COMPUTER VISION FINAL PROJECT
Sport Analysis for billiard matches

Marco Panizzo 2086521 Aaron Prevedello 2089401
Matteo Dal Nevo 2087919

July 21, 2024

Introduction

This project aims to develop a computer vision system for analyzing video footage of "Eight Ball" billiard matches. The primary goals are to detect the main lines of the pool table, localize the balls inside the playing field and classify them based on their category. The system needs to be adaptable, since various game recordings are considered. They differ in:

- Type of camera framing
- Number of balls in play
- Color of the pool table

The performance will be evaluated using metrics like mean Average Precision (mAP) for balls localization and classification and mean Intersection over Union (mIoU) for segmentation tasks. The project is developed in C++ using the OpenCV library.

Implementation

We identified seven main tasks to carry out the project:

- **Video Extrapolation:** given the video footage extrapolate its frames;
- **Table Detection/Localization:** given a billiard match image, find the corners of the pool table;
- **Balls Detection/Localization:** given a billiard match image and the corners of the pool table, find the bounding boxes around each ball in the playing field;
- **Hand Segmentation:** given a billiard match image and the corners of the pool table, isolate the players part inside the playable field;
- **Balls Classification:** classify the detected balls based on their category;
- **Metrics Computations:** implement metrics to evaluate system performance;

- **State Representation:** represent the current state of the game in a 2D top-view visualization map, to be updated at each new frame with the current ball positions and the trajectory of each ball that is moving;

Table Detection **(Matteo Dal Nevo, Aaron Prevedello)**

The initial approach to table detection aimed at identifying the main lines representing the table's borders and computing their intersections to determine the corners. The Canny edge detector was applied directly to input frames, followed by line detection using the Hough Lines transform. Despite a long time spent on tuning the parameters for Canny and Hough, the result consisted in detection of multiple overlaid lines that prevented from identifying precisely the table on all the images. This was caused also by the variety of the images in the dataset, as some games were easier to analyze with respect to others due to differences in perspective, ball distance, lighting conditions, and non uniform background. In addition, in some images the player's stick was also detected as a line, which, as an outlier, adversely affected the subsequent corner detection computations, leading to poor results.

An attempt was made to define a region of interest (ROI) based on these lines and use it as input for the GrabCut segmentation algorithm for foreground extraction, available in OpenCV library. Although this approach yielded satisfactory results in table segmentation, it required a reapplication of line detection using Hough Lines. The improvement in line precision was not sufficient, therefore a different strategy was attempted.

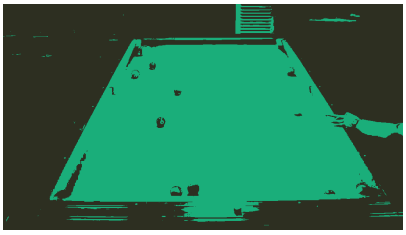
The new tentative aimed to detect the table's color and use a color-based mask to isolate the table region. After different attempts using Mean Shift algorithm, the choice has been K-means algorithm, yielding excellent results with two clusters, as the table represented the largest uniform area in the image, as in the figure below.

(a) Blue color table (b) Green color table

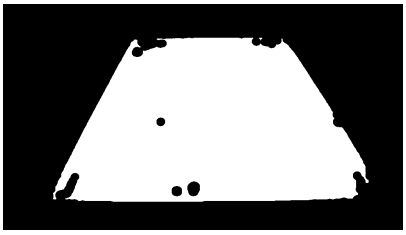
Figure 1: Detected colors of the tables

It was observed that the table's color consistently appeared as the second-largest cluster. This insight was used to extrapolate the table region. The other cluster generally had a darker color, suggesting a more robust method to identify the table color by selecting the brighter cluster, that has not taken in consideration.

