Computer Vision: Analyze a Basketball video

Ivan Martini
MSc in Computer Science, DISI
Trento, Italy
ivan.martini@studenti.unitn.it

https://github.com/iron512/CVassignment

***Abstract*—As the title says, the goal of the second assignment is to produce a code to analyze a Basketball video, more specifically a fragment of the match of the Aquila Basket Trento, playing home court. The requirements were quite general, giving a lot of degrees of freedom on which strategy to adopt. The goal of this report is not only to provide the information needed to setup and run the code, but also to explain some implementative choices I took, bringing in my point of view.**

I. Introduction

Since the goal of this essay is to just present my work, I decided to keep it short and easy, focusing on the parts that I find more relevant and interesting, giving also a peek of the decisions I took. The requirements were clear:

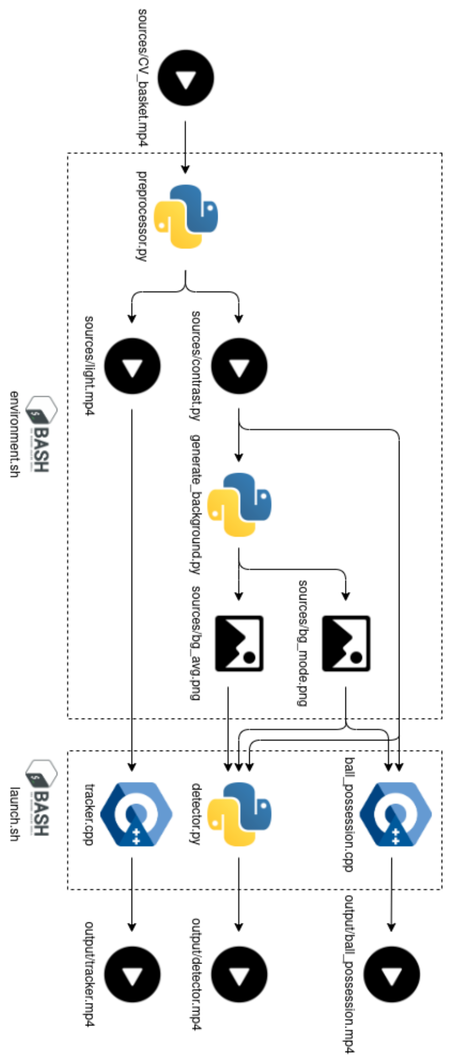
- Run a people detector on the whole video. Write on the output video the number of people detected at each frame in the scene. Qualitatively evaluate the accuracy of the detector when reporting the results.
- Choose one person in the scene (either player or referee) and try to track him. Plot the trajectory on the output video. Qualitatively evaluate the obtained results.
- Detect how many times the ball possession change (e.g. all 10 players go from one half of the court to the other one). Display the result on the output video.

I decided to split the three tasks in three separate and independent settings, in order to provide a cleaner and consistent code, which is easier to understand. Furthermore I decided to develop some section using `C++` and others using `Python`. Although the openCV lib is the same, I found some bug due to a mismatch on the supported languages (Using `Python 3.9` and the last version of openCV, gives some errors on tracking for example). I also wrote two `bash` scripts in order to avoid the manual configuration of the whole project. All the materials regarding both assignment can be found in the repository linked at the top of this report. Finally I decided to pursue the goal using just **computer vision tools** and avoid the use of pre-trained AI models, in order to dig more into the subject itself. This might have led to a minor quality of the output, but I am happy of the result and about what i learnt.

The structure of this report is build as follow: In the 2nd section there will a brief introduction on the global architecture and exaplantion of the complete pipeline while in the 3rd there will be a more detailed view of the three modules that address the 3 tasks. Finally the setup, which is a mere adaptation of the `README.md` file, with some personal reflection and a conclusive section to draft some further possible works and evaluate the whole work.

II. Overview of the Pipeline

The folder structure might look a bit confusing, since the number of file. To aid the understanding, I developed a graphical representation of the pipeline, presented below. All the important files are explained here, others indeed doesn't even appear, as they are just a set of auxiliary functions.



The pipeline starts from the the original video, which must be placed in the `sources` folder and be named "`CV_basked.mp4`". All the modules of the pipeline take the video as parameter, but in order to have the `bash` script to correctly run, these constraints are mandatory. The first module is called `preprocessor.py` which outputs, in the `sources` folder two more videos, both with a strong **gamma correction** applied:

- `CV_basket_light.mp4` has a gamma correction parametrized with $\gamma = 0.5$, resulting in very lightened video, which is going to be used in the tracking module. The gamma correction here prevent the tracker to lose the selected player (Morgan, #20), which wears a black jersey when it overlaps with the dark border of the field, rising the difference between the two
- `CV_basket_contrast.mp4` has a gamma correction parametrized with $\gamma = 1.7$, resulting in a rise of the contrast. In this way, the white jersey players pop up more on the field and their detection is more feasible (needed for both the detection and ball possession modules).

