Group name: CV 2024

Barizza Marco 2102412 - number of working hours: 170

De Vidi Riccardo 2131888 - number of working hours: 180

CV final project report

Sport video analysis for billiard matches

July 19, 2024

1 Introduction

1.1 Project goal

The developed computer vision system, given a video of a billiard match, provides high-level information about the status of the match and displays the positions of the billiard balls in a 2D top-view minimap for each frame of the video.
The system should be able to:

- recognize and localize all the balls inside the playing field, distinguishing them based on their category;
- detect the boundaries of the playing field;
- segment the area inside the playing field boundaries into the categories “cue ball”, “8-ball”, “solid balls”, “striped balls” and “playing field”;
- represent the current state of the game in a 2D top-view visualization map, to be updated at each new frame, with the current balls positions and the trajectory of each ball that is moving.

1.2 Assumptions and constraints

In the development of the system the following assumptions are made and the following constraints are imposed:

- the pose of the camera does not change during a single video clip, so the table remains in the same position in the image for all frames of the same video;
- the computer vision system can work offline;
- the segmentation and the classification steps should be done in an extemporary way for the first frame (i.e. the performances can not be improved by exploiting the knowledge given by the tracking phase) while for the last frame the tracking process can be exploited to enhance the performance;
- the time needed to run the algorithm should be short (at most about 4 times the video duration).

1.3 Notes about the delivered code

The computer vision system can be called from the command line of the virtual machine with `./billiardAnalysis <clip> <saveOutput>` (optional)
(e.g. `./billiardAnalysis game1_clip1 1` analyzes the video “game1_clip1.mp4” and saves the intermediate results of the algorithm and the obtained performance scores).
The Dataset folder is supposed to be placed in the same folder of the executable file and to have the same structure of the one linked to the assignment for what regards its subfolders and filenames. Similarly, the Minimap folder is supposed to be placed in the same folder as the executable. This folder contains the minimap378200.png image and two text files, minimapHoles.txt and minimapPlayingFieldBoundaries.txt, which describe the minimap properties. In particular, minimapHoles.txt contains the minimap holes coordinates (x, y), one hole per row, from the upper left to the down right ones, while minimapPlayingFieldBoundaries.txt contains the four minimap playing field corners coordinates (x, y), one per row with the same order.
From Section 2 the document structure substantially follows the one of the main file of the delivered code, “main.cpp”:

- playing field detection;
- warp matrix computation;
- balls localization;
- bounding box filtering;
- balls tracking;
- balls classification;
- 2D top-view visualization map generation;
- performance measurements (with filtered output of the tracking phase).

2 Playing field detection

The first step is the detection of the playing field, which is defined as the surface of the table occupied by the cloth. It is observed that from a top-view of the billiard table the playing field can be approximated with a rectangle, that all the videos of the given dataset do not seem to be affected by any distortion effect and that all the playing fields are completely inside the videos. Thanks to these observations all the playing field borders can be approximated with an affine transformation of a rectangle in each frame of each video. In a general setting, in which the camera position is not fixed with respect to the playing field during a video, it would be necessary to detect the playing field in all the frames of the video. Since by assumption the pose of the camera does not change during the video, the playing field remains in the same position in all the frames and so it is only necessary to detect it once and then extend the obtained result to all the frames of the clip. Riccardo thought of generating a mask of the playing field with the following steps:

1. detect the playing field borders;
2. approximate the detected borders with lines;
3. evaluate the intersections of the found lines to obtain an approximation of the playing field corners;
4. fill the shape described by the corners.

2.1 Playing field borders detection

The first consideration made regards the choice of which frame to use for detecting the playing field borders. Riccardo noticed in fact that in the first frames of all the clips one border of the field is partially occluded by a player while, on the contrary, in almost all the last frames of the clips all the borders of the field are not occluded (Figure 1). These observations bring us to uniquely rely on the last frames of the clip to identify the borders of the fields. This choice clearly prevents our computer vision system from working in real time, but this is not esplicitly expected by the assignment, and this choice should lead to better results than estimating the field border on the first frame.

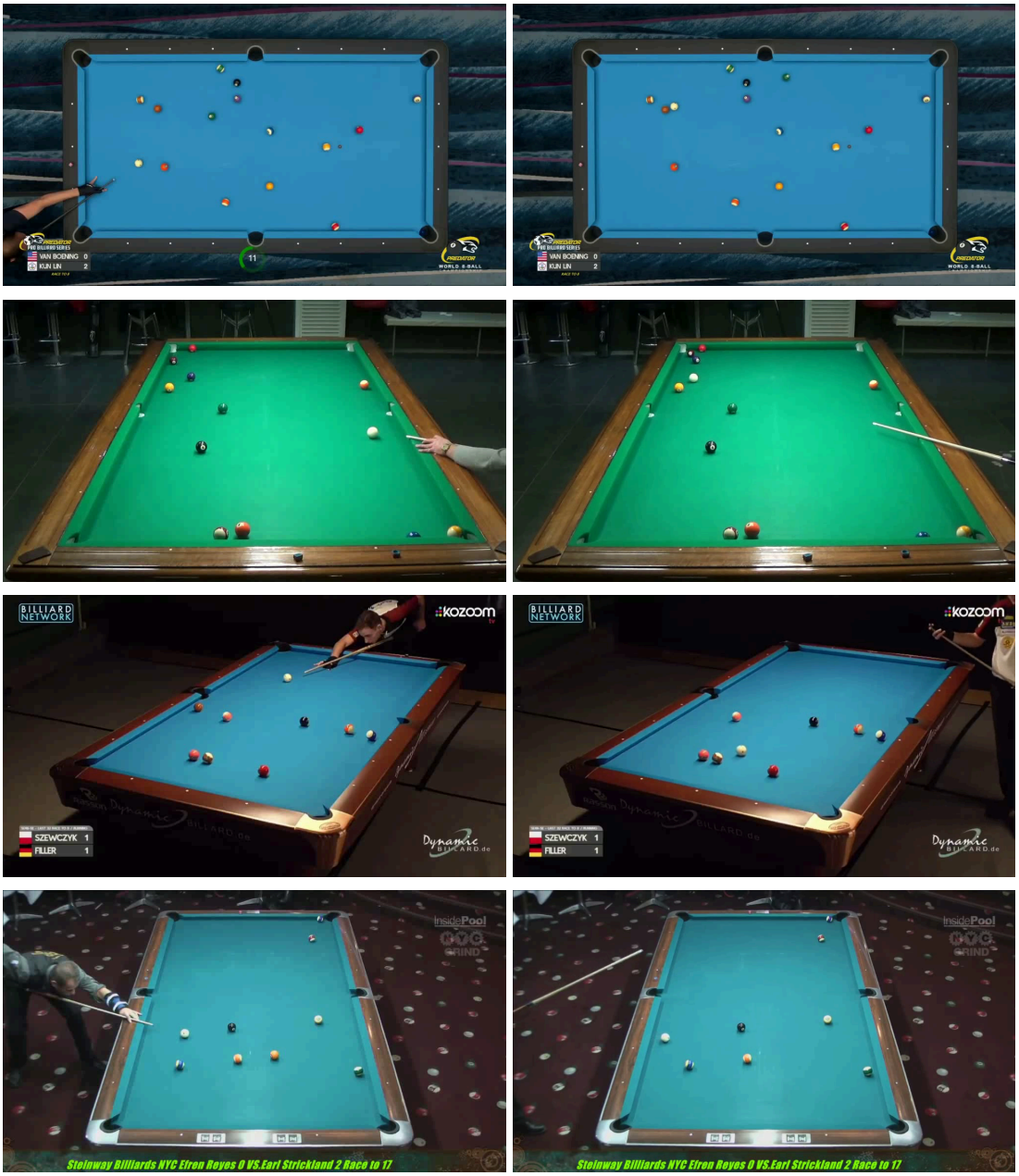


Figure 1: First (left) and last (right) frame of some videos

Riccardo decided to detect the borders by analyzing a linear combination of the BGR channels of the last video frame proceeding as follows. Firstly, evaluate the combination of

channels B + G - R to obtain a single channel image with high values in all the pixels with high B and G channels values and a low R channel value; this is done since all the playing fields in the dataset are blue or green and so the pixels belonging to the playing field should tend to have high B and G values and low R values, and hence high B + G - R values. It is interesting to notice that the sum of three CV_U8 values can, in principle, take values greater than 255 but the OpenCV function add() implements a saturation mechanisms such that the maximum values that can be taken by the sum of two Mat CV_U8 objects is 255 (thus avoiding overflow). The same mechanism prevents the minimum value assumed by the combination of channels B + G - R from being lower than 0.

Figure 2: game2_clip2 B + G - R

Going on, a binary threshold is applied to the BGR combination in order to discard all the pixels with a low value with respect to the playing field ones. These pixels are assumed to be part of the balls or of the background. As can be seen in Figure 3 this is not always the case and a better result for game2_clip2 could have been achieved setting a lower threshold. It is observed that all the playing fields that present low brightness shadows present this problem since in these darker areas the sum B + G assumes low values.

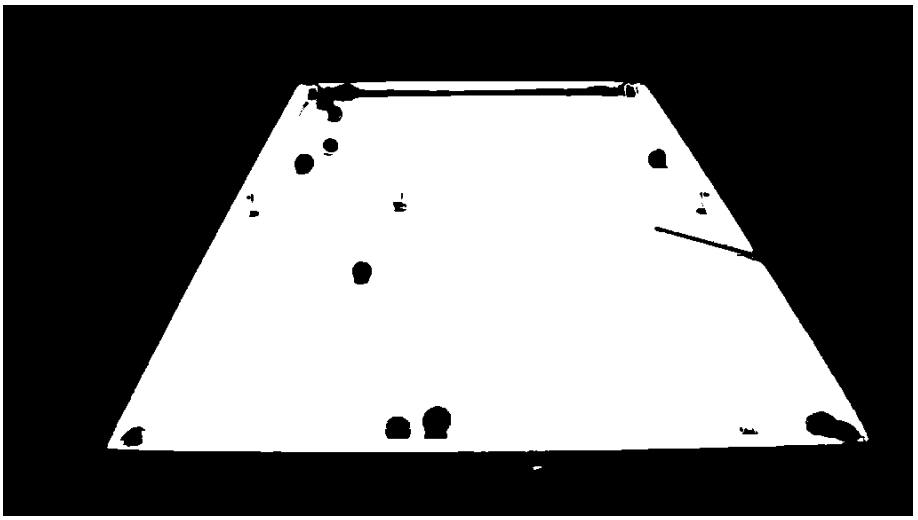


Figure 3: game2_clip2 B + G - R threshold

To conclude, the borders are found by applying to the resulting binary image the Canny algorithm. By this procedure an image with a low number of edges can be found in each clip in contrast to what would result from applying the canny algorithm directly to the BGR image (in which a large number of lines of the playing table would be detected).

2.2 Playing field borders to lines approximation

Firstly the Hough transform is applied to the image generated in the previous step with a set of parameters such that the number of lines obtained for each border is always greater or equal to one.

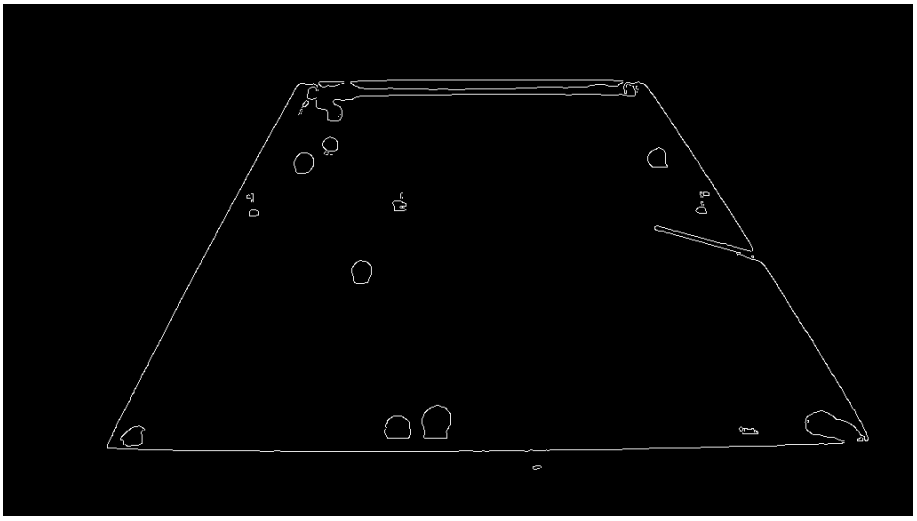


Figure 4: game2_clip2 edges

Then, K-means clustering is exploited to divide the lines into four classes, one class for each border. To apply the algorithm it is necessary to transform each line found in the previous step into a datapoint suitable for the task. Recalling that each couple (ρ, θ) given by the Hough transform describe the line

$$y = [-\cos(\theta)/\sin(\theta)]\, x + [\rho/\sin(\theta)],$$

each datapoint is defined as the vector $[\rho\cos(\theta), \rho\sin(\theta)]^T$. This kind of representation has been chosen after a process of trial and error in which different combinations of (ρ, θ) have been tested. This representation has been selected since the clustering obtained exploiting (ρ, θ) directly could end up classifying lines associated with different edges in the same class (as can be seen from Figure 5).

For what regards the approximation of each field border with a line, as a first trial, we tried to approximate it with the most distant line from the center of the frame for each cluster. Assuming that the center of the table is close to the center of the frame leads to approximate each field border with the most distant line from the center of the table in each cluster. This operation is necessary to select only one line per cluster and to filter the ones that are not given by the playing field border but by some shadows generated by the different planes where the field is defined (the inner lower area where the balls are present and the outer upper one). Unfortunately this approach has been discarded since the angular coefficient of the selected lines could differ too much from the right playing field borders coefficients.

The actual implemented function selects for each cluster a group of lines such that the distance from the center of the image is maximized and choose as (ρ, θ) coefficients the

mean between all the coefficients of the group. By this procedure a more robust approximation of the correct line should be found since it is chosen as the arithmetic mean between more than one result.