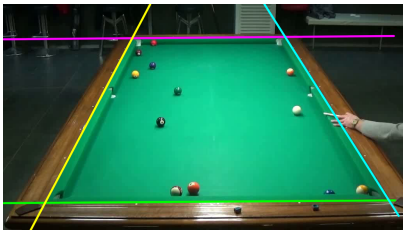
Since the resulting images contained other regions with the same table color, the goal was to isolate a single connected component corresponding to the table. Some images posed additional challenges due to poor lighting conditions affecting the table's upper part masking.



(a) K-Means Result

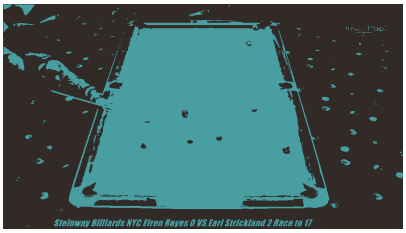


(b) Final Table Mask

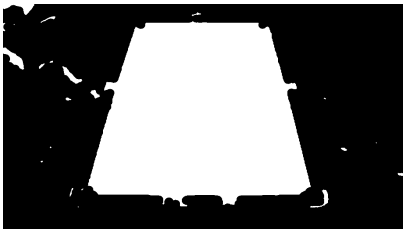


(c) Table lines and corners

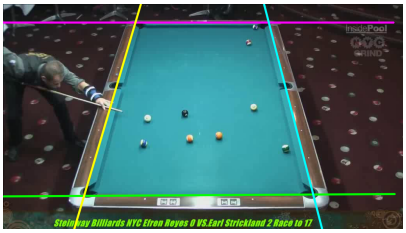
Figure 2: Table detection steps



(a) K-Means Result



(b) Final Table Mask



(c) Table lines and corners

Figure 3: Table detection steps

To address these issues, morphological operations were applied. Opening was used to smooth contours and eliminate protrusions, followed by closing to fuse narrow breaks in the image without increasing the table region's size. Various shapes of structuring elements were tested, with the ellipse shape yielding the best results due to its quite uniform action in all directions.

Subsequently, Hough Line detection was applied to the computed mask, and after fine-tuning the thresholds, lines corresponding to the table sides were successfully detected. To deal with cases where multiple lines overlapped appear on the same side, functions developed during the previous approach were used again to extract single representative line for each side.

The final step involved splitting the lines to identify each of the four sides and then computing the intersection points between pairs of lines. Information on (ρ, θ) extracted from the lines was used to infer their position in order to later identify the corners in precise order.
The thresholds for splitting the lines based on their slopes were adapted to the available dataset, which exhibited varied table orientations.
In fact vertical and horizontal lines were interpreted relatively to the image being visualized.
The final output of this procedure are the corners coordinates, that are computed in counter clockwise order, to be correctly processed in the future homography part.

Balls Detection and Hands Segmentation (Marco Panizzo)

The ball detection task was a fundamental part of the overall project, as the project's success depended on accurate ball detection. In the early stages of the project, various possible approaches to detect the balls were discussed. The main idea was to utilize the Hough Circle Transform provided by the OpenCV library, which had already been tested during the labs. Another possibility considered was using the Haar Cascade classifier, as seen in the Viola-Jones algorithm for face detection.

Since this task was performed after the table detection part, consideration was given to utilizing the results from this previous process, such as the segmentation of the table, which excludes the balls and thus provides them as black circular areas in the mask. Initially, this approach seemed the most reasonable. The idea was to apply the Hough Circle Transform on the table mask, limiting the function to the table area and not on the original image.
However, upon further testing, it was observed that the table mask degraded as successive game clips were processed. Consequently, this idea was dropped.

In parallel, datasets were created to try the Haar Cascade classifier.
According to the documentation, the OpenCV library provides an executable to train Haar Cascades (available till version 3.x of OpenCV). This function requires a positive dataset containing images of the objects of interest and a negative dataset containing irrelevant objects. After creating the datasets and conducting some quick tests, no meaningful results were observed. Given the long training times and uncertainty about the viability of this approach, the decision was made to return to the primary idea: using the Hough Circle Transform.