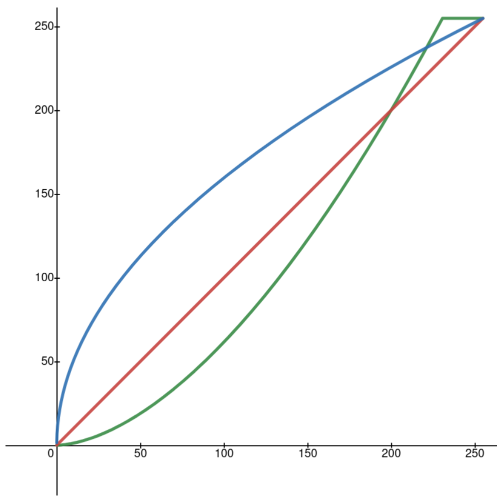


Fig. 1. The two gamma function used: the one for lightening the video ($\gamma = 0.5$, in **blue**) and the one for darkening it ($\gamma = 1.7$, in **green**), compared with the normal color mapping (in **red**)

After this step, the *highly contrasted* video gets through the `generate_background.py`, which is able to produce a clean background starting from the video. Basically one frame each 18 gets stored and using the **average value** or the **mode value** for each single pixel an image gets reconstructed, representing *the field without the players*. In a standard setting this image could be easily obtained by taking a picture from the camera before the match, but since this data were not available, I decided to recreate it.

The logic behind this reasoning is pretty heuristic, as I just bet on the fact that, even if the players go back and forth, each spot of the field will be empty most of the times, allowing me to get the empty field just by looking at the **mode of each pixel**. Furthemore the need for keeping both the average and the mode will be explained. All the operation described above, plus some other setup steps (as the compilation of the `CMakeLists` or the output folder creation) are the one covered in the `environment.sh` script.

The `launch.sh` script handles instead the three modules which actually perform the three required tasks. Each script writes a video in the `output` folder, which is the result of the computation done. The whole pipeline takes more or less 15 minutes to execute, due to the high computational cost of the reconstruction of the background.

III. Addressing the Tasks

A. Detection

The detection is the only one of the three final modules developed in `python`. The flow of the code starts from the enhanced video (the one with the darker gamma correction) and the background generated images (the **average** and the **mode** of the pixels). While I was deciding if the background subtraction were working better with one or the other, I noticed that in the region with an high variability (i.e. the adv banners and the screens in general) the difference between the mode and the average is quite marked. Through a subtraction of the images and a thresholding opportunely set, It is possible to produce a binary map white everywhere and black in those zones.

At this point it is possible to apply an `AND` function to the map without the billboards and the map produced by the background subtraction (the mode is used, but the avg works fine too) to have a clean map which excludes the billboards, which bring lot of noise to the image. During these steps, some erosion/dilation are applied, but the most interesting is the last one, which uses a non-squared kernel (9×3) to merge the region of the players. The choice of this value comes from the direct observation of the heuristic idea that there is the need of prioritizing the vertical merging than the horizontal one (the players are expected to be standing, not lying, ofc).

Now the obtained map is passed to the *findContours* function which produces the bounding boxes. Those bounding boxes are then checked to have some parameters (like a certain area or a certain ratio between height and width), discarding the non-suitable ones. The *white boxes* are the ones picked with a good confidence (1 or 2 people, depending on the occlusions), while the *gray boxes* are the ones which often represent many people together.

B. Tracking

For the tracking module, i chose to switch on C++. Other than the efficiency reasons, which I cared less, I had to change language, due to some "bug" I found during the development. I guess that the cause were some mismatch with the versions of OpenCV and Python 3.9.5 (as a virtualized machine with python 3.8 was working fine) but having no clue, I decided to change and "accept the challenge" of learning OpenCV also on this language.

I started tracking a referee using some standard OpenCV trackers, and the result were quite good, as the tracker were able to follow the human **most of the times**. However i had quite some time, so i decided to try tracking a player, which is not a requirement but far more interesting, dealing with occlusion and fast movements. After some iterations and refinements i produced a tracker which is able to follow **Jeremy Morgan** (The #20 of *Aquila Trento Basket*, the black team) over more than 1200 frames without interruptions, resulting in the successful tracking of 54% of the video (1200/2240).