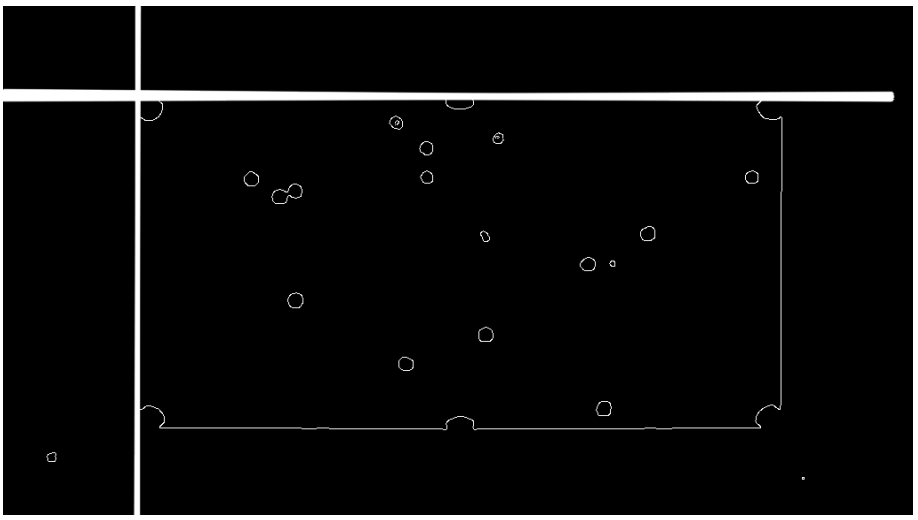


Figure 5: game1_clip1. Example of wrong clustering, the reported lines have been all classified in the same cluster

2.3 Playing field corners

The next step is to evaluate all the possible intersections between two lines. The intersections found can differs in number:

- four intersections if the lines are grouped in two sets of two parallel lines;
- five intersections if the lines are grouped in a set of two parallel lines and in a set of two non-parallel lines;
- six intersections if the lines are grouped in two sets of two non-parallel lines.

Every intersection is represented by the (x,y) coordinates of the intersection point (thus discarding the lines that generated the point).

To obtain the playing field corners, all the intersection points are analyzed and the first four that are found inside the frame boundaries are classified as corners. This step is robust only under the assumption that the lines extracted in the previous steps intersect in only four points inside the frame. This assumption implies that the playing field has to be completely inside the frame in order for the algorithm to work properly.

Finally the corners are sorted as represented in Figure 6. This step is useful both to correctly compute the projection matrix for the 2D top-view visualization map and to draw the playing field table segmentation mask. This step was developed by Marco with the sortCorners() function. In a first attempt the frame has been divided into four parts (upper left/right and lower left/right) and the corners have been directly classified making use only of their absolute position with respect to the image center:

- A: corner in the upper left quadrant;
- B: corner in the upper right quadrant;
- C: corner in the lower left quadrant;
- D: corner in the lower right quadrant.

At the beginning this seemed to be a reasonable solution but it does not work in the third video where the playing field is diagonal with respect to the camera point of view and both

the upper points fall in the upper right bin. As a consequence, the implemented function is slightly different and divides horizontally the image (two corners are in the upper part of the image and two in the lower part regardless of the orientation of the table) and then takes the one with lower coordinates value as the left one and the other as right one:

- A: corner in the upper part of the frame with lower x;
- B: corner in the upper part of the frame with higher x;
- C: corner in the lower part of the frame with lower x;
- D: corner in the lower part of the frame with higher x.

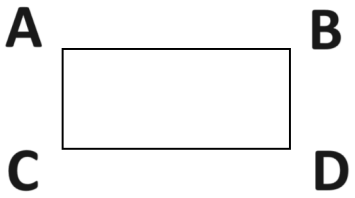


Figure 6: Playing field sorted corners

2.4 Mask generation

The playing field mask image is generated by exploiting the OpenCV drawing function drawContours(). All the pixels inside the shape ABDC are set to 1 and the other image pixels are set to 0.

2.5 Discarded approaches

Other attempts have been made applying K-means clustering to the image pixels. The aim of these approaches were to cluster the playing field pixels and the background pixels in two different sets. Different combinations of features have been tested: BGR values, HSV values, BGR values + pixel coordinates and HSV values + pixel coordinates. The pixel coordinates have been used to take into account not only the pixel color but also its position in the frame since all the playing field pixels are localized in the center of the image. These approaches have been discarded since in many cases they were not capable of correctly isolating the playing field from the frame background. This problem was particularly noticeable in videos in which the table brightness is not uniform and the playing field pixel values in the brightest and darkest areas were wrongly classified as background. Moreover there were also cases in which the background and the playing field have similar pixel colors and hence the algorithm completely failed merging the playing field with the background.

3 Warp matrix computation

To project a point in the playing field into a point in the minimap the warp matrix that links the two points coordinates has to be computed once the position of the minimap in the frame is fixed. As a first step, it is necessary to correctly link the corners of the playing field with the one of the minimap, and so to understand if the table is framed in a view from above or in a side view.

Riccardo suggested solving the problem with the tableOrientation() function that exploits the fact that when the playing field is not framed from a top-view there is a prospective effect such that the rectangle of the playing field becomes more like a trapezoid. By setting two thresholds on the difference respectively between the xs and ys corners coordinates it is possible to correctly understand if the table is horizontal or vertical/diagonal. At this point the wrap matrix is computed by calling the OpenCV function getPerspectiveTransform() with the four couples of points of the table and the minimap as parameters.

4 Balls localization

The goal of this step is the creation of a set of bounding boxes containing the balls on the playing field in the first and the last frame of the video. Each bounding box should also include a confidence measure on how much it is confident of containing a ball, a parameter that is necessary for the mAP computation. To accomplish this task, assuming the balls borders to be close to circles, Marco proposed to estimate the balls positions and radius exploiting the the HoughCircles() OpenCV function which also provides a confidence value for each detected circle that is equivalent to the number of votes that it has received in its identification.
A first attempt was made applying the transform directly to the Canny output of the BGR and the H channel of the HSV color space representation of the frames. These methods were able to find all the balls in the playing field but also led to a high number of false positives points, especially if the H channel was used. In order to get better performance, other approaches have been tested.
After different tries, the final implementation proposed by Riccardo exploits both BGR and HSV representations of the frame and can be divided into four step:

1. generate a mask on the H channel;
2. generate a mask on the BGR image;
3. merge H and BGR masks;
4. detect the circles in the mask obtained in (3) and generate the corresponding bounding boxes.

This approach led to better performances for what regards the discarding of false positive points related to the presence of gamer's hands and holes but led also to losing some ball in some clips (as can be seen from the bounding box localization images in section 10). After trying to tune as best as we could the parameters involved, considering the number of false positives against the number of false negatives and how these choices affect the final metrics, we decided to tune the algorithm in order to minimize the number of false positives paying the price of missing some balls.

4.1 Mask on H

In order to generate a mask for the balls based on the H channel of the HSV representation (and hence on their color) of the frame, the table mask is imposed to the frame, the filtered frame is converted from the BGR to the HSV color space and its H channel is isolated by splitting the obtained image.
Then the table color is evaluated calculating the histogram of the H channel (with 30 bins), saving the index of the bin with more votes i and evaluating its central value as

$c = 180/30i - 180/30/2$. The mask is generated by fixing to zero all the pixels of the H channel image with values outside the range $[c - \Delta_H, c + \Delta_H]$ with Δ_H = 10 and to 255 all its pixels with values inside the same range. The obtained mask has pixel value of 255 for the playing field pixels that are not occluded by balls and 0 value pixel for the balls ones. As a consequence, by taking the inverse of the mask a first ball segmentation is obtained. The mask obtained is then further processed by applying an erosion to avoid the table holes to be detected as balls. Since as outcome of this procedure the balls have not a circular shape, first a gaussian filter with an aperture size of 9 and a standard deviation of 3 and then a binary threshold are applied to the obtained image. By this process it is also possible to reduce the distortion introduced by the slicing. An example of the outcome of this process is shown in Figure 7.

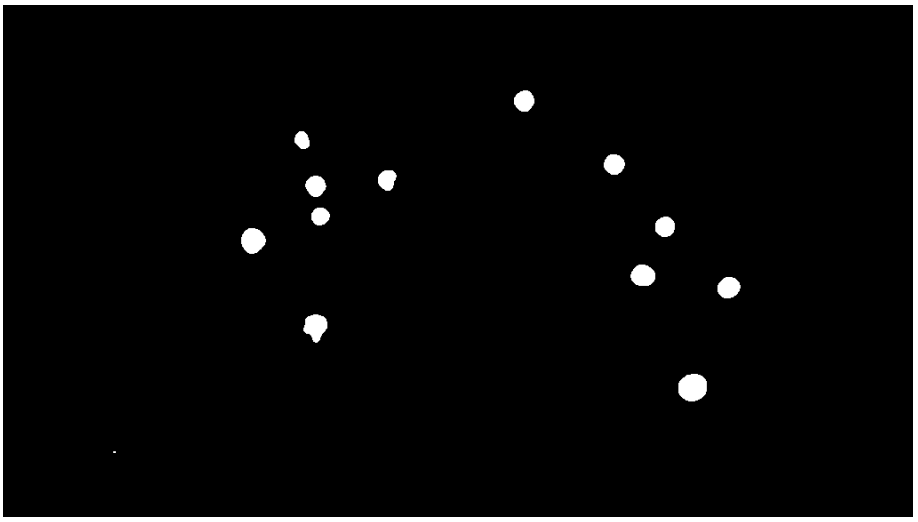


Figure 7: game2_clip2 MaskH

4.2 Mask on BGR

As previously done, the B + G - R channel combination is computed but only for the pixel inside the playing field. A binary threshold is applied to the combination in order to reduce the distortion and split the image in two classes.
As for the maskH process, an erosion, a gaussian filter and a threshold are applied to the result of the previous step. This procedure is made since, as can be seen from Figure 2, a lot of balls in this representation results dark and hence easily detectable.

Figure 8: game2_clip2 MaskBGR

4.3 Mask on H and BGR

The mask H and BGR are summed in order to merge the results given by the two approaches into a maskS (Figure 9). This is done since it is possible that a ball is only detected by one of the methods.
The contours of the generated mask are then found and the convex hull of each contour is generated. A mask is generated by the convex hulls found starting from the third bigger one in order to cancel out the hands and the cue stick hulls that are bigger than the balls ones.

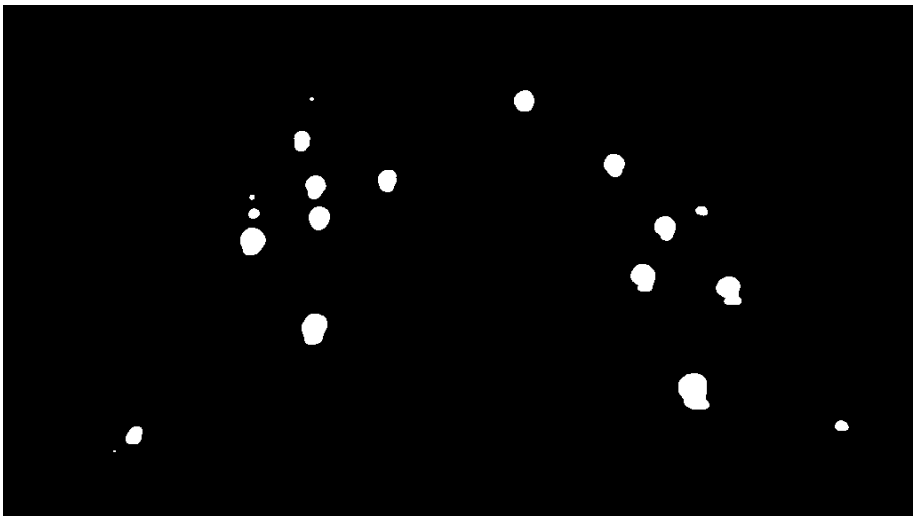


Figure 9: game2_clip2 maskS

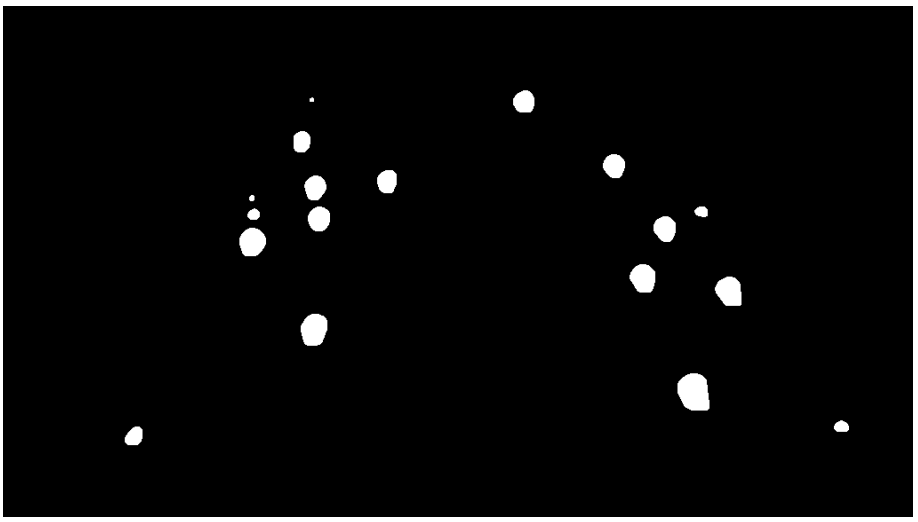


Figure 10: game2_clip2 Convex Hulls

4.4 Circles detection and bounding boxes

Once the Convex Hull mask is obtained, the HoughCircle() function is applied to that mask to get the circle positions. Starting from the (x,y) coordinates of each circle center, its radius and its number of votes, a bounding box is created choosing as (x,y) coordinates the center of the circle minus its radius (that correspond to the upper left corner of the bounding box) and as width and height twice the radius. The obtained bounding box is not as precise as we desired and this is due to the fact that the chosen threshold used on the previous masks can lead to balls that have a radius that can be greater or smaller than the real one. Even if the HoughCircle() function allows to choose a minimum and a maximum radius and the distance between two centers, in order to be able to detect balls close to each other where a perspective effect is present and given the different dimension of the balls in different clips (i.e. small balls in game 1_clip1 and bigger balls in game1_clip2), these parameters are not tight. If the center of the ball is not found precisely (i.e. when there is a strong brightness effect) and a too small radius is found, the bounding boxes selected are not so precise. Since the distance between the centers of two circles is chosen to be small, it may happen that if the analyzed ball is a striped one (or even balls with a shadow), two bounding boxes really close are found (as can be seen in the game1_clip2 first frame and game2_clip2 first frame).

5 Filtering holes with project points

The balls localization process still gives as false positives some holes (especially in the case of top-view of the playing field), some playing field points and background points given by the presence of a hand or a cue stick. In order to remove the false positives related to the holes and hence increase the score in the performance evaluation, a projectPointsFrame() and purgeTableHoles() functions have been inserted before proceeding with the tracking phase. The projectPointsFrame() function computes the projected points of all the balls found in the first frame exploiting the warp matrix and the then the purgeTableHoles() removes the projected points whose distance from the minimap holes (whose positions are assumed to be known) is above a certain threshold.

For what regards the false positives given by the presence of the hand or the cue stick no solution has been found to discard that bounding boxes (as can be seen, for example, in game1_clip2 and game1_clip3 results).

