Light-Weight Tracker for Sports Applications

Vitalii Vovk, Sergiy Skuratovskyi, Pavlo Vyplavin, and Ievgen Gorovyi
It-Jim
1 Konstitucii Sqr., Kharkiv 61045, Ukraine
vitalii.vovk@it-jim.com, ssnake@it-jim.com, pavlo.vyplavin@it-jim.com, ceo@it-jim.com

***Abstract*— In the paper, we describe technical details of multi-player sports tracker system. We demonstrate that proper in-depth analysis of video frames sequence may provide a lot of useful information required for sports analytics. Object detection and tracking steps are analyzed. Novel ideas for efficient filtering of false detections and irrelevant tracks are proposed. Also, we show an example of how the object tracking information used for the regions of interest location allows streaming sports events without human operators. At last, important practical implementation details, as well as initial experimental results, are discussed.**

Keywords—computer vision, sports tracking, OpenCV, camera controller

I. INTRODUCTION

Artificial intelligence (AI) is a rapidly growing discipline, providing plenty of possibilities for process automation, human-computer interaction, smart systems, internet-of-things (IoT), etc. Since most of the information goes through the human visual system (HVS), visual scene understanding has a very high impact on the efficiency of AI systems and applications. Computer vision (CV) [1] discipline provides an extensive list of methodologies for an automatic interpretation of visual information (images, videos) and useful information extraction [2].

CV in sports has a particular interest [3], and various practical applications are considered. In [4], a system for detection of soccer goal shots is described. Example of an algorithm for goal event detection itself is described in [5]. Ball detection and tracking is also a quite popular research direction [3], [6]. Nowadays, one of the most actively studied applications is related to human tracking algorithms [3], [7]-[8].

In the paper, we consider important technical details of a custom multi-player sports tracker system. The block-scheme of an image-processing part of our system is shown in Fig. 1. It consists of four main and three additional optional blocks:

1) A ***detector*** is responsible for primary object detection on a raw input image;
2) ***Object filter*** filters-out non-players and non-referee objects;
3) ***Object tracker*** assigns detected objects to existing tracks or makes new tracks for unassigned objects;
4) ***Track filter*** drops tracks, which are unlikely to be true and important, based on the object's velocity and position;
5) ***ROI estimator*** finds the most relevant part of the current frame;
6) ***Camera controller*** emulates camera movement by estimation of the current camera position and zoom, given previous camera state and current ROI;
7) Finally, the ***transformer*** prepares an output frame by cutting the estimated camera rectangle from the input frame and, optionally, applying any additional transformations.

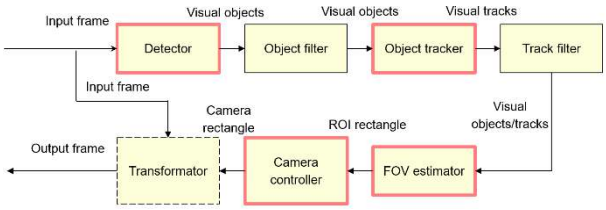

Figure 1.Image processing pipeline of custom sports tracker.

Object filter and track filter can be considered as parts of the detector and object tracker, respectively.

In this paper, we describe only the main procedures of our sport-tracker system: object detection, object tracking, ROI estimation, and camera controller emulation.

The rest of the paper has the following structure. Section II describes the principles of object detection and filtering. Developed object tracking pipeline is discussed in Section III. In particular, we define an adaptive re-weighing principle, state update, and player trajectory extrapolation procedure. In section IV, we describe the main principles of stable ROI estimation and smooth camera movement emulation. Section V contains information about some of the practical improvements of the tracker and its real application. Finally, we make conclusions about the described system.

II. FAST OBJECT DETECTION

Detection is one of the most important parts of the pipeline. Our detection stage consists of the following steps:

1) background estimation;
2) moving object detection;
3) object filtration.

A. Background estimation

In our system, objects are parts of the image that differ from the background. From this definition, it is clear that objects can be found from the difference between the current frame and background (Fig. 2,*a-b*). Since detection conditions can vary over time, we need to estimate the background dynamically. The following ways of background estimation are possible:

1) Correct one: set background buffer size N and step k. For each i^{th} frame F_i, where $i=j\cdot k$, $j = 0,1\ldots$, we update background as:

$$\begin{aligned} B_j &= \left(B_{j-1}\cdot j + F_i\right)/(j+1) && \text{if } j < N, \\ B_j &= \left(B_{j-1}\cdot N - F_{i-kN} + F_i\right)/N && \text{if } j \geq N. \end{aligned} \quad (1)$$

In other words, we estimate the mean frame in a floating window of size N. This requires holding a buffer of depth N for input frames, which can be an undesirable effect because of the additional memory consumption.