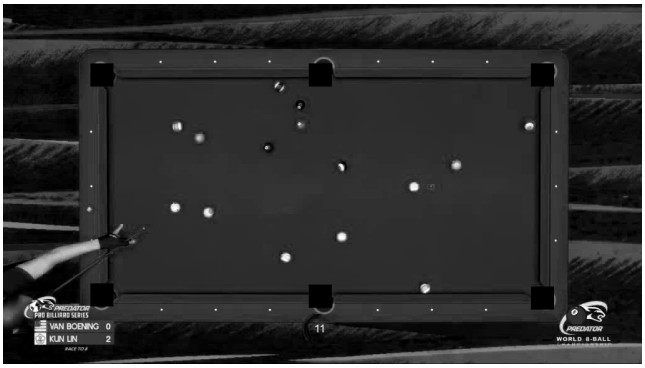
The Hough Circle Transform was the final selected approach. Firstly, tests were conducted to evaluate the circles detected by the algorithm using different function parameters. These tests were applied to the original image, a gray-scale image, a Sobel image, and a Laplacian image. This allowed an understanding of how the algorithm would perform on these different types of images. To speed up the process, the parameters were controlled using track-bars on the images so that the results could be adjusted in real-time.
In the original and gray-scale images, the results were far from optimal. However, for the Sobel and Laplacian images, the results were decent. The Laplacian approach clearly suffered from the double ringing effect due to the second derivative.

Among all the methods, the Sobel approach yielded the best results according to the evaluation, but it was still not optimal. Even after significant time spent on tuning, it was not possible to find parameters that simultaneously achieve the detection of all the balls with a low presence of false positive circles. Part of this issue was due to the fact that the derivative also highlighted some components that were not balls. By analyzing the images to reduce this "noise," it was decided to introduce a threshold transformation on the Sobel images to create a binary image.

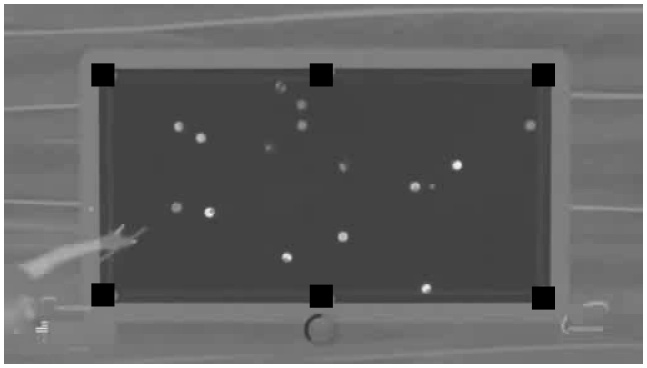
This issue could be partially addressed by tuning the first parameter of the Hough Circles Transform, the param1 (threshold for edge detection), but the results differed from the initial approach. By adjusting the threshold value, most of the lower sparse white areas that added false circles could be removed.
To better characterize the remaining areas, a Gaussian blur was applied before the Hough Transform. This additional step helped detect some previously undetected circles.
Despite these adjustments, the solution remained sub-optimal, as some balls in certain clips were

still missed. However, it was recognized that this process could be a valuable approach, so ways to increase the contrast between the table field and the balls were explored to help the Sobel filter find stronger edges around the missed balls.

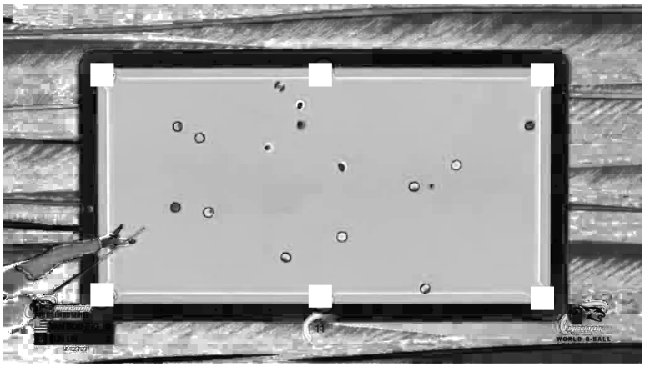
Upon reviewing the study cases, it was noticed that the area surrounding the balls (the table) typically did not have a high red color component. Exploiting this observation, isolating the red channel of the BGR image, treating it as a gray-scale image, and processing it with the previously discussed procedure resulted in detecting more circles.
Delving deeper into this concept, all the possible single channels of various color spaces were examined: BGR, HSV, Lab, YUV, Luv, and HLS.
The focus was on the channels that highlighted the balls most effectively across all clips. The selected channels were the Red channel from BGR, the V component from YUV, and the Saturation and Value components from HSV. Combining the circles detected from these channels provided a nearly perfect overall detection but also introduced some different false positives and clearly more circles for the same ball.