6 Ball tracking

Ball tracking has the main objective of tracking the ball path in order to draw its trajectory in the minimap but, as will be shown, can also be used to enhance the localization and classification performances on the last frame (it is possible to get better performance also in the first frame, as described in the section 11).
From a theoretical point of view, two strategies could be exploited to keep track of ball localization and classification (needed for the plot of the minimap): compute the balls localization and classification process for each frame or use a tracking technique. Both the possibilities have been taken into consideration.
Computing the balls localization and classification at each step seems at first sight a quite radical and expensive (from a computational point of view) approach but led to unexpected results in terms of time performances (indeed only few seconds were needed to process the whole video). The main drawback of this strategy is that, while the balls obviously have always the same shape (and hence the segmentation could possibly fail only when they are very close to the playing field boundaries), the false positive balls (i.e. the hands) change shapes and so they appear and disappear during the whole video. This phenomena also affects the classification because the implemented method for classifying the balls (as will be shown in the next section) is based on a value evaluated by analyzing the detected balls brightness. Another drawback of this strategy is that when the white ball moves on the playing field its brightness changes and hence both the localization and the classification can fail in some frames of the video (thus appearing and disappearing on the minimap). In addition, the classification can differ from frame to frame since the moving balls can change appearance during the video. For these reasons, after some tests, this approach has been discarded.
For what regards instead the use of a tracking algorithm, we decided to use a MultiTracker that is, according to the OpenCV documentation, a “naive” implementation of multiple object tracking algorithm that tracks objects independently without any optimization across the tracked objects. A tracker is created for each object and is added to each bounding box. In our case, the same kind of tracker has been used for tracking all the objects.
One important advantage that comes for free from the use of a tracking algorithm is that it is possible to keep the same label given by the classification of the first frame for the whole video. Clearly this can also be a disadvantage if the initial classification fails and also if a false positive bounding box is found in the first frame, since it will be kept also in the last one (see game1_clip3 last frame results).
The first choice to be taken regards which kind of tracker should be used. After trying with almost all the trackers available in the library (MOSSE, MedianFlow, KCF, Boosting CSRT, MIL, TLD) and evaluating their performances in terms of time needed and number of correct tracked ROIs over the total number of ROIs, the CSRT tracker results to be the best one for what regards the capacity of correctly and precisely tracking the balls in motion (especially the white one that in some video moves quite fast). Since we want the best performace (also because drawing a wrong trajectory in the minimap would not make any sense), the CSRT tracker should be used, which leads to facing the problem of how to make it work faster (in

fact it results almost 7-8 times slower than other trackers like MedianFlow or KCF that, on the other hand, were not able to correctly track the balls). Figure 11 shows the time needed to the tracking process in the case of game1_clip1 with a “standard” implementation, namely tracking all the objects given by the segmentation process that, in this case, are 18 elements[1].

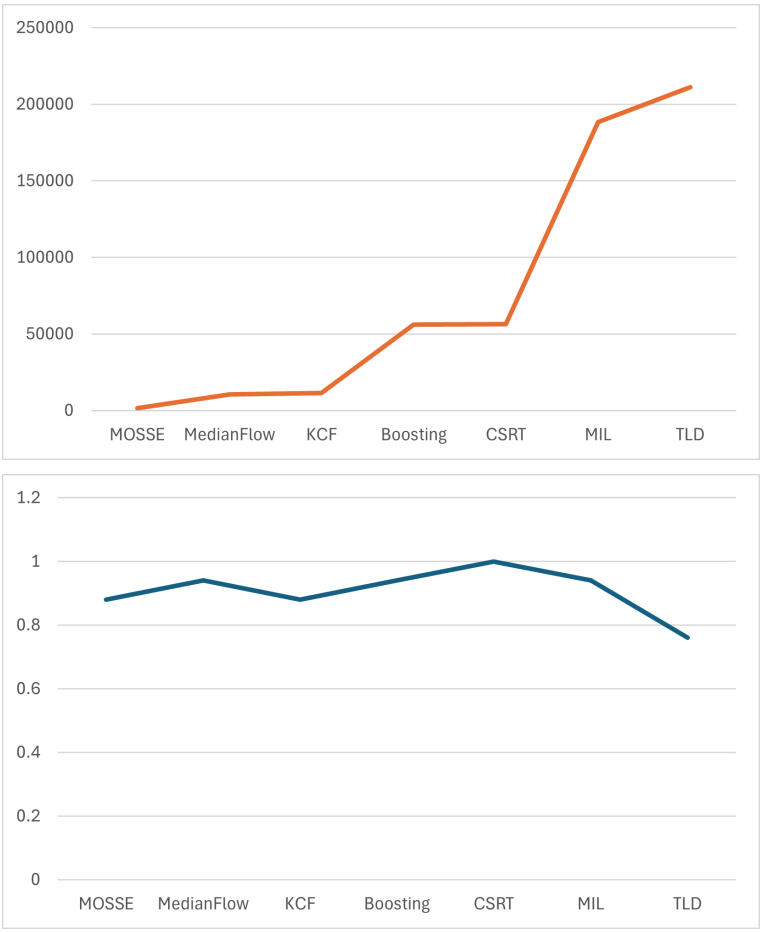


Figure 11: Time needed in [ms] (orange) and number of correct tracked ROIs over the total number of ROIs (blue) of different Trackers in Game 1 clip 1

Another possibility we tried was to find a way to enhance the performance of other faster trackers such as MedianFlow or KCF by adding some kind of pre-process of the video frame.
Riccardo tried to improve MedianFlow and KCF performance by trying to isolate the balls in motion from the balls that remained still and the playing field. This has been achieved by removing (i.e. putting its BGR value to (0,0,0)) the playing field with a double threshold on its mean color and all the balls found from the segmentation of the last frame from the field. This strategy does not seem so robust but has been sufficient for us to understand that, since only slightly better performance has been achieved on a subset of video, it was not a good strategy.
A first attempt in the direction of improving CSRT performance has been tried by Marco and consisted in trying to analyze only a subset of the video frames, thus significantly increasing the speed of the algorithm. Unfortunately this approach does not lead to good results since

[1] Performances evaluated on DELL laptop i7-11800H, 16 Gb ram, NVIDIA GeForce RTX 3050 Laptop

the tracker loses the balls position from a frame to the next one and hence becomes ineffective.
A solution that really improved CSRT performances has been found, after different tries, reducing the number of tracked ROIs. A first bounding box selection has already been made by the purgeTableHoles() function in the previous step but, while this strategy seems to work well for what regards hole detection, it still left a too high number of ROIs (ideally all the balls in the playing field). A solution has been found by distinguishing ROIs that move from the one that remains still and track only them. From a practical point of view, this has been achieved by removing all the ROIs whose center position is close (above a certain pixel distance) to the same ROI in the final frame obtained by the same segmentation process (this is made by the motion() function). This procedure really makes the difference since it significantly reduces the number of ROIs from more than 10 to 2-3 (or maximum 5 in case of false positive detection due to the presence of a hand in the first frame). As a consequence we have been able to reduce the time needed by the CSRT from 60-80 s to at most 10-15 s. The outcome of the motion() function is stored in a variable itMoves that would also be exploited in the plotting of the minimap phase.
While discussing about the kind of tracker to use, another choice should have been taken about the dimension of the bounding boxes: in fact a small bounding box usually leads to a more precise ball localization during the tracking but, in order to track the white ball correctly, a larger dimension of the bounding box needs to be chosen due to its relative high speed of motion. After several attempts, the final choice for the bounding box dimension has been twice the box dimension. If we focus on the CSRT algorithm (even though similar behavior has been found also with MedianFlow and KCF), by choosing a dimension of 2.5 times the box dimensions, the tracking process tents to become too imprecise and, if two balls collides, there have been situations in which the ball that goes into the hole is lost (Figure 12) or both the tracker of the two balls keeps linked to only one of the two ball in the next frames (Figure 13).

Figure 12: game3_clip1 with ROIs dimension 2.5 times the original bounding box (image above) and 2 times (image below). The tracking of the ball that falls in the hole is lost with higher ROIs dimension and leave false positive on the playing field (highlighted in red)

Figure 13: game3_clip1 with ROIs dimension 2.5 times the original bounding box before the collision (image above) and after (image below). The two trackers merges together into only one ball (highlighted in red)

The output of the tracking process is a dataset in which is stored the position at each frame for the balls in motion while always the same position (of the first frame) for the ones that stand still. The final step of this phase has been to store the projection of each point for each frame that would be necessary to plot the 2D top-view visualization map.

7 Balls classification

The aim of ball classification is to correctly classify each ball detected by the previous steps as “cue ball”, “8-ball”, “solid ball” or as “striped ball”. Each of these classes will be encoded by a label id: white ball (1), 8-ball (2), solid ball (3) and striped ball (4). In practice, the result of this step is a vector of integers in which each index contains the label associated to the bounding box at the same index in the bounding box set.
The first criteria taken into account for classifying the white ball has been made by Marco and was based on counting the number of white pixels in BGR domain (where a pixel is considered white if each BGR channel value is over a certain threshold) for each ball and label as white the one with the high number of occurrences. Even though this method led to decent results (the white ball was correctly classified on 5 up to 10 videos) the chosen one leads to better performances. Another approach is to analyze the first frame in the HSV domain, in which the “perfect” white color should be (0,0,255) and label as white ball the one with the lowest mean values for the first two channels and the highest one for the third. In general both these approaches suffers from the fact that the white ball in the video often is not as bright as expected and hence the “white pixel threshold” should be quite low (150-160), thus leading to a lot of misclassifications due to other stripes balls or also solid balls with a strong lighting effect that could result to have an higher number of white pixel than the correct one. A similar approach has also been considered to classify solid and striped balls: after having set a threshold on the number of white pixels, the label “striped ball” was given to the balls over that threshold and the label “solid ball” to the ones above that threshold. Again, the lighting and the circle around the ball number make a lot of solid balls (sometimes even the black one) to be wrongly classified as a striped one.
The chosen implementation was proposed by Riccardo and it consists of two steps:

1. Extract the patch contained in each bounding box and, for each patch, evaluate a (scalar) metric that describes how bright the patch is. Classify all the balls as “solid ball” or “striped ball” exploiting the brightness metric;
2. Classify, regardless of the previous labeling, one ball as “8-ball” and one ball as “cue ball” by overwriting the classification obtained in the previous step.

7.1 Classification of solid and striped balls

To begin, each patch is extracted from the image and converted in the Lab color space. Then, the histogram of the L channel (with 64 bins) of each patch is calculated and the brightness metric is evaluated. Since the objective is to compare the different brightness of each patch, a value should be chosen. A first attempt has been made by encoding the values assumed by the histogram into one single value exploiting the following formula:

$$\sum i[h(i) > 0]$$

in which h is the histogram and i is the index of the histogram bins. This choice has been made to consider only the non-zero bins indices (thus discarding the bins values). This technique has then been discarded since many solid and striped balls have histograms with a large number of non-zero bins in the same positions, and so this method was not able to discriminate between the two classes. In the final implementation the formula used to encode the histogram values is:

$$\sum ih(i)^2,\ i > v$$

With respect to the previous formula, this one gives more weight to the brighter patches and considers only the brightness levels over a certain value v (that we fixed to the number of bins divided by 1.75 after having observed that the histogram of the “striped balls” usually assumes high values over this threshold while the solid ones have low values).
To conclude, the mean of the metric of every patch is evaluated and each ball is classified as a solid one if its brightness metric is below the mean and as a striped one otherwise.

7.2 8-ball and cue ball classification

Once the patches are converted in grayscale, the “8-ball” is classified as the one whose patch contains the largest number of pixels under a certain threshold and as the “cue ball” is classified as the one whose patch contains the largest number of pixels over a certain threshold, similarly to what Marco proposed. It is interesting to notice that this classification method for the “8-ball” and the “cue ball” works better than the original one that relies on the whole BGR color space exploiting less information (only the grayscale representation). In fact in almost all the cases the classification of these two balls results correct.

The overall method instead is not as effective as desired for what regards the solid/striped classification: for example in the game2_clip1 (first frame) at the end of the classification phase all the balls found are classified as solid, thus implying that the “cue ball” was firstly the only one classified as “striped ball” since it has a very high brightness level with respect to the striped ones.

8 2D top-view visualization map

The plot of the point with the warp matrix was not exactly what was expected: even with the reasonable assumption that the camera does not move during the video, the coordinates of the bounding boxes (and so also of the projected point) change by a few pixels, thus making the balls projected shaking and this happen not only for the balls that really move during the video but also for the ones that stand still. This phenomena is due to the tracking process that slightly changes the coordinates of the bounding boxes in each frame. In order to reduce this effect, Marco exploited the output of the motion() function defined inside the tracking phase to split the balls that change position from the one that stands still. By doing this, if a ball belongs to the ones that move it is actually plotted in the minimap at each frame with its current position (and the path is drawn) while, if a ball does not move, the position of the first frame is plotted in each video frame. This procedure leads in general to good results, even though it clearly is not able to remove the “vibrations” of those balls that will move in the following frame of the video. The trajectory drawing has been made by exploiting the knowledge of each projected point in each video frame and plotting a line connecting one every two points. This procedure is possible exploiting Riccardo’s projectPoints() function that, by storing the ball position along the whole video, allows it to go “backward in time” and plot lines between ball centers at different instants. The choice of not plotting each center but skipping one is done in order to reduce the shaking effect of the ball trajectory (if more points were skipped, when a ball hurts the edge of the playing field and changes direction a misleading superposition of trajectories would appear in the final image Figure 14).
In the minimap the following color code is used: white color for the “cue ball”, black color for the “8-ball”, blue color for the “solid balls” and red color for the “striped balls”.

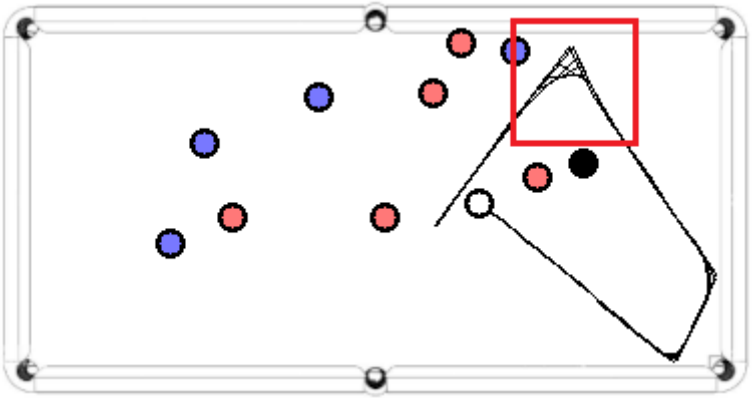


Figure 14: game4_clip2 trajectory plotting one each six points

The outcome of this phase is not anyhow as close as we thought to the image described in the project assignment in which the ball trajectory was described by a straight line. Another important difference is that the line that we plotted started from the ball center and hence the lines describing the trajectory of the balls are not connected. This is a difference from the image in the assignment that we considered acceptable since the centers of the balls actually do not collide and hence the trajectory of their centers do not match in any position.
What could happen in the case of a ball that goes inside a hole is that the tracker remains stuck in the neighborhood and is still plotted in the minimap. In order to remove them from the plot, Riccardo defines a function becomesATableHole() that checks if the center of the

ball goes close to a hole or outside the boundary of the playing field during the video. As a consequence, the ball is not plotted in the minimap anymore. The results obtained exploiting this function are quite good in general (almost all the balls are correctly removed) except for the game1_clip4. This is because the final bounding box of the ball in game1_clip4 (that instead does not go into the hole) has approximately the same position of the bounding box of the ball in game2_clip2 and hence the choice was between getting the right result in the first video or the second one (Figures 15-16).

