Ball tracking in a Volleyball environment

Luca Cazzola, Alessandro Lorenzi

SIV, AY 2023/2024

1 Introduction

The following document describes the solution together with the overall thought process to achieve ball tracking in a volleyball scenario, without the use of state of the art deep learning architectures, which are generally preferred for this kind of tasks. Part of the image data used in the dataset and all the videos used for testing are provided by the working team. The proposed solution will be at the end compared with a YOLOv5 model, trained on the same task.

1.1 Challenges

The main issue about tracking a ball in sports is related to its size and speed. Balls used in volleyball have generally a defined texture and shape, but it's hard to discriminate just by considering those properties:

- **Shape** : the ball tends to deform into an elliptical shape. On the top of that, contours blends with the background due to the ball's speed which is generally too high even for commonly used TV cameras to capture perfectly.
- **Color & Texture** : the ball is also often spinning, which makes color and texture information not always meaning full or available.

For the provided reasons, the first step in order to identify a set of **"ball candidates"** needs to rely on motion-related properties. That's because it's the only factor which is guaranteed to be constant in the context of volleyball (during an action the ball can't be stopped, otherwise that's considered a fault).

Figure 1: Frame example

The presented proposal is based first on identifying **moving entities** and construct a bounding box around them (**detection step**). After that a classifier is going to be trained to discriminate what regions are more likely to contain a ball and which are not (**classification step**).

2 Detection

This project works under the assumption that the **camera is fixed**. With the provided circumstances, an easy way to extract motion information would be applying the difference between the current frame and a background one. Since a **background frame** is not provided what could be used instead is the previous frame. Problem is that by doing so, in some cases it might happen that the ball moves so fast that the computed difference is affected by the noise of its previous position, which is not optimal. What can be done to avoid this problem is to estimate a background frame by sampling a good amount of random frames from the video and compute the median among them. That approach might include the ball itself as background, but it's very unluckily to do so since it's constantly moving. The result of this operation, if done into the gray-scale domain, will be a gray scale map in which gray scale is directly related to what evolves over time in the current frame (aka. moves since camera is fixed).

In a common volleyball match the main **moving entities** are the players and the ball. After that, there might be some noise led to the net (it might shake if hit by the ball), the referee and overall background noise. To reduce the unwanted information a set of **filters and morphological operations** is applied:

1. **Thresholding** : makes a better distinction between actual moving entities and slight variations in the image, the used threshold is based on the average gray scale of the image plus a constant factor.
2. **Erosion** : after thresholding is applied it's very common to have many isolated white spots due to slight changes in the scene exposure or other sources of noise which might exceed the threshold. An small erosion operation is helpful if the spots are small enough to be completely eliminated.
3. **Expansion** : useful both to restore the original size of remaining white regions and make them intersect with each other, joining together regions belonging to the same entity
4. **Median blurring** : helps to fill even more gaps and holes that might still exist.

At the end, what's obtained is a map of white regions representing the moving entities in the scenario.

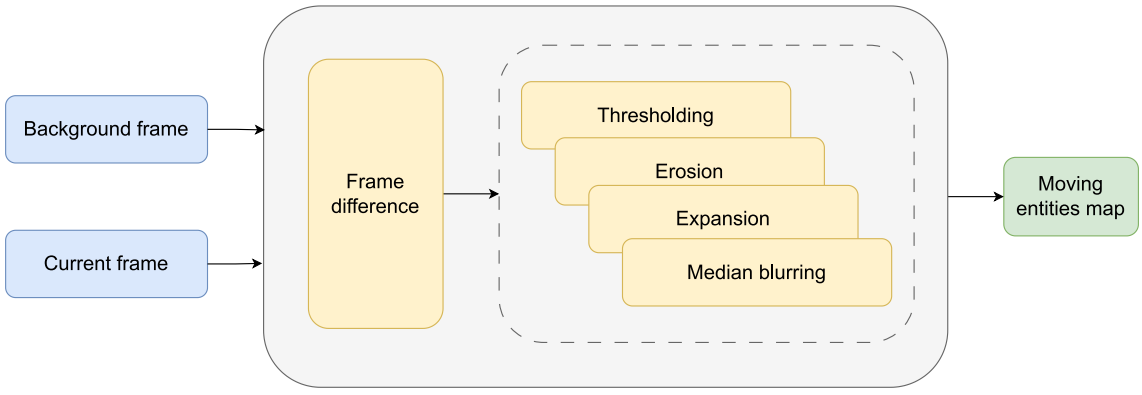

Figure 2: Moving entities extraction process

Now the objective is to **extract bounding boxes** from the obtained map. An easy way to do so is by computing the **contours** of the white regions. Each contour will ideally correspond to a different entity, so it's just needed to place a bounding box around each contour. The detected bounding boxes

might be a lot, but fortunately there are some criteria that can exploited to discard the ones which are most likely to not contain a ball:

