3-642-21934-4_37
23. Liu, T., Moore, A.W., Gray, A.: New algorithms for efficient high-dimensional nonparametric classification. J. Mach. Learn. Res. **7**, 1135–1158 (2006)
24. Maaten, L.v.d., Hinton, G.: Visualizing data using t-SNE. J. Mach. Learn. Res. **9**, 2579–2605 (2008). http://jmlr.org/papers/v9/vandermaaten08a.html
25. McInnes, L., Healy, J., Saul, N., Großberger, L.: UMAP: Uniform manifold approximation and projection. J. Open Source Softw. **3**(29), 861 (2018). https://doi.org/10.21105/joss.00861
26. Mendi, E., Clemente, H.B., Bayrak, C.: Sports video summarization based on motion analysis. Comput. Electr. Eng. **39**(3), 790–796 (2013). https://doi.org/10.1016/j.compeleceng.2012.11.020
27. Morra, L., Manigrasso, F., Canto, G., Gianfrate, C., Guarino, E., Lamberti, F.: Slicing and dicing soccer: automatic detection of complex events from spatio-temporal data. In: Image Analysis and Recognition - ICIAR 2020, Lecture Notes in Computer Science, vol. 12131, pp. 107–121. Springer (2020). https://doi.org/10.1007/978-3-030-50347-5_11
28. Müller, O., Caron, M., Döring, M., Heuwinkel, T., Baumeister, J.: PIVOT: a parsimonious end-to-end learning framework for valuing player actions in handball using tracking data. In: Proceedings of the International Workshop on Machine Learning and Data Mining for Sports Analytics (MLSA 2021), Communications in Computer and Information Science, vol. 1571, pp. 116–128. Springer (2022). https://doi.org/10.1007/978-3-031-02044-5_10
29. Norgård Rongved, O.A., Hicks, S.A., Thambawita, V., Stensland, H.K., Zouganeli, E., Johansen, D., Riegler, M.A., Halvorsen, P.: Real-time detection of events in soccer videos using 3D convolutional neural networks. In: Proceedings of the 2020 IEEE International Symposium on Multimedia (ISM 2020), pp. 135–144. IEEE (2020). https://doi.org/10.1109/ISM.2020.00030
30. Oytun, M., Tinazci, C., Sekeroglu, B., Acikada, C., Yavuz, H.U.: Performance prediction and evaluation in female handball players using machine learning models. IEEE Access **8**, 116321–116335 (2020). https://doi.org/10.1109/ACCESS.2020.3004182
31. Poličar, P.G., Stražar, M., Zupan, B.: openTSNE: a modular Python library for t-SNE dimensionality reduction and embedding. bioRxiv preprint (2019). https://doi.org/10.1101/731877
32. Quiroga, J., Carrillo, H., Maldonado, E., Ruiz, J., Zapata, L.M.: As seen on TV: automatic basketball video production using gaussian-based actionness and game states recognition. In: Proceedings of the 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), pp. 3911–3920. IEEE (2020). https://doi.org/10.1109/CVPRW50498.2020.00455
33. Rezatofighi, H., Tsoi, N., Gwak, J., Sadeghian, A., Reid, I., Savarese, S.: Generalized intersection over union: A metric and a loss for bounding box regression. In: Proceedings of the 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR 2019), pp. 658–666. IEEE (2019). https://doi.org/10.1109/CVPR.2019.00075
34. Schlipsing, M., Salmen, J., Tschentscher, M., Igel, C.: Adaptive pattern recognition in real-time video-based soccer analysis. J. Real Time Image Proc. **13**(2), 345–361 (2017). https://doi.org/10.1007/s11554-014-0406-1
35. Shih, H.C.: A survey of content-aware video analysis for sports. IEEE Trans. Circuits Syst. Video Technol. **28**(5), 1212–1231 (2017). https://doi.org/10.1109/TCSVT.2017.2655624
36. Shingrakhia, H., Patel, H.: Sgrnn-am and HRF-DBN: a hybrid machine learning model for cricket video summarization. Vis. Comput. **38**(7), 2285–2301 (2022). https://doi.org/10.1007/s00371-021-02111-8
37. Szubert, B., Cole, J.E., Monaco, C., Drozdov, I.: Structure-preserving visualisation of high dimensional single-cell datasets. Sci. Rep. **9**(1), 8914 (2019). https://doi.org/10.1038/s41598-019-45301-0
38. Taud, H., Mas, J.: Multilayer perceptron (mlp). Geomatic approaches for modeling land change scenarios, pp. 451–455 (2018). https://doi.org/10.1007/978-3-319-60801-3_27
39. Tavassolipour, M., Karimian, M., Kasaei, S.: Event detection and summarization in soccer videos using Bayesian network and copula. IEEE Trans. Circuits Syst. Video Technol. **24**(2), 291–304 (2014). https://doi.org/10.1109/TCSVT.2013.2243640
40. van den Tillaar, R., Bhandurge, S., Stewart, T.: Can machine learning with IMUs be used to detect different throws and estimate ball velocity in team handball? Sensors **21**(7), 2288 (2021). https://doi.org/10.3390/s21072288. (**Part of special issue: Sensors in Sports Biomechanics**)
41. Verucchi, M., Brilli, G., Sapienza, D., Verasani, M., Arena, M., Gatti, F., Capotondi, A., Cavicchioli, R., Bertogna, M., Solieri, M.: A systematic assessment of embedded neural networks for object detection. In: Proceedings of the 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2020), pp. 937–944. IEEE (2020). https://doi.org/10.1109/ETFA46521.2020.9212130
42. Wold, S., Esbensen, K., Geladi, P.: Principal component analysis. Chemom. Intell. Lab. Syst. **2**(1–3), 37–52 (1987). https://doi.org/10.1016/0169-7439(87)80084-9
43. Wolpert, D.H.: Stacked generalization. Neural Netw. **5**(2), 241–259 (1992). https://doi.org/10.1016/S0893-6080(05)80023-1
44. Zhao, B., Lu, H., Chen, S., Liu, J., Wu, D.: Convolutional neural networks for time series classification. J. Syst. Eng. Electron. **28**(1), 162–169 (2017). https://doi.org/10.21629/JSEE.2017.01.18
45. Zolfaghari, M., Ghanei-Yakhdan, H., Yazdi, M.: Real-time object tracking based on an adaptive transition model and extended Kalman filter to handle full occlusion. Vis. Comput. **36**, 701–715 (2020). https://doi.org/10.1007/s00371-019-01652-3