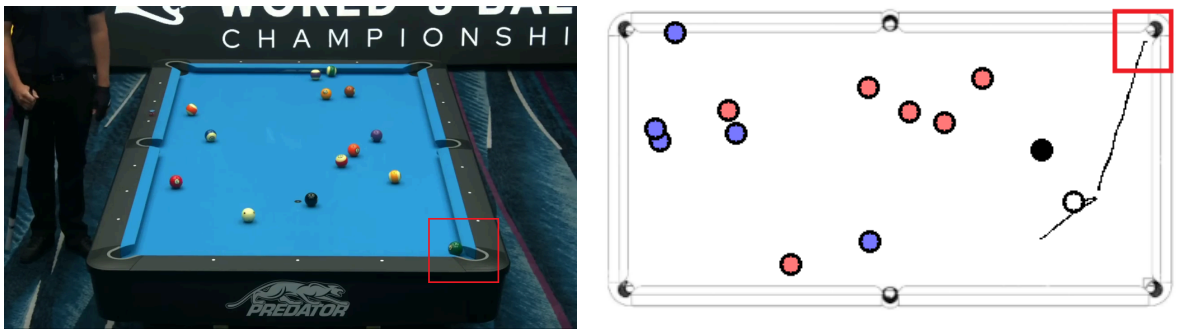


Figure 15: game1_clip4, ball wrongly discarded

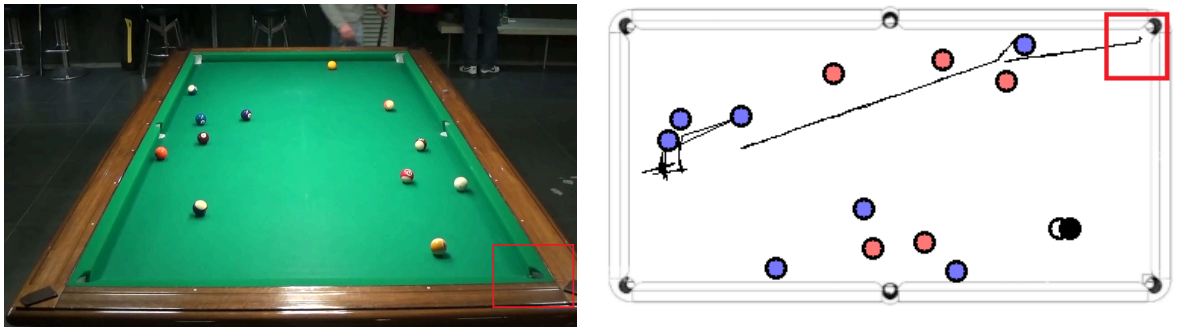


Figure 16: game2_clip2, ball correctly removed

It is important to notice that in any case the position of the center of the bounding box used to project the point into the minimap, in the case of playing fields with a strong perspective effect, should not be chosen as the point to be projected. The right one should be the point of contact between the ball and the playing field. This difference may lead to a lower precision in the point representation in the minimap.

9 Performance measurement

The performance measurements are taken exploiting the two metrics described in the assignment. The choices to make regard the dimension of the ball radius and if, for the last frame, exploiting the extemporaneous localization/classification or the results given by the tracking phase. In the case of the radius of the balls, what comes out of the localization are often circles with an approximately correct radius of the ball, but there are cases in which very small circles are found with respect to the actual dimension of the ball. This leads to a wrong or misleading performance evaluation in terms of IoU and hence low values of both mAP and mIoU. Marco tried to overcome this problem by considering the same radius for

each ball, chosen as the median value of the radius found by HoughCircles(). The choice of taking the median instead of the mean was to make the process robust to outliers (i.e. very small or big radius that could be detected, even though the initial tuning on the min-max dimension of the circle in the Hough function should partially prevent this problem). The main drawback of this choice is that it works well when there is not a strong perspective effect (i.e. game1_clip1) while in the other cases it produces bad results. In the end the value chosen is the one suggested by Riccardo, that is to keep the original radius found for each ball.
The confidence sorting process in the mAP is based on the confidence values taken directly from the HoughCircle() function output.
For what regards instead the decision between exploiting the tracking outcome or use the extemporary ball localization/classification to compute the metric on the final frame, we choose to proceed with the first option. A reason for exploiting the knowledge given by the tracking instead of by segmenting and classifying the last frame is that, especially when a strong perspective effect is present, the final position of two balls can be very close and hence the localization process finds only one ball instead of two (Figure 17). In this context the results given by the tracking are more robust. As said before, one important drawback is that a wrong initial localization/classification compromises also the performances of the last frame. On the contrary, by computing again the ball localization/classification for the last frame there may be the possibility to get rid of the false positive bounding boxes given by the hand and the cue stick that were found in the first frame.

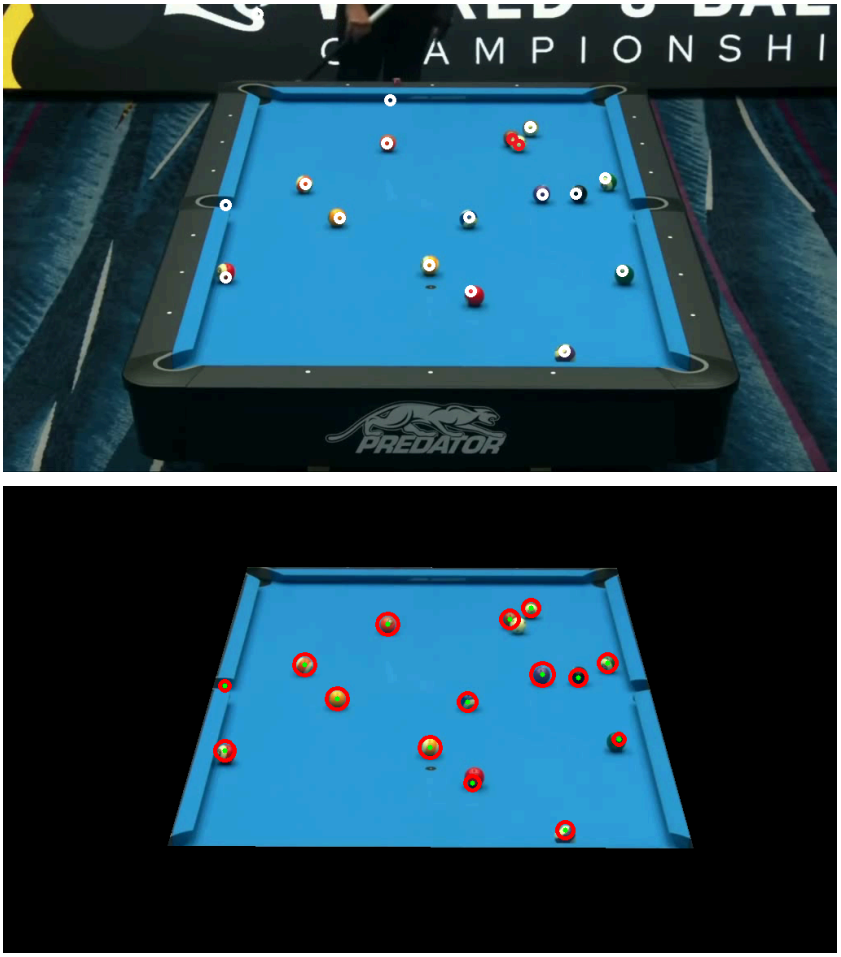


Figure 17: game1_clip2. Centers obtained with tracking (upper image) and with HoughCircles() (lower image). Computing the localization/classification again would make the white ball disappear.

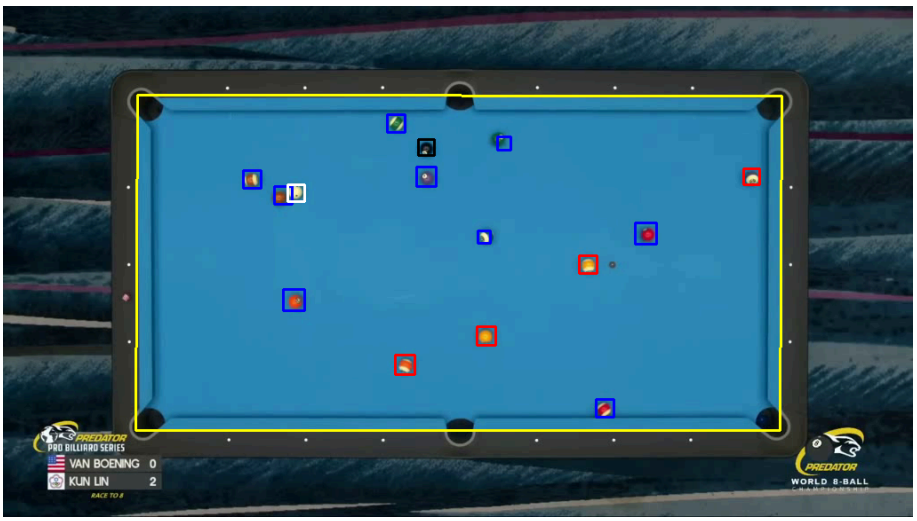
The datasets used to compute the mAP and the mIoU for the first frame is the balls localization/classification output filtered by purgeTableHoles(), while the dataset of the last frame, since the outcome of the tracking has been chosen to compute the last frame performances, is the dataset with the updated coordinates of the balls that move filtered with purgeBecomesATableHole(). This function works similarly to the BecomesATableHole() function, thus removing from the dataset the points whose projection goes in the holes neighborhood or outside the playing field.
For what regards the ground truth frames with which the metrics are computed, the frames used are the one taken directly from the video.
For what concerns the mIoU metrics, another choice to be taken regards the background/playing field segmentation method. A first try has been done by obtaining the background segmentation by subtracting to a white image the masks of all the balls and the playing field, where the playing field has been segmented by filtering each pixel of the original frame by keeping all the point in a range defined by the mean color of the playing field computed before. This approach does not lead to good performances since the segmentation exploiting the mean color of the table fails along the edges of the plane in which the balls move and hence a lot of pixels that actually belong to the playing field are wrongly classified as background. On the other hand, the hand is correctly classified as background. Riccardo's approach is more coarse and is based on the initial playing field segmentation mask. The background mask is obtained by subtracting the playing field mask to a white image (Figure 18) and the playing field mask is the one obtained by the segmentation process minus the balls segmentation masks. Looking at the final metrics, Riccardo's method has been chosen as the best one because it gives a higher intersection over union ratio. In both the solutions described, the balls masks have been generated exploiting the centers, the radius and the ids obtained with the localization/classification process described above. Each mask is generated by fixing to 255 all the pixels belonging to the circles described by the above data for each label id.

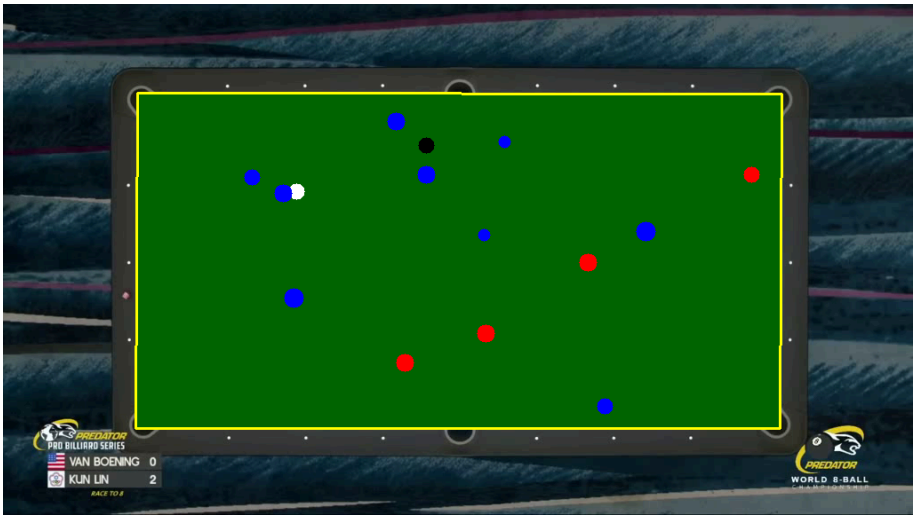
Figure 18: game1_clip1 (left) and gam1_clip2 (right), comparison between ground truth and Marco’s approach (in the middle) to background segmentation and Riccardo’s one (the lower one)