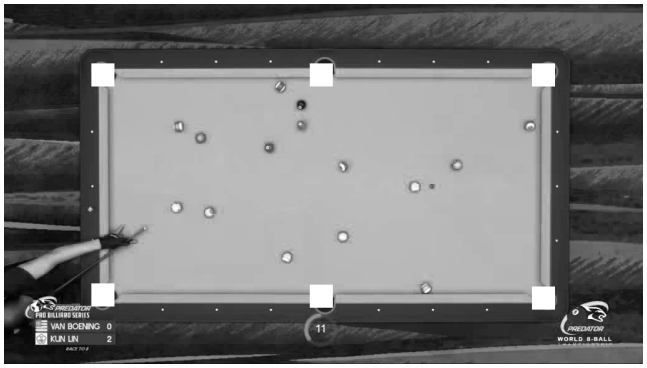
(a) Red channel of BGR



(b) V channel of YUV



(c) S channel of HSV

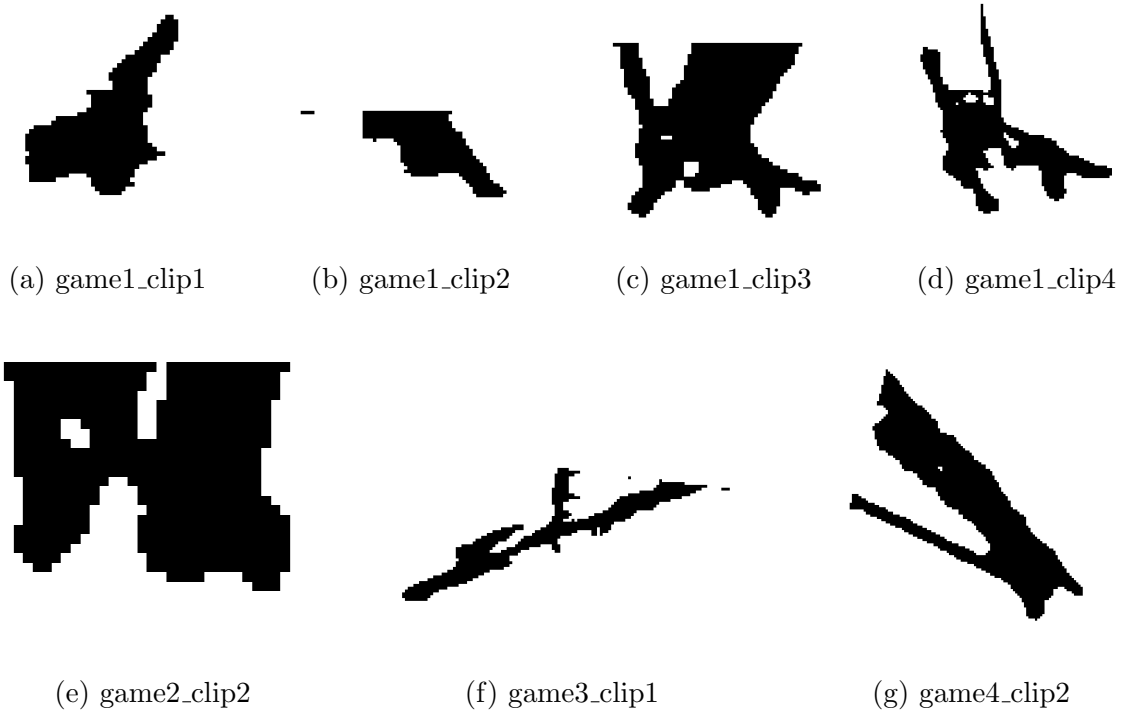


(d) V channel of HSV

Figure 4: Selected channels from the different color spaces

The successive steps involved merging the circles corresponding to the same balls and filtering out false positives as much as possible. Some false positives still arose from ball components such as shadows, changes in brightness, or the colored parts of striped balls seen from certain perspective. Another issue was the false positive circles on the player's arm and hand. To handle as many cases as possible, different functions were created:

- `mergeCircles`: This function merges circles based on the distances between their centers. If the centers are close to each other (under a specified threshold), the circles are merged.
- `filteredCircles`: This function checks if the circles are within the playable portion of the table and removes all circles outside of it.
- `mergeBoundingBoxes`: Some artifacts around the balls can be considered circles themselves. However, these artifacts are usually much smaller compared to the actual ball (but still larger than the minimum radius of the Hough Circles). This function checks this condition, and if a smaller circle is detected close to a bigger one (within specified thresholds), it will be removed. Additionally, if two bounding boxes share a significant amount of area, they will be merged.