Omar A. Mures M.Sc. in High-Performance Computing, is a researcher and instructor at the University of A Coruña (UDC), Spain. As a Computer Architecture Group (GAC) member, he specializes in applying High-Performance Computing to various domains such as Artificial Intelligence (AI), Computer Vision (CV), and Computer Graphics (CG). His research interests mainly focus on cutting-edge topics such as synthetic data generation, segmentation, detection, data analysis, and real-time rendering. He has also worked with several industry partners like Cinfo, S.L.

Javier Taibo Ph.D. in Computer Science, is an Associate Professor in the Department of Civil Engineering at the University of A Coruña (UDC), Spain. He is a member of the "Models and Applications of Distributed Systems" (MADS) research group. Dr. Taibo's research interests include computer graphics, real-time rendering, terrain visualization, virtual reality, multimedia, panoramic imagery, video encoding, streaming and GPU computing. He is co-inventor of two patents licensed by UDC to industry partners and has co-founded the UDC spin-off companies: Ilux Visual Technologies, S.L. (2007); and Syntheractive, S.L (2011), sold to CINFO, S.L.

Emilio J. Padrón is researching and teaching in the University of A Coruña, Spain, where he received the Ph.D. degree in Computer Science in 2006. As a member of the Computer Architecture Group, his research is focused on the application of High Performance Computing to Data Analytics (mainly targeting scientific simulations) and to the triangle Computer Graphics-Computer Vision-Artificial Intelligence.

Jose A. Iglesias-Guitian is a Research Scientist, "Ramon y Cajal" and InTalent fellow, at the University of A Coruña (UDC) and CITIC. His areas of expertise span visual computing, computer graphics, scientific visualization, and rendering, as well as their intersection with computer vision and machine learning. José worked before with various organizations, including the Computer Vision Center (UAB), Disney Research, the University of Zaragoza, and the Visual Computing Group at CRS4 (Italy). He holds a Ph.D. in Electronic and Computer Engineering from the University of Cagliari (2011) and an M.Sc. in Computer Science from UDC (2006).