2) Fast one: set background "buffer size" N and step k. For each i^{th} frame F_i, where $i=j\cdot k$, $j = 0,1\ldots$, we update background as:

$$\begin{aligned} B_j &= \left(B_{j-1}\cdot j + F_i\right)/(j+1) && \text{if } j < N, \\ B_j &= \left(B_{j-1}\cdot (N-1) + F_i\right)/N && \text{if } j \geq N. \end{aligned} \quad (2)$$

Equation (2) describes background as an autoregressive model of first-order $x_j = \alpha x_{j-1} + e_j$ with $\alpha = (N\text{-}1)/N$ and $e_j = F_i/N$.

If color representation is discrete (any integer types) and thus has a limited number of levels, we need to estimate buffer size N carefully. It can be shown that to update background value x to x+1 we need the value of the current frame to be at least

$$y \geq x + 0.5N. \quad (3)$$

To update background value x to x-1 need the value of the current frame to be at most:

$$y < x - 0.5N \quad (4)$$

For example, for 8-bit images (256 possible color levels) this means that it is impossible to increase background if it has value $x >$ (255-0.5N), and decrease for $x \leq$ (0.5N). So setting N to some big number can lead to the non-updatable background, and setting N to some small number leads to the stronger influence of the current frame, and thus reduces filtration properties (background stability).

B. Moving object detection

We detect only objects that move over time and thus can be discriminated from the background. Objects are blobs, which are detected on binarized images after subtracting the background from the current frame (Fig. 2,*c*):

$$D = \begin{cases} 1\ if\ \text{abs}(F-B) \geq t, \\ 0\ if\ \text{abs}(F-B) < t, \end{cases} \quad (5)$$

where t is a threshold, which defines the sensitivity of the detector, abs(·) is the operator that returns the absolute value of its argument.

To reduce blobs fragmentation, the image is then filtered with the Gaussian filter and then is binarized for the second time with another threshold.

C. Objects filtering

The next step is to filter out the noise caused by artifacts in image and non-player movement (movement outside play-field). We use the size of the detected blobs to provide filtering. The problem is in a wide range of possible blobs sizes: players on backplane can have a small size, and near-plane false objects can have a relatively large size (Fig. 2,*d*). To normalize sizes of the objects, we use scale map (see Fig. 2,*e*) that has high scales for far objects and low scales for near objects.

Using such a scale map also helps to remove outliers, simply by setting small values for regions outside of playfield (Fig. 2,*f*).

After filtering, we estimate bounding boxes around detected objects and pass them to the next step, namely, tracking.

III. Object Tracking

Object tracking is a two-step process. First, we need to assign detected objects to existing tracks. Second, we need to update the state of all tracks with the new information coming from detections.

A. Assignment

At first, we need to assign newly detected objects to existing tracks. To do this, we find the distance between the object bounding box and bounding box of track extrapolated to the current frame.

Let us define i-th object's bounding box as b^i_{obj}, j-th track's bounding box as b^j_{track}. We define $|\cdot|$ as an operator, returning area of a rectangle (bounding box), and $\cdot\cap\cdot$ as an operator, returning overlapping rectangle of two bounding boxes. Then

$$\begin{aligned} p_{ij} &= \frac{\left|b^i_{obj}\cap b^j_{track}\right|}{\left|b^i_{obj}\right|}, \\ r_{ij} &= \frac{\left|b^i_{obj}\cap b^j_{track}\right|}{\left|b^j_{track}\right|}, \end{aligned} \quad (6)$$

and measure of similarity between i-th object and j-th track is defined as:

$$f_{ij} = \frac{2(p_{ij}\cdot r_{ij})}{p_{ij}+r_{ij}}. \quad (7)$$

Using (7), we can find the best match within detected objects for any existing track and assign them.

For objects that are not assigned to any track but still have a reasonable value of f_{ij}, we create new tracks as a copy of j-th tracks, and assign them. This is called a track *multiplication*.

For the rest of the objects, we create new empty tracks.