- `filterBoundingBoxes`: This function checks the number of pixels with a color similar to the table. If this number is higher than a specified value, the bounding box will be discarded.
- `HandMaskFiltering`: This function creates a mask of the hand (hand and stick), which is necessary since different false positives were detected in the hand region. To segment the image, the saturation channel of the HSV color space was utilized and a threshold was computed to generate a binary image.
  The hand was separated from the table (where the table is white and the balls are black). The region of interest (ROI) was computed to fill the background with solid black. Then, connected components were used to keep the hand black and remove all the balls. Now, the hand is solid black, and the background is black as well. In the field, there are no balls. To keep only the black hand, the ROI was repeated, and the background was filled with white. Lastly, the bounding boxes that present black pixels inside of them are deleted.

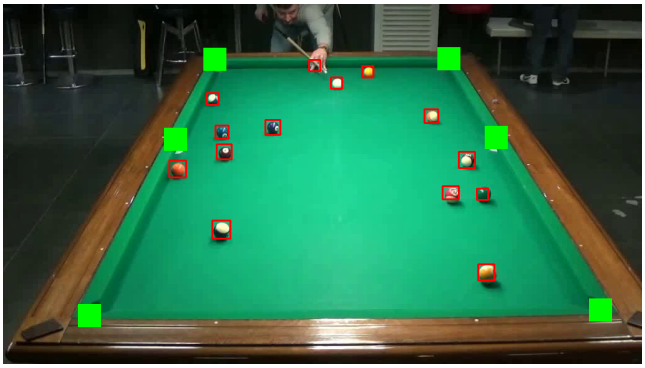


(a) game1_clip1 (b) game1_clip2 (c) game1_clip3 (d) game1_clip4

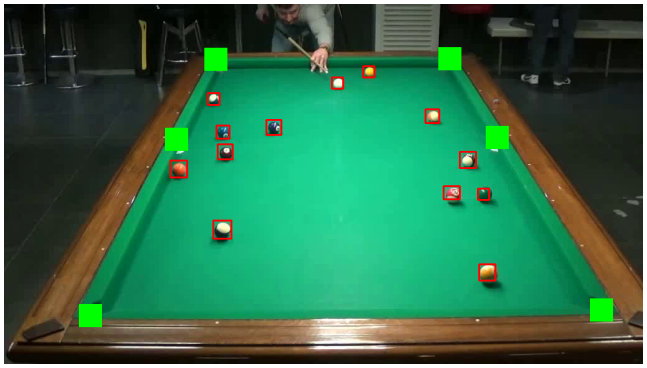
(e) game2_clip2 (f) game3_clip1 (g) game4_clip2