10 Results

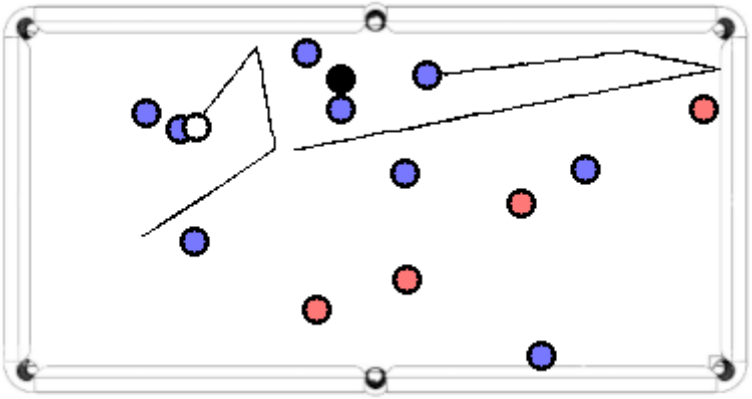
Game 1 clip 1

mAP First: 0.71875	mIoU First: 0.683544

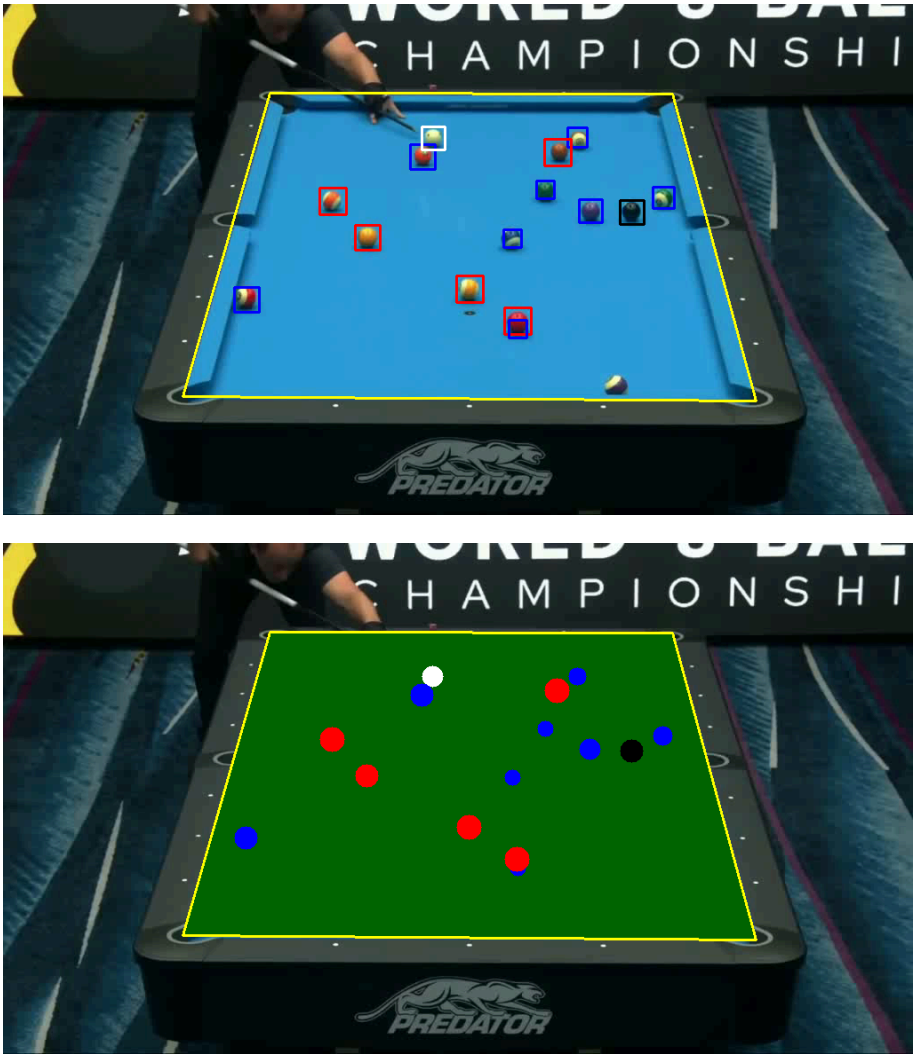




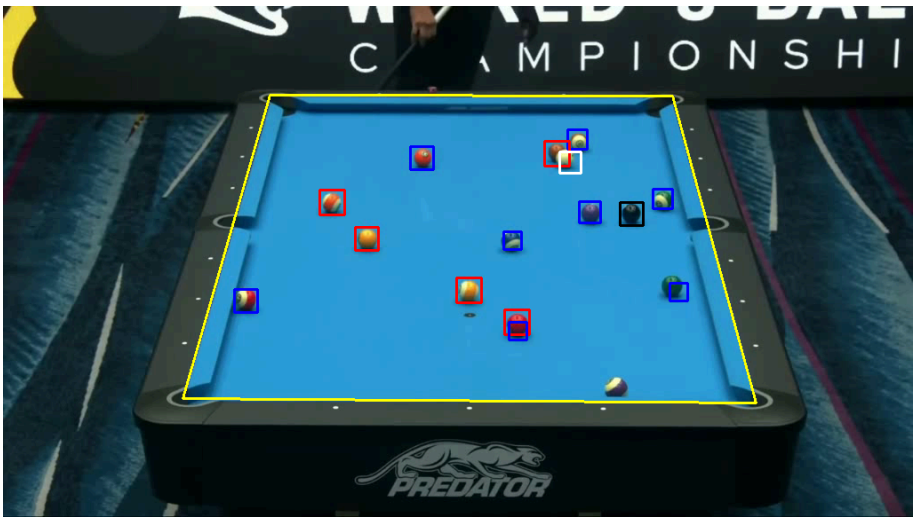
mAP Last: 0.650162	mIoU Last: 0.667932

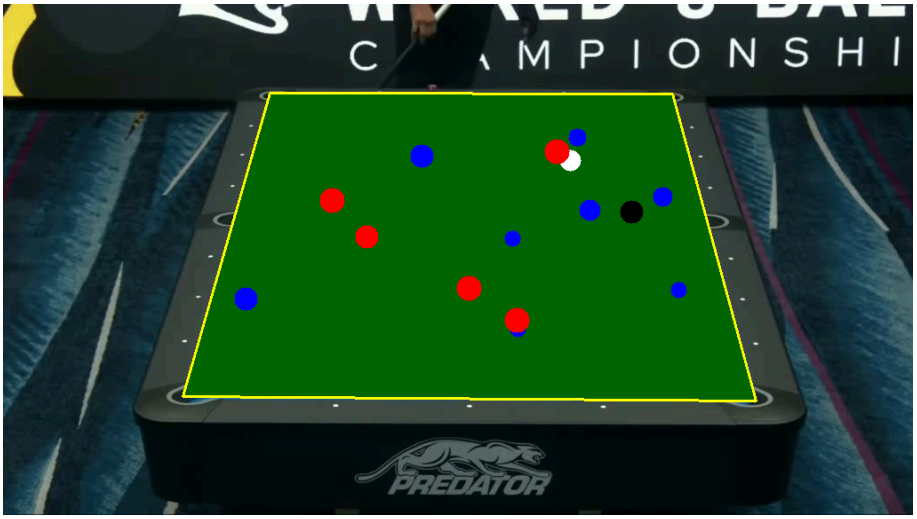


Game 1 clip 2

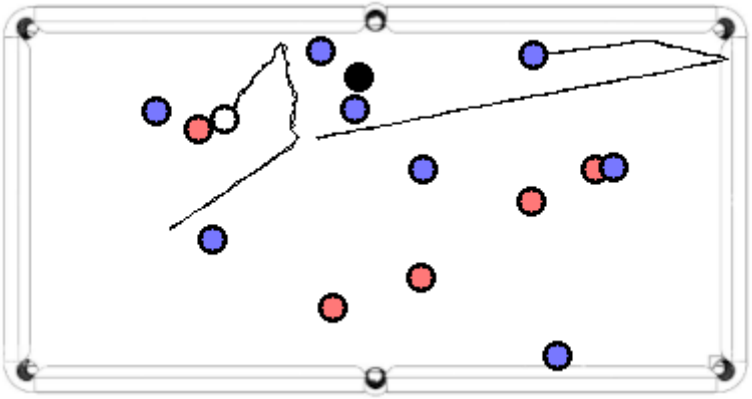


mAP First: 0.545455	mIoU First: 0.647716

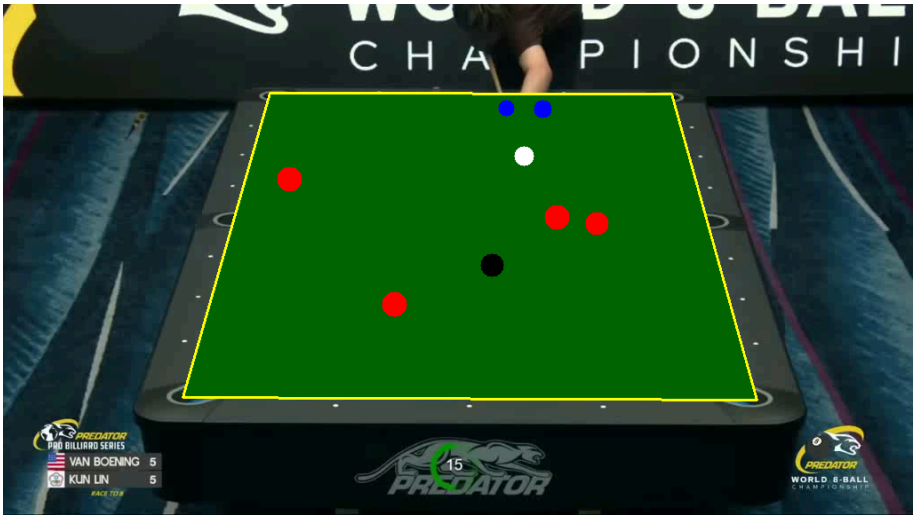




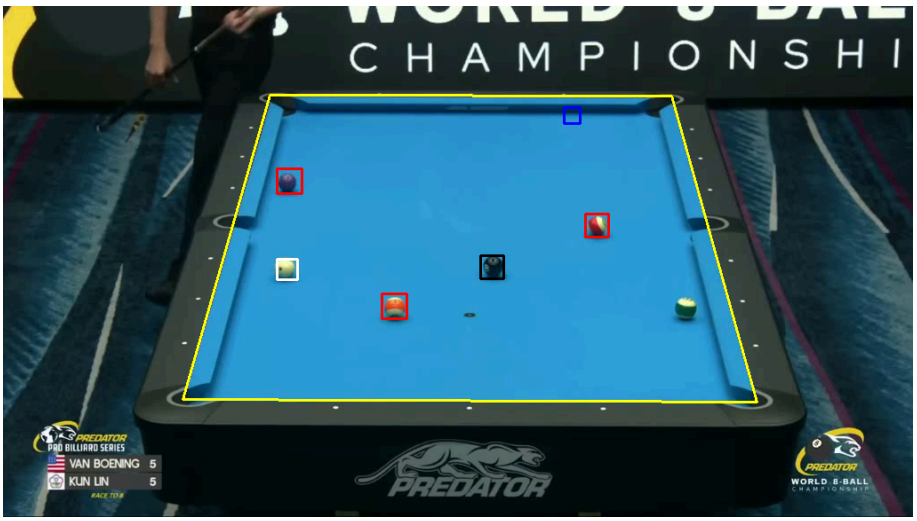
mAP Last: 0.295455	mIoU Last: 0.595799

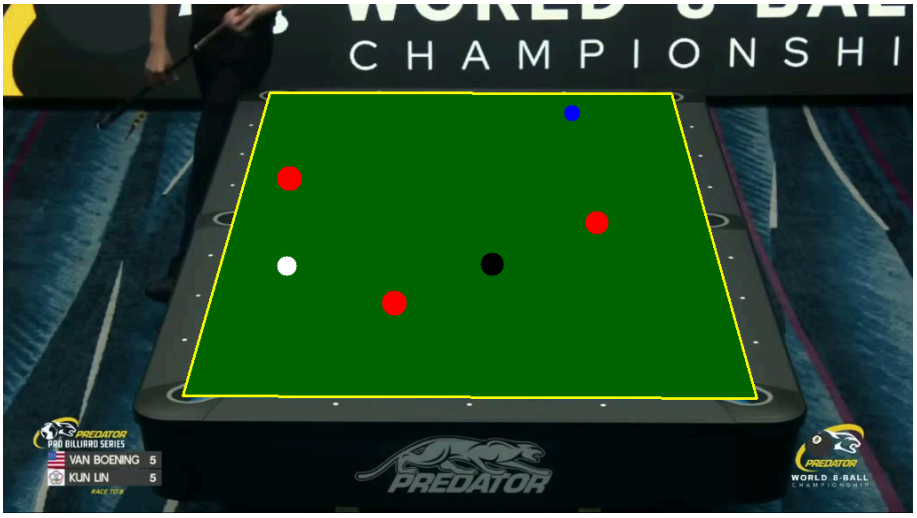


Game 1 clip 3

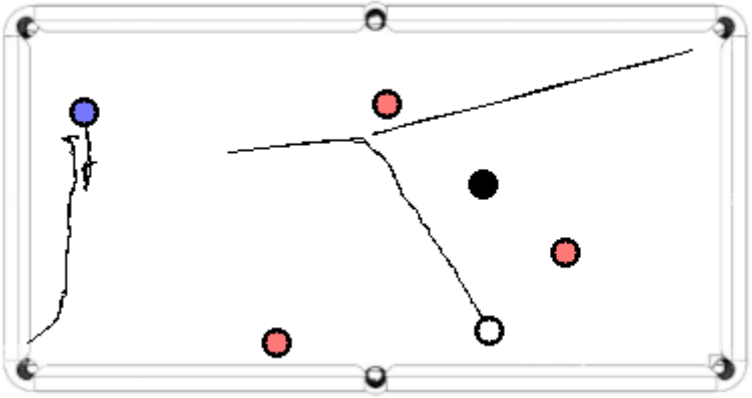


mAP First: 0.55303	mIoU First: 0.624285

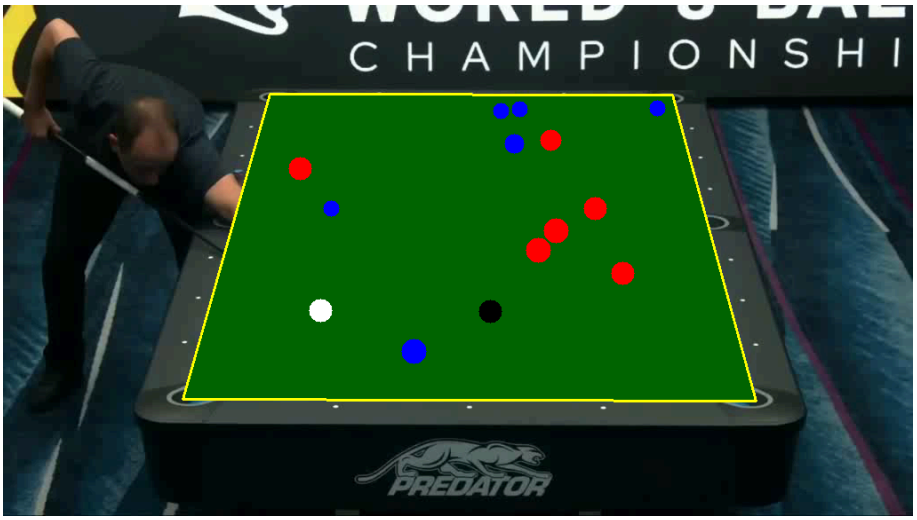




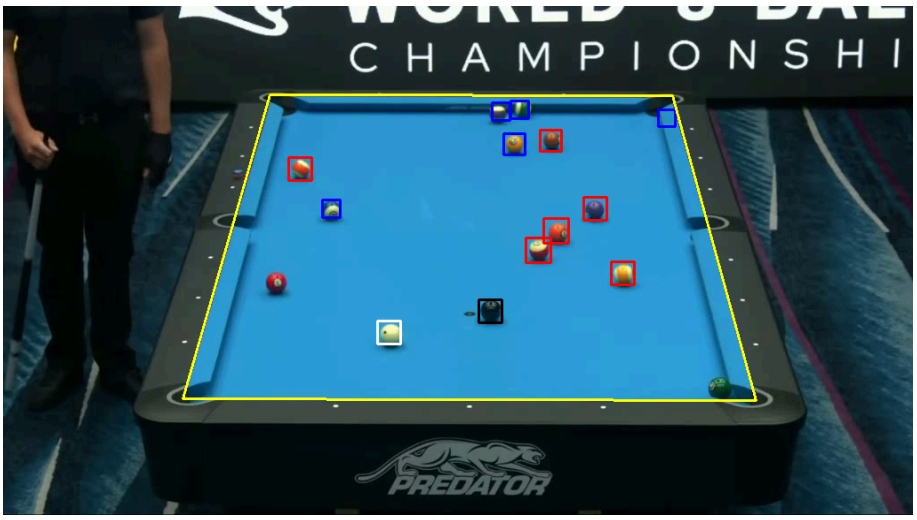
mAP Last: 0.613636	mIoU Last: 0.655687

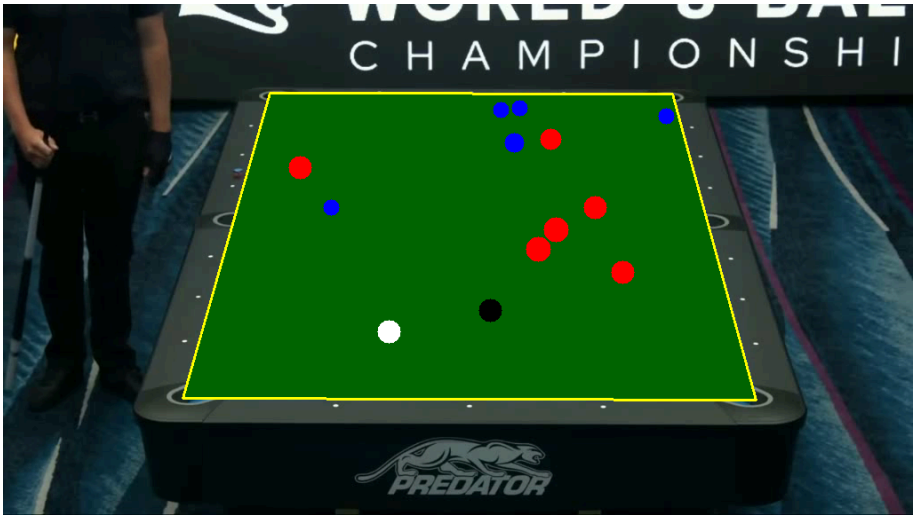


Game 1 clip 4

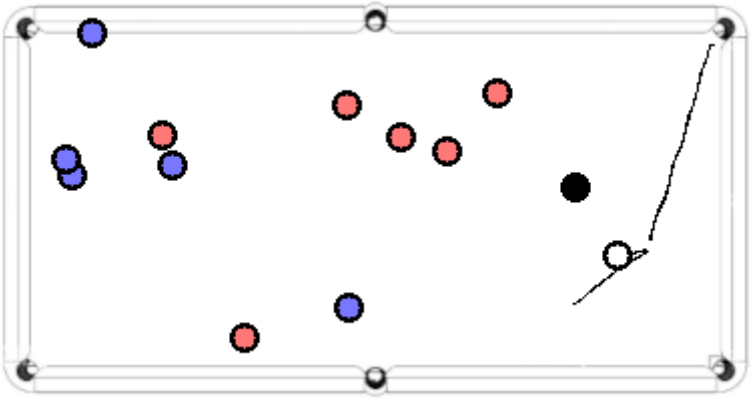


mAP First: 0.659091	mIoU First: 0.674964

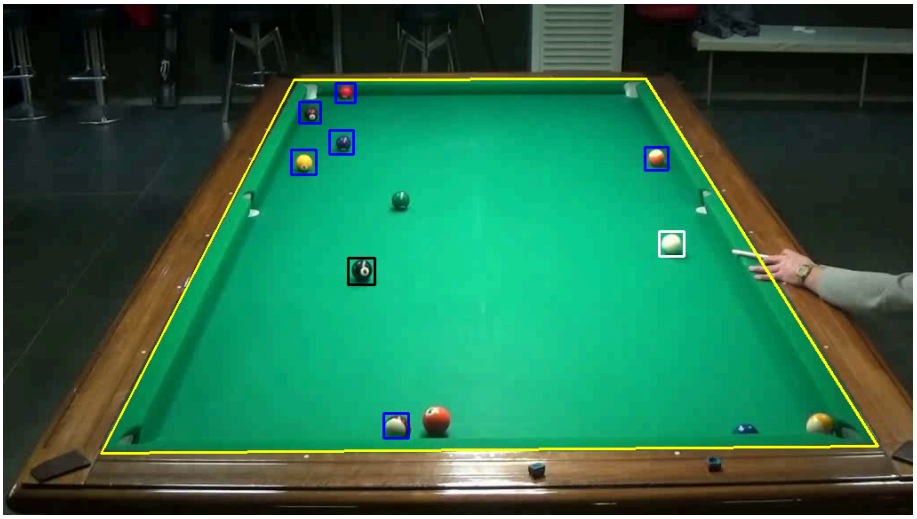


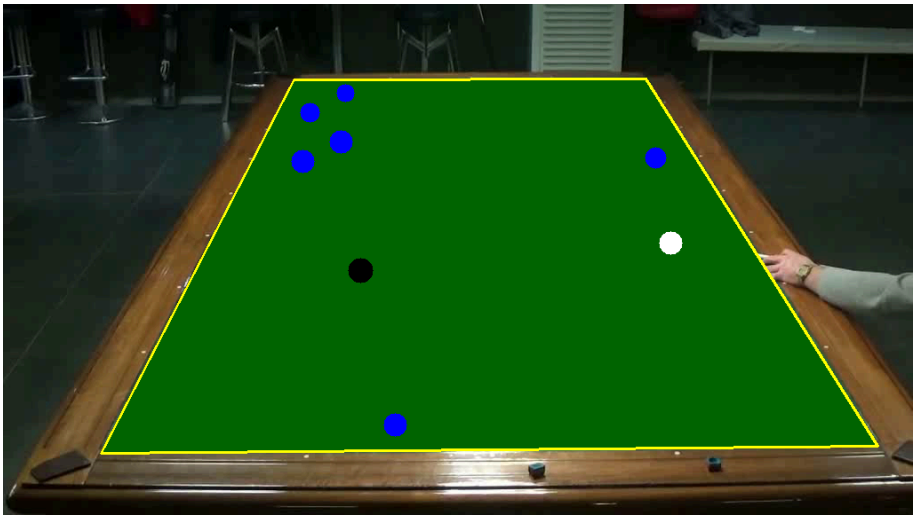


mAP Last: 0.613636	mIoU Last: 0.644024

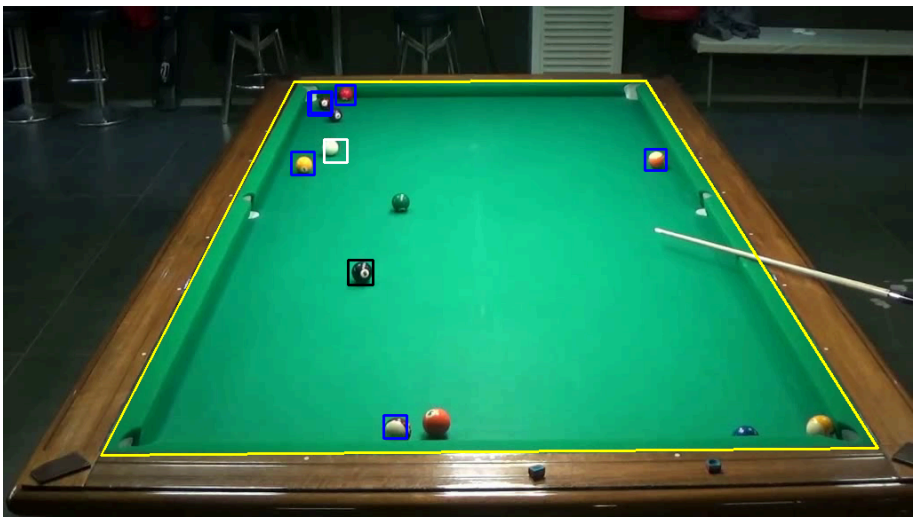


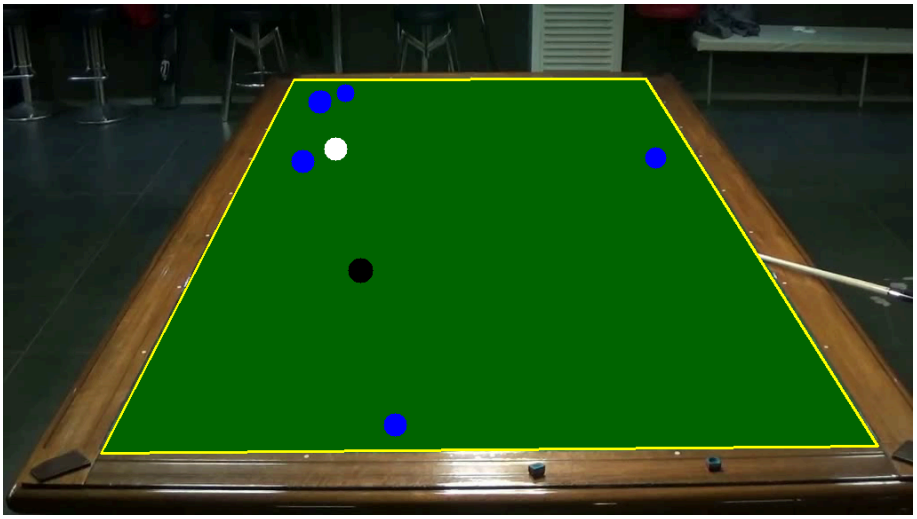
Game 2 clip 1



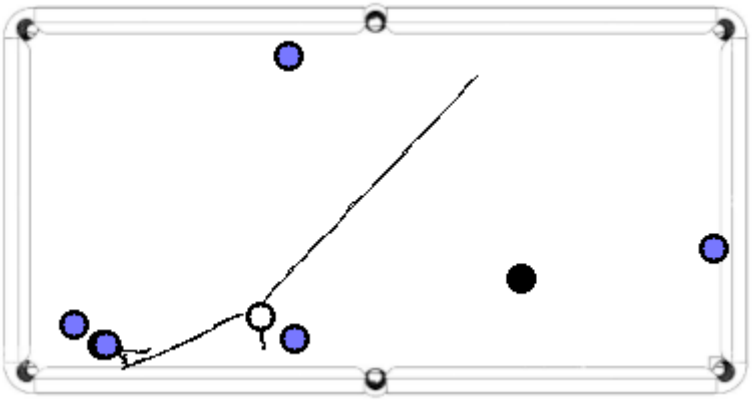


mAP First: 0.492424	mIoU First: 0.637355

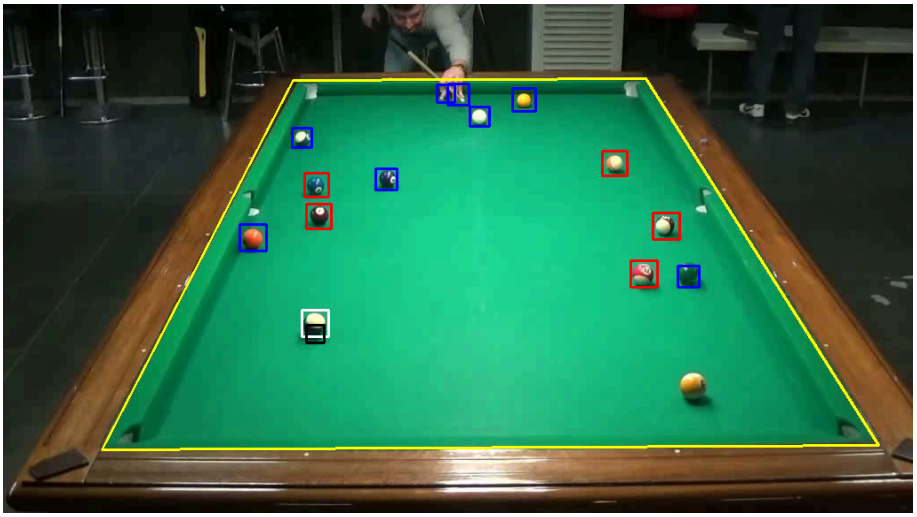


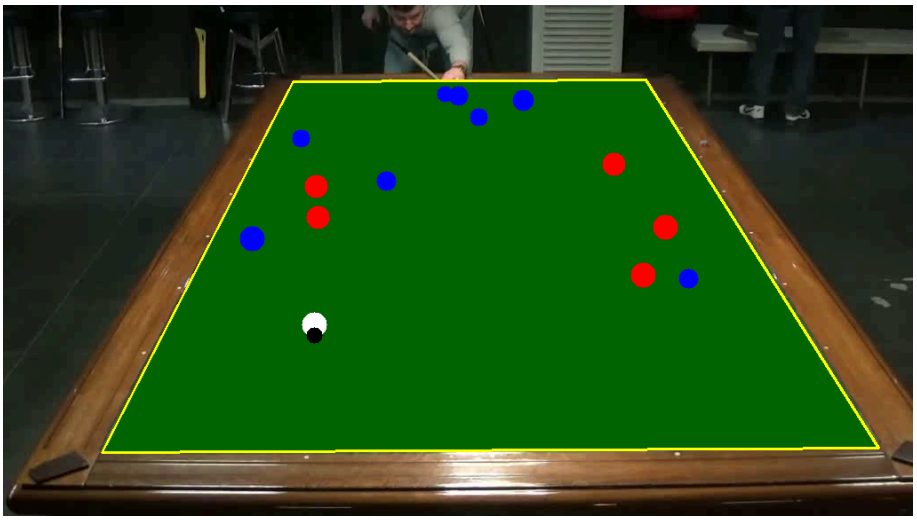