B. State update

Track state contains the next main parameters:

- the track's position (center of the bounding box);
- the track's velocity;
- the track's bounding box;
- detections count;
- consequent misdetections count.

For each new track, we set its state as follows. We set the track's position equal to the object's position (center of the bounding box), the track's velocity to zero, and the track's bounding box equal to the object's bounding box. We set the number of detections equal to 1 and the number of consequent misdetections equal to 0.

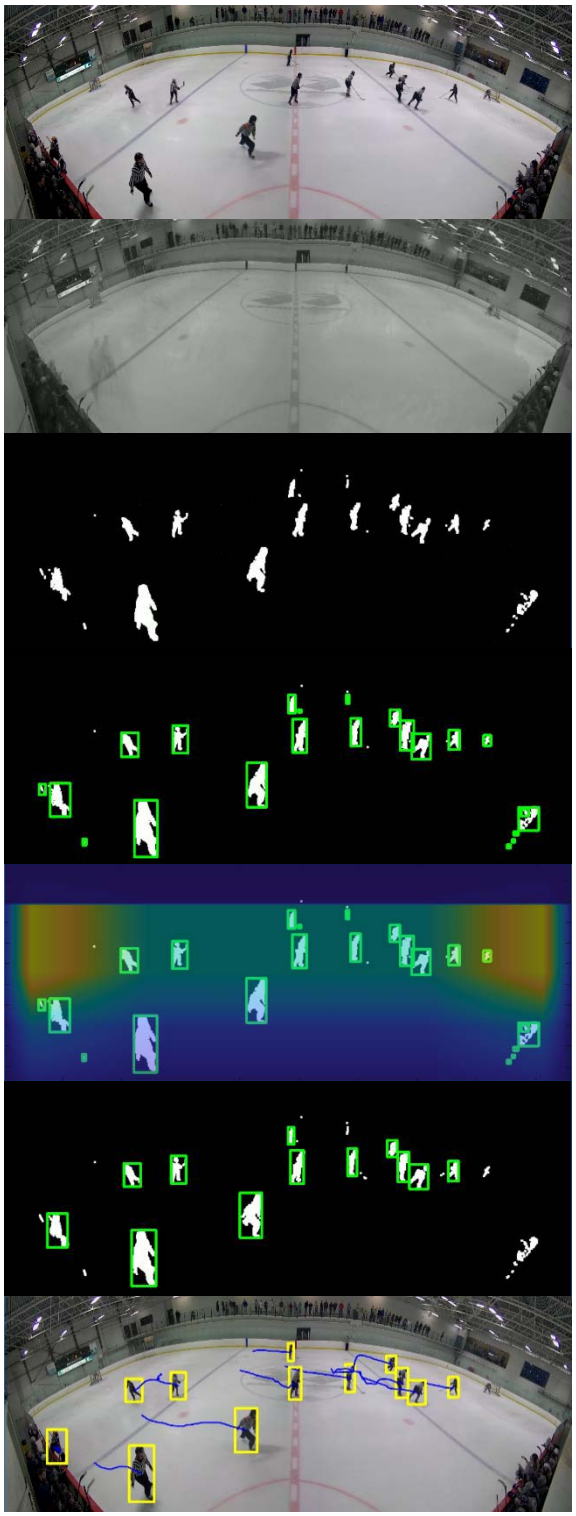

Figure 2. Blobs detection steps: source image (*a*), estimated background (grayscale) (*b*), diff frame after threshold applying (*c*), detected non-filtered blobs (*d*), scale map (*e*), filtered objects (*f*) and built tracks (*g*)

For each assigned and multiplied track, we update the state as follows. First, we update the track's position and velocity using the alpha-beta filter:

$$\begin{aligned} &\text{Prediction} && \mathbf{x}_k^* = \overline{\mathbf{x}_{k-1}} + \overline{\mathbf{v}_{k-1}}\Delta t \\ &\text{Residual} && \mathbf{r} = \mathbf{x}_k^* - \mathbf{x}_k \\ &\text{Update} && \overline{\mathbf{x}_k} = \mathbf{x}_k^* + \alpha \cdot \mathbf{r} \\ & && \overline{\mathbf{v}_k} = \overline{\mathbf{v}_{k-1}} + \beta/\Delta t \cdot \mathbf{r} \end{aligned} \quad (8)$$

where $\overline{\mathbf{x}_m}$ is the estimated track coordinate at the frame m; $\overline{\mathbf{v}_m}$ is the estimated track velocity at the frame m; Δt is the time step between the previous and the current frame; α and β are the alpha-beta filter parameters. Varying α and β, we can find an optimal balance between smooth tracking and filter sensitivity. After that, we set bounding box size equal to the size of the last detected bounding box, increment number of detections and set the number of consequent misdetections to 0.