- **Size** : put a threshold on a minimum and maximum size that the bounding box needs to have. Since the camera is fixed it needs to be in a position such that the whole playing field (or most of it) is represented, this implies the ball is pretty small, especially if compared with the players. The observed average size of a bounding box containing a ball has been shown to be in the range $[150, 3000]$ pixels on average.
- **Aspect ratio** : Since the ball has at most an elliptic shape, the aspect ration of the box generally is in between [0.5, 2]. Everything exceeding these limits can be really often safely discarded.

Here's an example of intermediate steps results on actual data

(a) Frame difference result

(b) Moving entities map

(c) Detected bounding boxes (displaced in red)

Figure 3: Image preprocessing chain

3 Classification

3.1 Features

Since the search space is much smaller compared to before it's possible to start reasoning on actual ball properties, which implies proper features need to be chosen. Good candidates are mainly related to shape, texture and color. In the last implementation of the project, the team decided to use **Histogram of Oriented Gradients (HOG)** features, which are powerful descriptors which basically build a distribution by using the counts occurrences of gradient orientation [3] in the localized portion of an image.

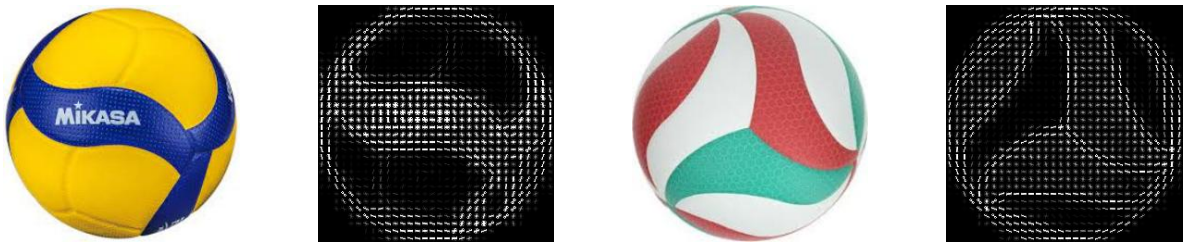

Figure 4: HOG features visualization

The shown examples are naturally optimal toy scenarios, in which HOG descriptors would be very effective at capturing both the shape and texture properties of the ball. On average, the actual input image would be **at best** more of something as follows

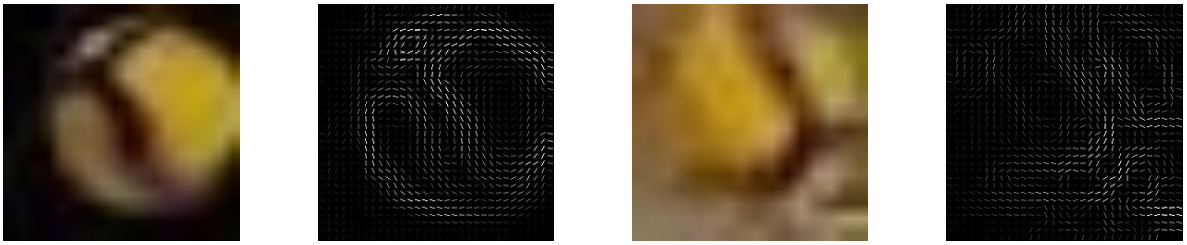

Figure 5: HOG features visualization - Real case (16 by 16 pixels regions)

Some information is retained as it can be seen.

3.2 Training & dataset

The chosen model to make classification with is **random forest**, which is very flexible and a solid starting point for many machine learning applications. Since a **negative set** is needed to make binary classification a good amount of images related to "non-ball" has been created by simply extracting random regions of the frames belonging to the video source. This approach might by chance include the ball in the extracted regions, but it's unluckily to perfectly generate a region containing just the ball, as it's only one in the entire scene and relatively small, which means it can't contribute much to the generated HOG histograms.

The dataset is pretty small, (~ 1200) positive samples (ball) and (~ 1000) negative samples (non-ball). In order to make the model less prone to overfitting, the number of features has been reduced to 20 with **PCA (Principal Component Analysis)**.