mAP Last: 0.265152	mIoU Last: 0.556324

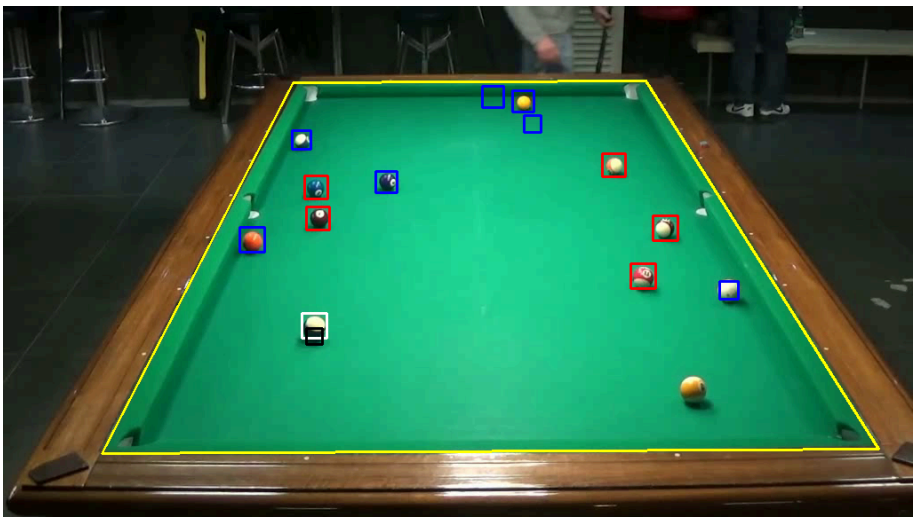


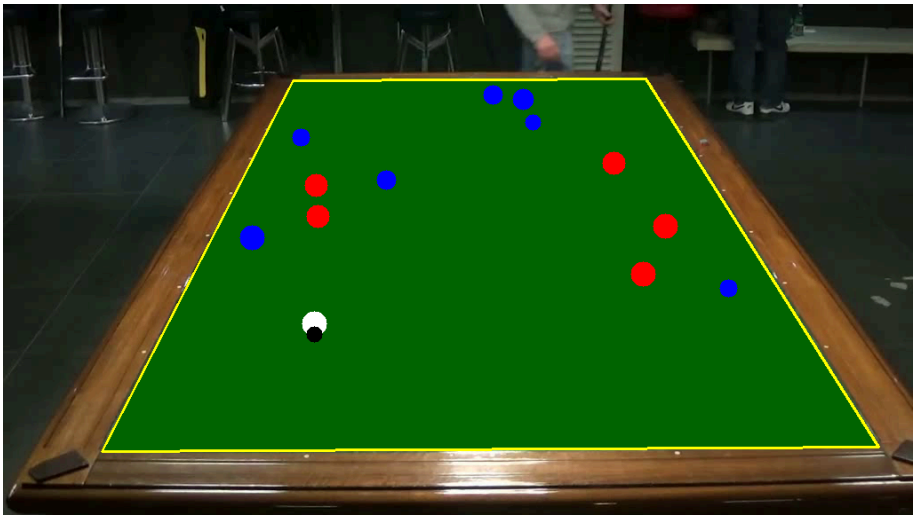
Game 2 clip 2



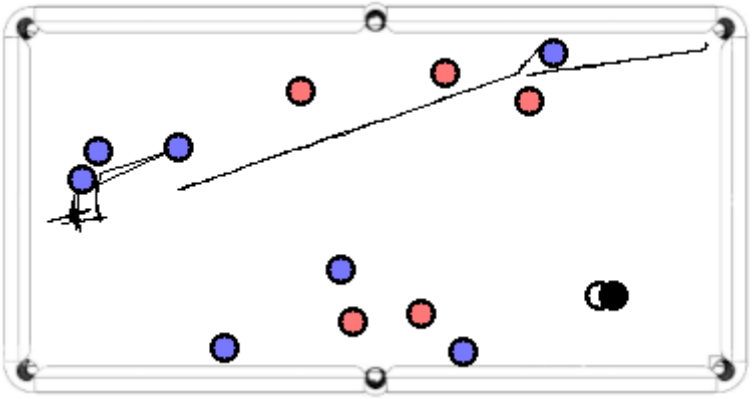


mAP First: 0.155844	mIoU First: 0.439732

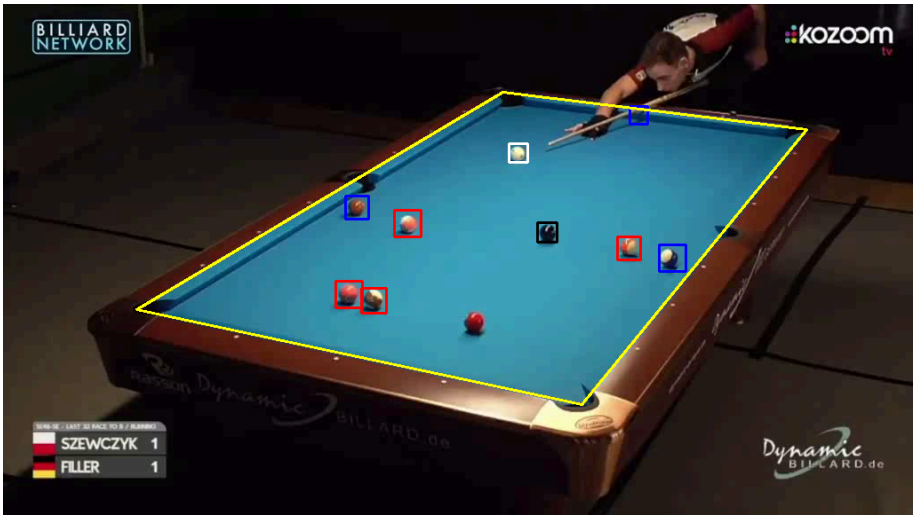


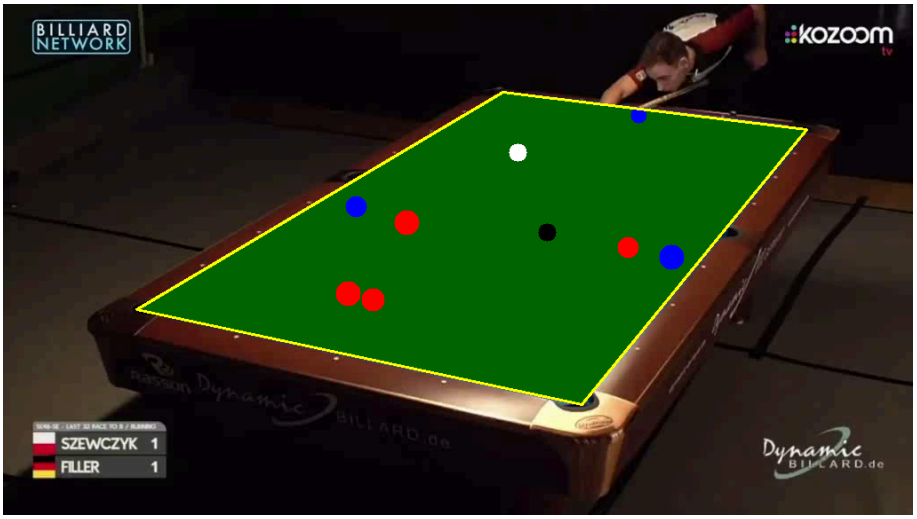


mAP Last: 0.159091	mIoU Last: 0.408284

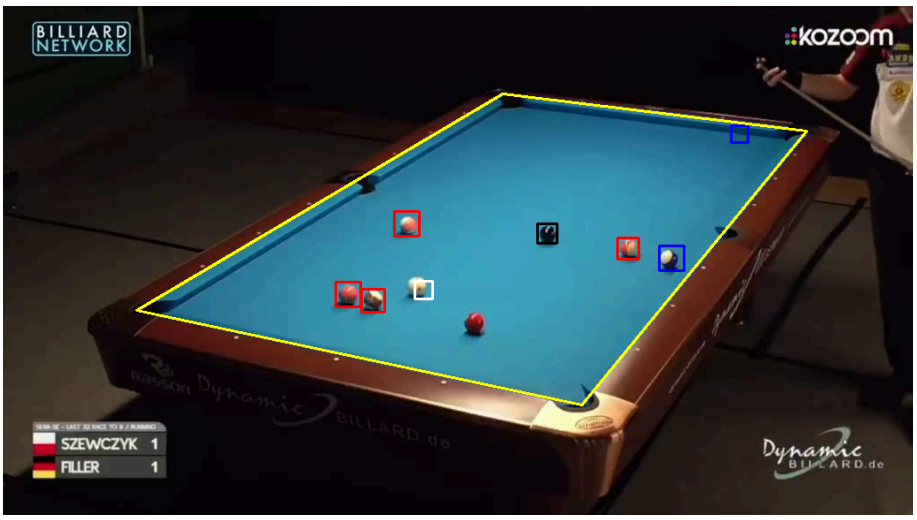


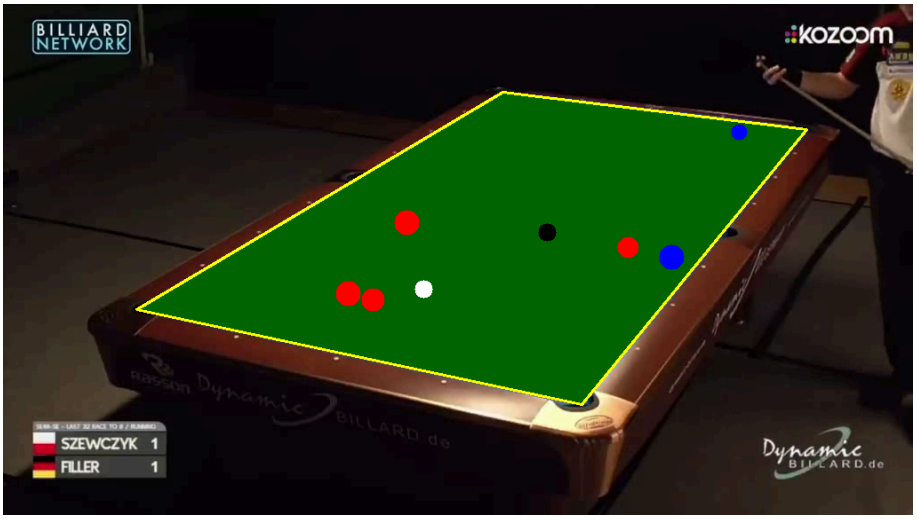
Game 3 clip 1



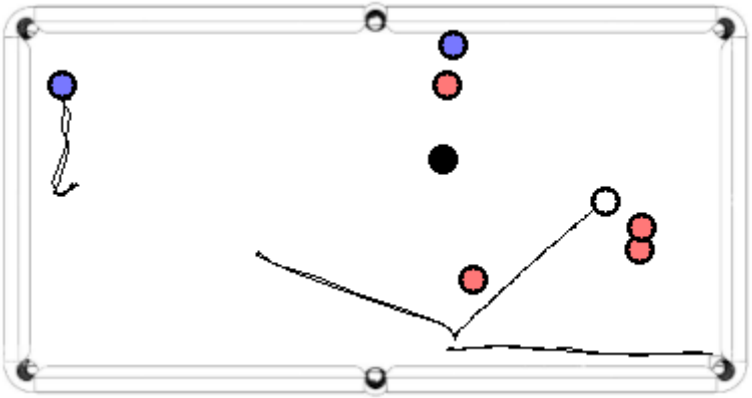


mAP First: 0.681818	mIoU First: 0.680639

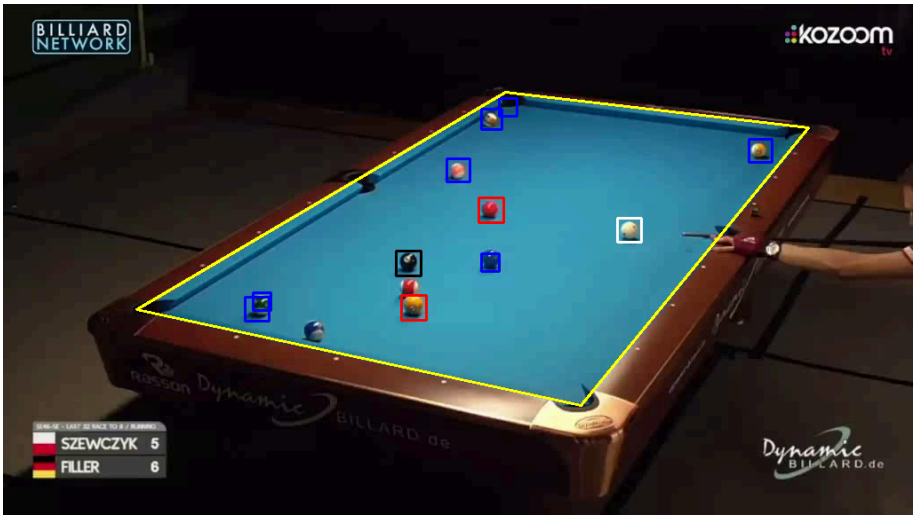


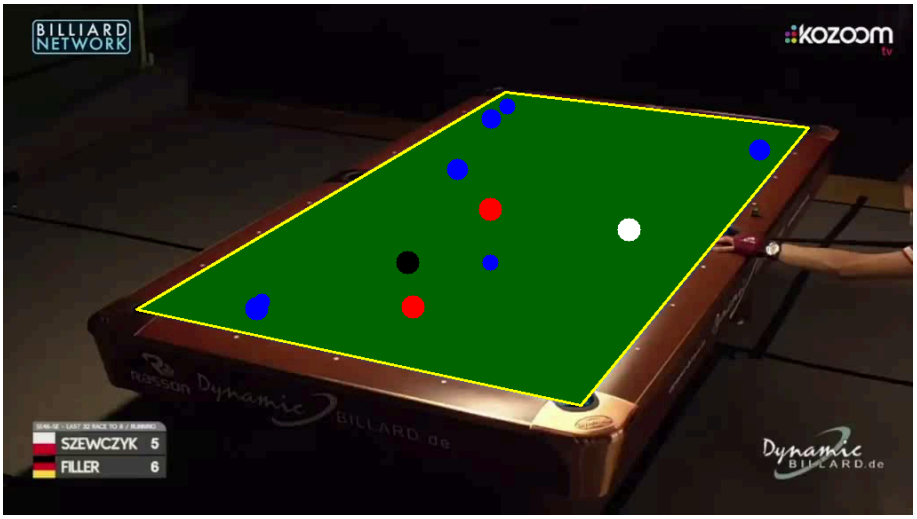


mAP Last: 0.363636	mIoU Last: 0.594775

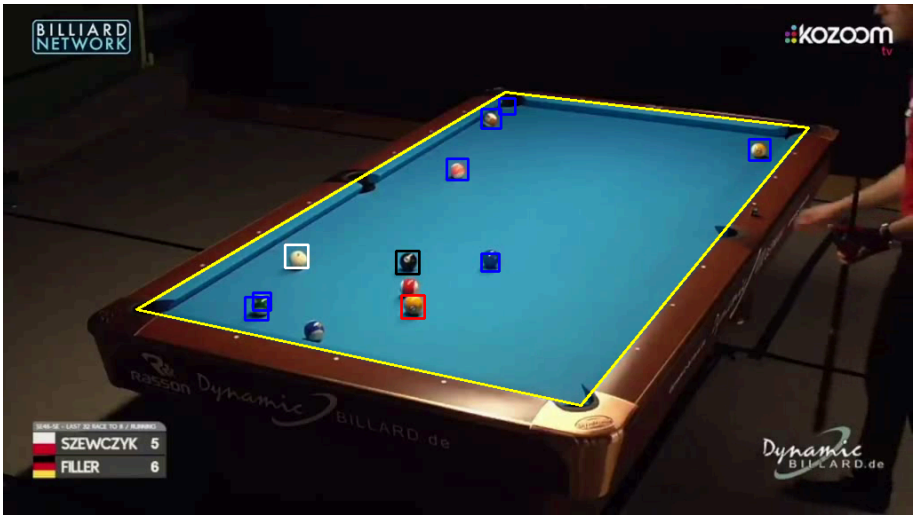


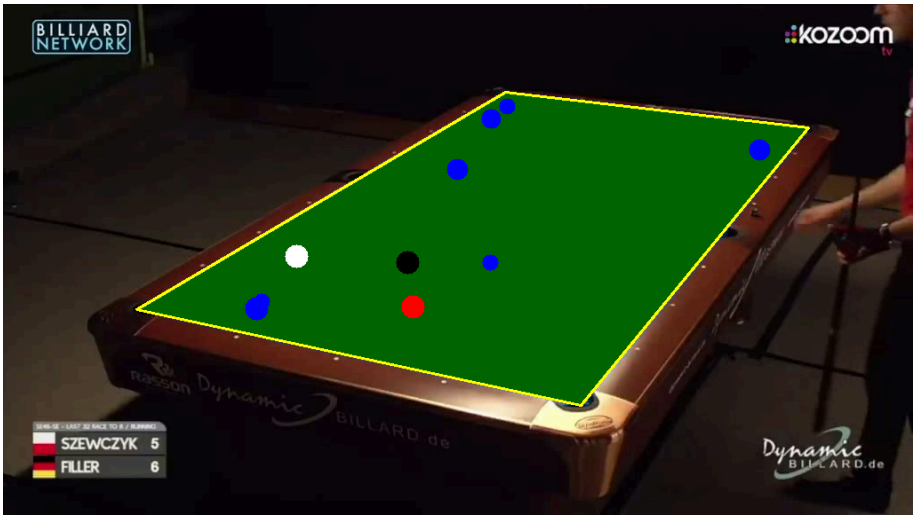
Game 3 clip 2



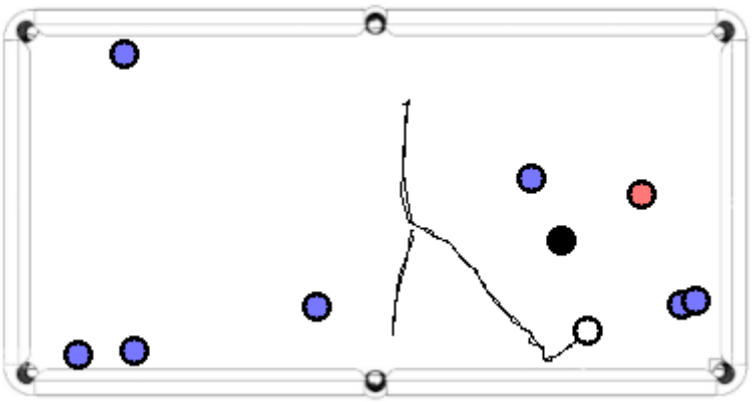


mAP First: 0.477273	mIoU First: 0.575144

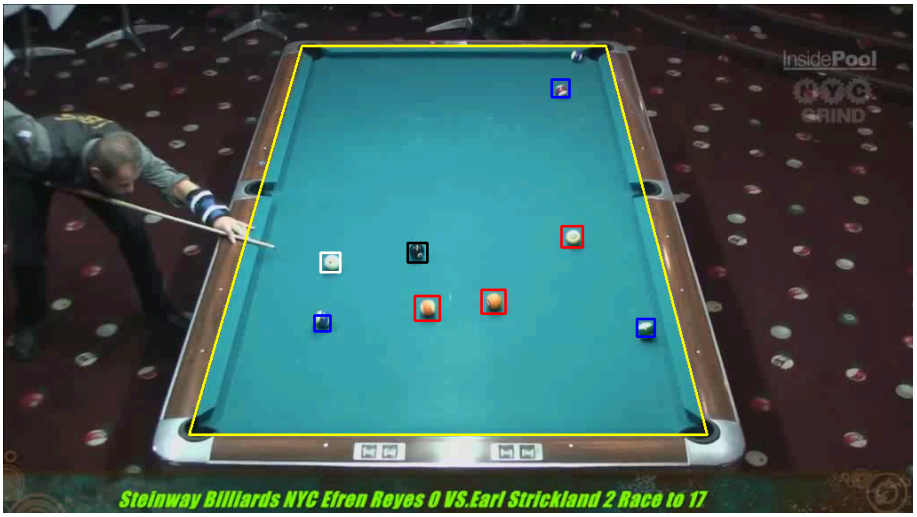




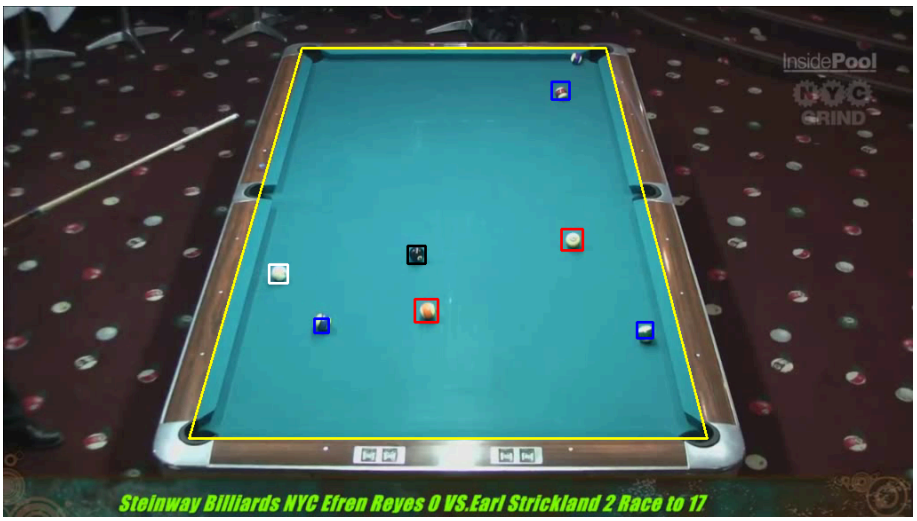
mAP Last: 0.477273	mIoU Last: 0.548171

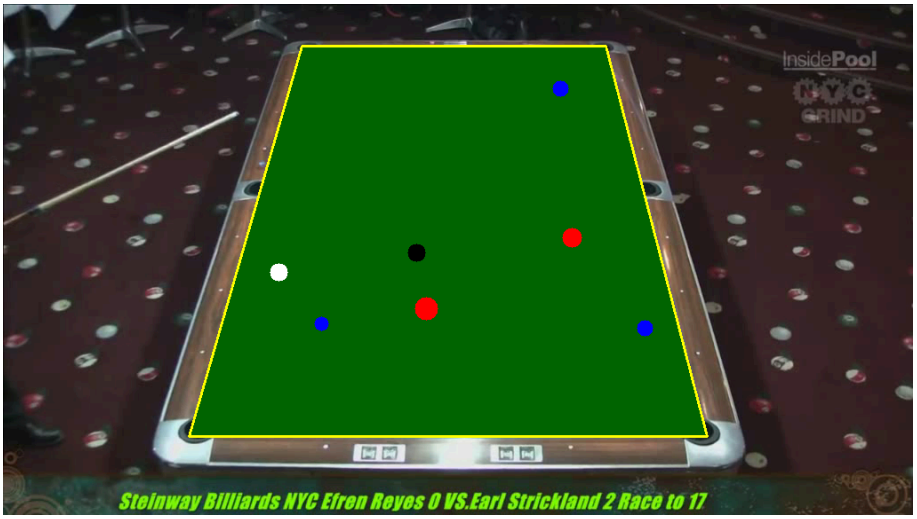


Game 4 clip 1

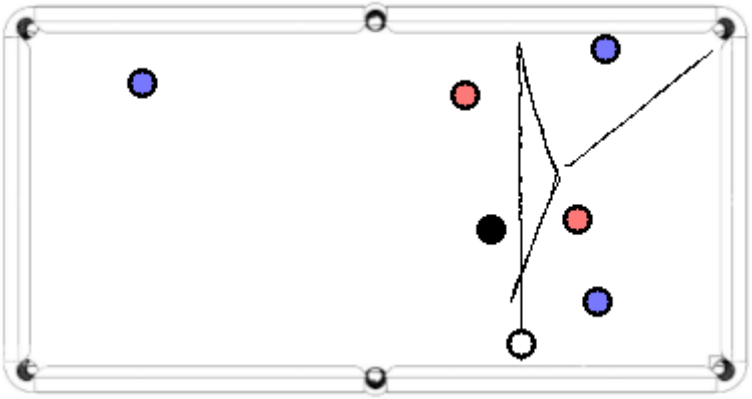


mAP First: 0.5	mIoU First: 0.63426

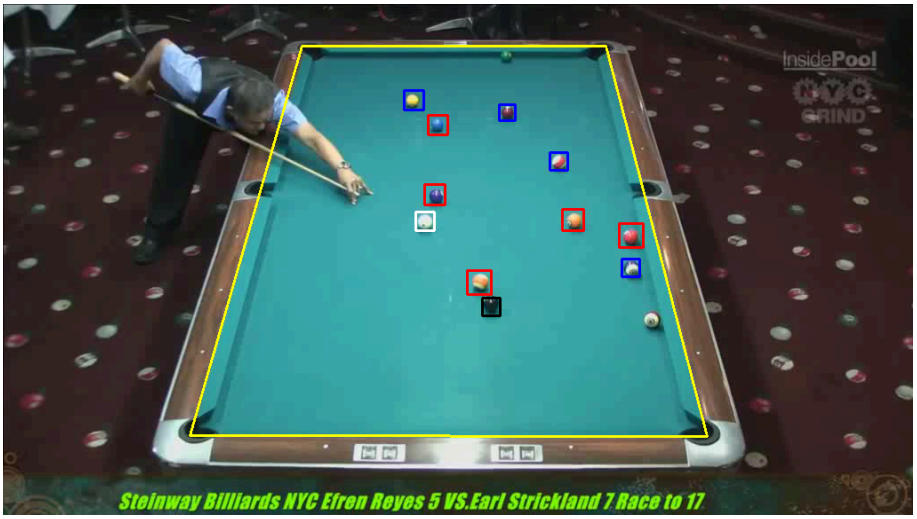




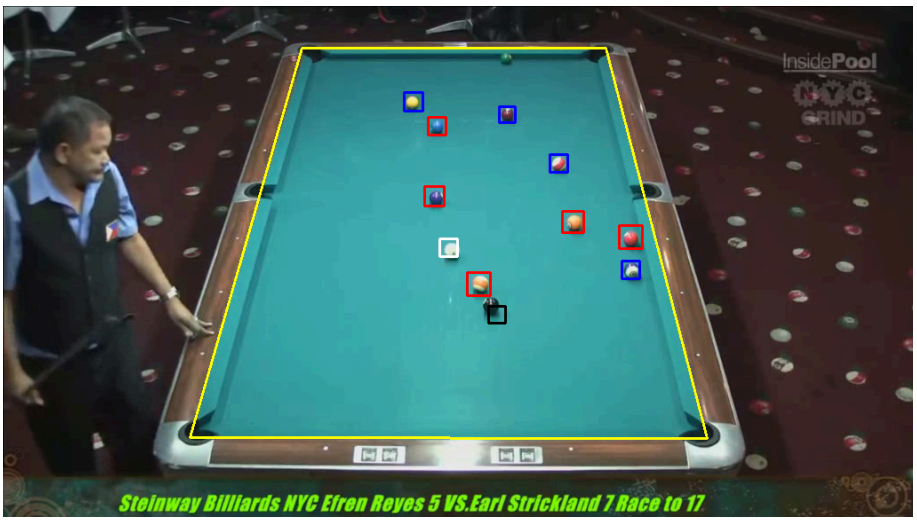
mAP Last: 0.681818	mIoU Last: 0.664233

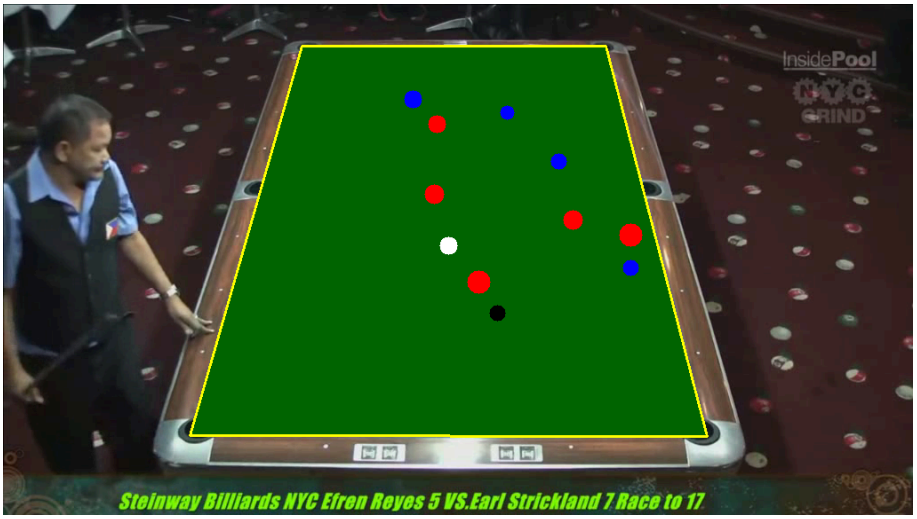


Game 4 clip 2