For each unassigned track, we update the state as follows. We update track position and velocity using the alpha-beta filter:

$$\begin{aligned} \overline{\mathbf{x}_k} &= \overline{\mathbf{x}_{k-1}} + \overline{\mathbf{v}_{k-1}}\Delta t, \\ \overline{\mathbf{v}_k} &= \overline{\mathbf{v}_{k-1}}, \end{aligned} \quad (9)$$

and increment number of misdetections.

C. Track filtering

To reduce the number of false tracks, we return only tracks that satisfy the following conditions for each new frame:

- detections count exceeds the threshold;
- misdetections count is less than the threshold.

These steps allow to filter out short-time objects and objects that disappeared.

Example of tracked objects is shown in Fig. 2,*g*.

IV. Region of Interest Estimation and Camera Emulation

Object tracks have many possible further usage scenarios, for example, automatic camera controller (emulation of an operator).

Having updated track model parameters, we can find a region of interest (ROI), or bounding box surrounding most "interesting" objects within given constraints: aspect ratio, minimal/maximal size, frame size. To do this, we assign states of the tracks to clusters and find a subset of clusters that holds the maximum information and at the same time can be enclosed in the constrained bounding box. If two disjoint clusters cannot be enclosed in such a bounding box, the less informative one (less "interesting") is dropped from further processing.

A. Clustering

To find clusters, we use Ward's method, which relates to the Lance–Williams family of agglomerative hierarchically clustering algorithms [9] and minimizes the total within-cluster variance.

Ward's distance between two clusters is given by:

$$d(C_i, C_j) = \frac{n_i n_j}{n_i + n_j} \rho^2 \left(\sum_{w \in C_i} \frac{w}{n_i}, \sum_{s \in C_j} \frac{s}{n_j} \right), \quad (10)$$

where n_i and n_j are cardinalities of clusters C_i and C_j respectively, and $\rho(x, y)$ is an Euclidean distance between x and y. Thus, Ward's distance is a distance between clusters' centers weighted on their cardinalities.

Lance–Williams algorithm then recursively updates distances between clusters on every iteration t of cluster merge:

$$d^t_{(ij)k}(C_i \cup C_j, C_k) =$$

$$= \frac{n_i+n_k}{N} d^{t-1}_{ik}(C_i, C_k) + \frac{n_j+n_k}{N} d^{t-i}_{ij}(C_j, C_k) - \quad (11)$$
$$- \frac{n_k}{N} d^{t-1}_{ij}(C_i, C_j),$$

where n_i, n_j and n_k are cardinalities of i-th, j-th and k-th clusters respectively, $n_i + n_j + n_k = N$; i and j are selected to minimize inter-cluster distance at iteration t-1: $\min_{i,j} d^{t-1}_{ii}(C_i, C_j)$.

By sequentially merging closest clusters, we can build a dendrogram, which represents hierarchical relations between elements (see Fig. 3). Then, we form resulting clusters by setting threshold as a scaled difference of max inter-clusters distances:

$$T = c \cdot \max_t (d^{t+1} - d^t), \quad (12)$$

where d^t is a minimal pairwise distance between clusters at iteration t.

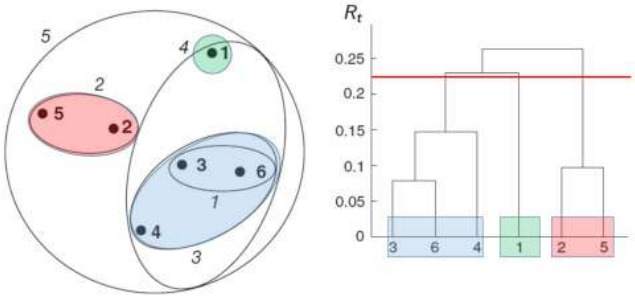

Figure 3. Clustering example. Clusters (left) and hierarchical dendrogram (right) [10]

The weight for each cluster is set as a sum of the weight of cluster members.