The whole system works on the gamma corrected version, which allows the black jersey to be more evidently separated by the black outer region of the field. Three different CSRT tracker are initialized, as they are the most reliable ones (and of course expensive). Since most of the movement are done along the horizontal axis, the tracker are initialized with the same *dimension (w/h)* and *y position* but they are traslated over the *x position*. This results in a tracker for "the left part" of the player, one for the "right one" and one for the whole person. The three trackers gets updated at each frame and their relative position is recorded. In this way I had the possibility to develop a correction system, which takes their relative position and their average displacement: If a tracker gets too far (w.r.t its precedent values) from the other two it gets reset to where it should be (i.e. on the player or over its left/right side). Through these counter-measures I was able to keep the tracking of the player around frame 450 (which were lost due to the poor contrast with the field), frame 720 (minor occlusion) and frame 970 (major occlusion), improving a lot the result of the plain CSRT trackers.

C. *Ball possession*

The last task were to determine the ball possession. My first idea was to implement a Kalman Filter, or some predictive tracking objects on the ball, and use the relative position or the histogram of the near area to determine the team. This solution however didn't work at all as I expected, so i decided to move on something simpler but still effective. Just tracking the movement of the players is more or less enough to determine the attackers and the defenders.

Using just a background subtraction algorithm with the generated background it is therefore possible to produce the binary map representing the areas where there are some movements. Summing each half makes easy to determine where the players are and who's holding the ball. When plotting the output, I decided to insert a little margin of confidence, which checks if the number of white pixel is 10% bigger on one of the two size, and if not just sets the possessor as unknown. This results in few part of the video without any prediction, but when the the ball possession is determined, the confidence is higher.

IV. Setup

This section is the section which provides the necessary steps to execute the code. Most of the job is already done by the scripts (which you can read and understand), but still a few tweaks are necessary.

Note: Any changes will be reported in the `README.md` *file, inside the repository. Check that to be sure to run the correct commands*

First is mandatory to have `python` installed, `opencv` and `cmake`. I runned the code on two different OS (Ubuntu and Archlinux), and it worked fine on both, so i *suppose* that these are the only requirements. I had no chance of running it on Mac (finger crossed), and i definitely not want to run it on Windows.

Then copy the `CV_basket.mp4` video in the `sources` folder and preserve the name. This operation is not mandatory for the pipeline itself (all the files take parameters), but is **required** (and suggested) in order to use the bash scripts.

Run the code for enabling the `bash` scripts to be run (might require *sudo* permissions):

```
chmod +x ./environment.sh
chmod +x ./launch.sh
```

Set the environment
```
./environment.sh
```

And launch the code
```
./launch.sh
```

V. Conclusion and Evaluation

During the development of this project, I appreciated a lot better what has been taught in class, building a better understanding of some ground concepts of Computer Vision. As required there is a qualitative evaluation of the work done.

a) Detection: The detection has been the hardest part, as I wanted to develop most of the detector on my own. The use of the artificial background helped me a lot, producing a clean subtraction, where many method I tried failed in this sense. These result are however to be explained. Looking

Least people found	3
Most people found	14
Average people found	8.25

at the video is clear that most of the *low people count* frames are due to the inability of my algorithm to distinguish how many people are there in presence of occlusions or dense groups of people. On the other hand, the people to detect should be more or less 15 (10 players, 3 referees and 2 coaches), and in few frames the algorithm identifies a person in places where just a shadow is reflected on the field.

As explained, the algorithm relies only on the binary map of the background subtraction (opportunely transformed) to detect the movement. Using some segmentation algorithm based on color regions or histogram could increase a lot the accuracy.

b) Tracking: The tracking was really interesting, even if most of the job has been executed by the tracker object itself. For this reason I tried to tweak different parameters in order to improve accuracy. As explained above, the original tracker were losing the player after more or less 450 frames, while using the combination of *gamma correction* and the "three trackers" *correction algorithm* I extended the duration to 1200 frames ca. These improvement brought the portion of video where the tracker is able to follow the player from 20% to 54%. I guess that tweaking these parameters could only improve slightly the result, while having a multiple camera view or a different perspective (from the top, which reduces the occlusions) could really change the game here and provide a more reliable result (with clearly higher development costs).

c) Ball possession: This section is the easiest to evaluate, as the result is quite consistent. An idea that came to my mind to have the algorithm more reactive (as now the detection of the change is slightly late) could be checking the *24 seconds timer* areas, and try to identify when the clocks reset.

The ball possession changes 4 times, and all 4 times are detected consistently by the algorithm. There are few frames when the owner is uncertain, but no *false changes* are detected.