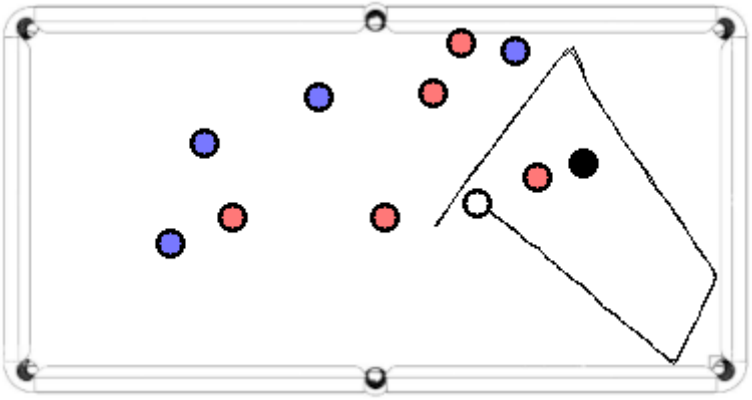


mAP First: 0.598485	mIoU First: 0.616235





mAP Last: 0.143939	mIoU Last: 0.480943



Overall mAP: 0.482299
Overall mIoU: 0.601502

11 Another possible approach - exploit the tracking phase for classification and false positive removal from the first frame

Assuming that a “good” segmentation is made (i.e. all the bounding boxes given to the tracker are only the ones belonging to balls), the white ball can be chosen as the first bounding box in the video that moves. This option was suggested by Riccardo and in practice worked in all the clips of the dataset. Another possible “less radical” solution is, since as outcome of the tracking phase a subset of balls that moves can be extracted and the white ball must belong to that set, it could be possible to search for it only there (thus increasing the probability of a correct classification).

If the knowledge of the tracking phase could have been exploited also for the classification of the first frame, a post-tracking phase with the objective of removing the false positive points left after the segmentation process of the first frame can be added. These kinds of problems are often due to the presence of a hand of the gamer inside the playing field that the localization process has wrongly detected as a ball. Marco thought of dealing with this problem by introducing a function that checks if the final positions of the bounding boxes (i.e. in the last frame of the video) fall outside the playing field. A first attempt has been made by comparing the balls positions with a mask containing white pixels if the point belongs to the playing field and zero outside. This strategy that seems reasonable at first sight had some drawbacks: after the tracking process, it happens that the ball localization is not always precise and hence the center of the ball (especially if the ball goes into a hole) can go outside the playing field (especially when there is a strong prospective effect due to the position of the camera with respect to the playing field) and the ball is discarded. In order to overcome this problem, the function should compare the position of the ball with respect not to the original mask but to a dilated version of it.

Unfortunately also this implementation has its drawbacks: in fact by enlarging the dimension of the area of accepted points, false positive points related to hands have not been deleted.

By this process it is possible to at least reduce the number of false positive balls and hence avoid both the plotting of the balls and the drawing of their trajectories (game1_clip3 and game2_clip2).