B. Outliers removing

Unfortunately, it is often the case when not all clusters can be enclosed by a constrained rectangular box (see Fig. 4). In such cases, we need to leave as maximum information as possible, but drop some clusters from being enclosed. The following algorithm achieves this.

First, we sort all of the clusters by distance from the weighted mean position:

$$d_i = \sqrt{(x_i - x_c)^2 + (y_i - y_c)^2},$$
$$x_c = \frac{1}{N}\sum_{i=1}^{N} w_i x_i, \quad (13)$$
$$y_c = \frac{1}{N}\sum_{i=1}^{N} w_i y_i,$$

where w_i is a weight of the i-th cluster, x_i, y_i are coordinates of the i-th cluster, N is a total number of clusters.

Then we repeat the following steps: we select a cluster with the minimal value of d_i, remove it from the sorted sequence and put it to the output cluster set U. We build a minimal-size constrained rectangle around set U. If it is possible, we compute values:

$$a = \frac{q_i}{q_{i-1}}, \quad (14)$$

$$b = \frac{s_i}{s_{i-1}},$$

where q_i is a weight of all clusters within the bounding rectangle (region of interest, *ROI*) at step i, and s_i is a ROI area at step i. Value a shows a relative increase of weight, and b shows a relative increase of ROI area.

If ROI breaks the constraints, or $\frac{a}{b} < t$, where t is a predefined threshold, which defines required minimal "density" of clusters, we remove the last added cluster from the set U.

The described approach is illustrated in Fig. 4.

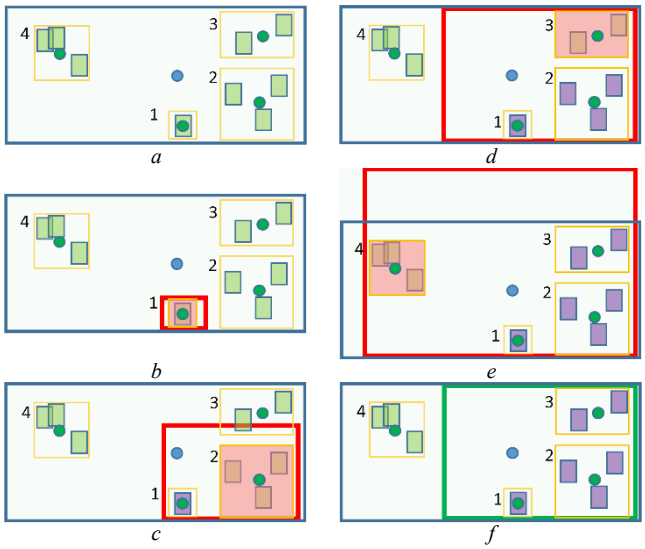

Figure 4. Outliers removing example: initial setup (*a*); sequential adding closest clusters #1-#3 to the output set (*b-d*); constraint violation for the cluster #4 (*e*); resulting ROI, enclosing clusters #1-#3 (*f*).

Legend: the red box is a current ROI; the pink rectangle is a cluster under test; the violet rectangles are objects that will be in the final subset; the green rectangles are detected and tracked objects; the blue circle is a weighted mean of clusters centers; the green circles are clusters' centers.

C. Smooth camera movement emulation

After the ROI is found, we can use it to emulate the operator's work, and update virtual camera position and zooming.

To provide a smooth camera movement, we use ROI buffering and filtering.

The buffer stores last M estimated ROIs (we use M=30 in our setup). Then, having a current camera rectangle on the i-th frame $r^i_c = [x^i_c, y^i_c, w^i_c, h^i_c]$, we estimate the weight of each ROI r^j in a buffer as an *intersection over union (IoU)* between the camera rectangle and ROI:

$$q_j = \frac{r^i_c \cap r^j}{r^i_c \cup r^j}, \quad (15)$$

where $r^i_c \cap r^j$ is an intersection of camera rectangle and j-th ROI, and $r^i_c \cup r^j$ is an intersection between them.

Then we use ROI with the largest q_j to update the current camera rectangle using alpha-beta filtering independently on each rectangle property. This allows changing both the camera position and zoom smoothly. We also use thresholds on minimal camera position and zoom changes to make camera behavior more stable and realistic.