Here follows the training results :

	precision	recall	f1-score	support
0	0.97	0.91	0.94	160
1	0.94	0.98	0.96	221

Overall accuracy : 0.95

Figure 6: Random forest training results

Now it's possible to test how the model works on the video source. A smart thing to do is instead of running the class inference on the RGB domain for each bounding box, it's possible to extract features directly in the previously computed difference image. That makes the shape of moving entities more meaningful, especially by separating the background and enhancing contours.

Figure 7: Detection example

In the above example the bounding boxes in which the confidence threshold of positive class (ball) satisfy a threshold of 0.8 are displaced in green instead of red. Naturally there can be seen false positives.

In order to make a comparison a **YOLOv5** classifier has been trained on the same data (excluding the negative set since it's not needed).

4 Conclusions

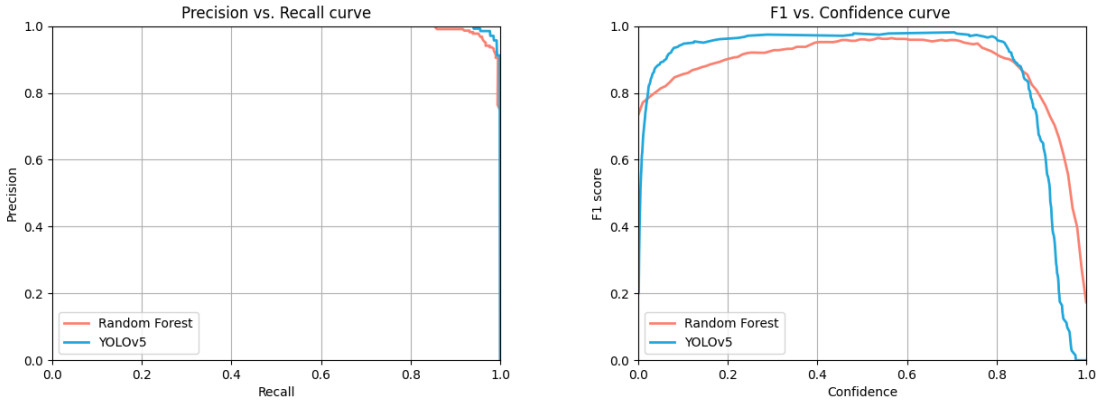

Figure 8: Random Forest - YOLOv5 comparison

As the Precision vs. Recall curve shows the HOG features seem to be good enough at discriminating between a ball and a non-ball object, there's not a significant difference with the trained YOLO model, which is supposed to provide a deeper and more flexible feature extraction. What can be said is that the two classifier behaves similar on the provided set. By looking at the F1 vs. Confidence curve the only noticeable aspect is that the Random Forest model seem to work much better respect to the YOLO one at higher confidence thresholds.

By running the two approaches on the same input video it's pretty clear that both methods are partially effective. Random forest classifier detects the ball more often and at higher confidence respect to the YOLO model, but it's also true that more false positives are detected. With no surprise, YOLO is much more effective at detecting a ball when its texture is well defined, which happens when it's moving slow and it's not rotating too fast. On the other hand, the provided Random forest solution seem to be more effective on fast moving scenarios, in which the most problematic part is the detection phase itself.

Considering the reached performances together with the fact the used dataset is pretty small (ball samples are just $\sim$ 1000) the reached result is promising. Having more good data available would probably considerably improve the performance of both models and enhance the differences between the two, probably in favour to the YOLO model by intuition. A possible alternative solution might also be an union of the two presented techniques. For instance, what could be done is applying the YOLO feature extraction to the regions obtained in the detection step, bypassing the task of identifying the ball's region of interest, which is the most challenging part when it's moving rapidly.

Code available at [2]. **Dataset available** at [1]

References

[1] Luca C. & Alessandro L. *Dataset*. Version 0.1. Dec. 2023. URL: https://universe.roboflow.com/lucazzola-ai-projects/ball-detection-volley/dataset/4.

[2] Luca C. & Alessandro L. *volleyball-BallTracking*. Version 0.1. Dec. 2023. URL: https://github.com/lorenzialessandro/volleyball-BallTracking.

[3] Mrinal Tyagi. *HOG (Histogram of Oriented Gradients): An Overview*. July 2021. URL: https://towardsdatascience.com/hog-histogram-of-oriented-gradients-67ecd887675f.