V. Practical Realization and Improvements

In practice, computational performance of the proposed multiple-object sports tracker, as well as memory consumption, are very important. It is especially reasonable for embedding devices with low memory and no possibility to use high-parallel computations using GPU. To achieve reasonable performance, we follow the following rules:

- minimize the number of operations on images. We process images only at the first step (detection) for background estimation and blobs finding;
- if possible, downsize images before processing. We achieve good tracking quality on images of 100 px height (with input frames having 4K resolution);
- use simple and fast models. In our tracker we use linear objects movement model because of the relatively high frame rate compared to object speed;
- avoid the use of image buffers to reduce memory consumption. We do not use any image buffers in our tracker. The only images that stored in memory are low-resolution downsized current frame, background, and reweighing coefficients;
- optimize algorithms where possible to reduce memory usage and increase performance.

Following these rules allowed us to build extremely fast sports tracker with camera controller implementation that can be used in sport games translations. Our implementation can easily process 4K video in real-time on a single core of the i7-3630QM CPU.

Algorithms implemented in our sports tracker are general and can be applied to different kinds of sports activities, such as hockey, soccer, basketball, etc. with only minor changes in thresholds (see Fig. 5).

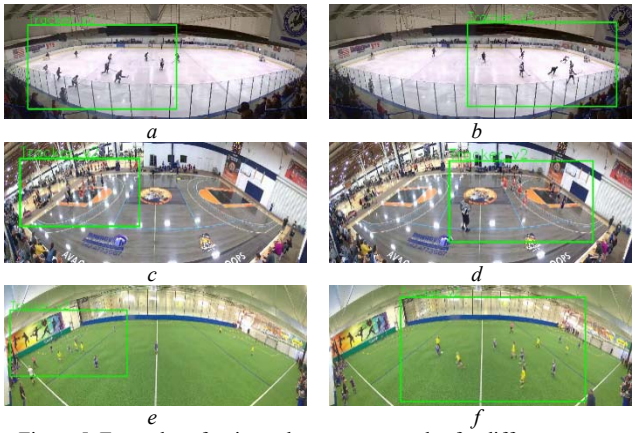

Figure 5. Examples of estimated camera rectangles for different sports activities: hockey (*a*, *b*), basketball (*c*, *d*) and soccer (*e*, *f*)

Conclusion

We have described several important aspects required to build a multi-object sports tracking system. Outputs of each image processing stage were analyzed and discussed. It was shown that proper blobs rescaling allows to filter out false object detections and keep only useful data. For tracking part, we have proposed smart control of the trajectory updates giving a way to discard unnecessary tracks. Finally, proper utilization of outputs from CV part and combining with virtual camera controller provided a smooth real-time picture of sports activities with no human operator engagement.

Proposed sports tracking system has low computational and memory requirements and can be applied to various sports activities.

References

[1] D. Baggio, S. Emami, D. Escriva, "Mastering OpenCV with practical Computer Vision Projects." / Packt Publishing, 2012.

[2] M. Doinea, C. Boja, "Machine learning techniques for data extraction and classification in computer vision software" / Proc. 13th Int. Conf. informatics in economy, Bucharest, Romania, 2014

[3] Computer Vision in Sports (Advances in Computer Vision and Pattern Recognition), Springer; 2014.

[4] Chen SC, Shyu ML, Zhang C, Luo L, Chen M, "Detection of soccer goal shots using joint multimedia features and classification rules." / In: Proceedings of the fourth international workshop on multimedia data mining (MDM/KDD2003), pp 36–44, (2003)

[5] D'Orazio T, Leo M, Spagnolo P, Nitti M, Mosca N, Distante A, "A visual system for real time detection of goal events during Soccer matches." / Comput Vis Image Underst 113(5): 622–632, (2009)

[6] Yu X, Sim C, Wang JR, Cheong L, "A trajectory-based ball detection and tracking algorithm in broadcast tennis video." / ICIP 2:1049–1052, (2004)

[7] Yi Wu, J. Lim, M.-H. Yang, "Online Object Tracking: A Benchmark", IEEE Conference on Computer Vision and Pattern Recognition, June 23-28, 2013.

[8] [8] Automatic multi-player detection and tracking in broadcast sports video using support vector machine and particle filter, "IEEE International Conference on Multimedia and Expo, July 2006", pp. 1629-1632.

[9] F. Murtagh and P. Legendre, "Ward's Hierarchical Clustering Method: Clustering Criterion and Agglomerative Algorithm" (2011).

[10] https://www.coursera.org/learn/vvedenie-mashinnoe-obuchenie