Figure 5: Examples of hand and stick mask

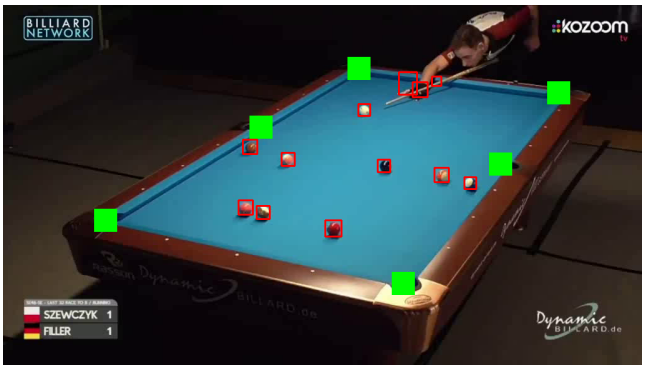
Detailed explanations of all the functions involved in this part are provided both in the header and .cpp files.
Additionally, squares were inserted to hide the holes (pockets) of the field. The overall function that handles the process of ball detection and hand segmentation is `ballsHandDetection`. The function takes as input an image from the game footage and the table corners. It then returns a vector of rectangles that correspond to the bounding boxes and the mask of the hand, which is used subsequently.
Some examples can be seen in the following images.



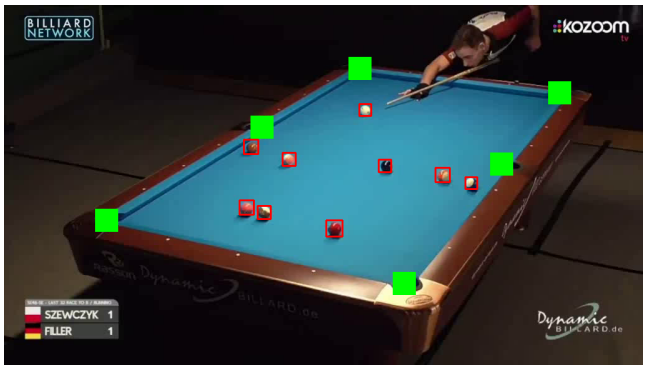
(a) Without hand mask



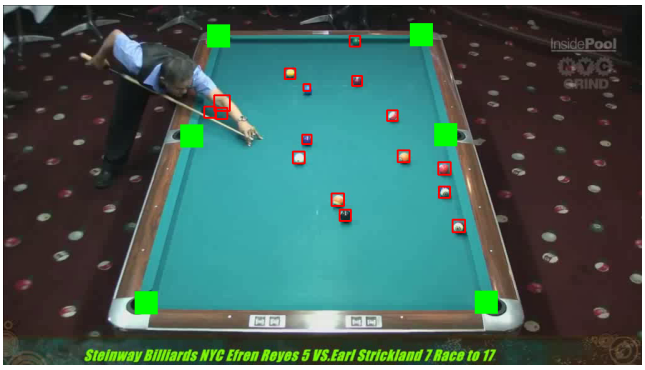
(b) With hand mask



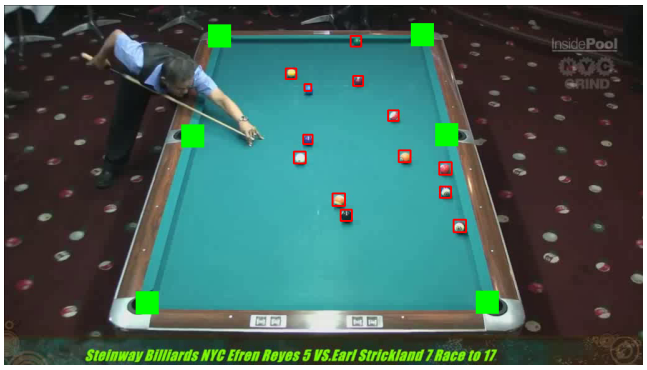
(c) Without hand mask



(d) With hand mask

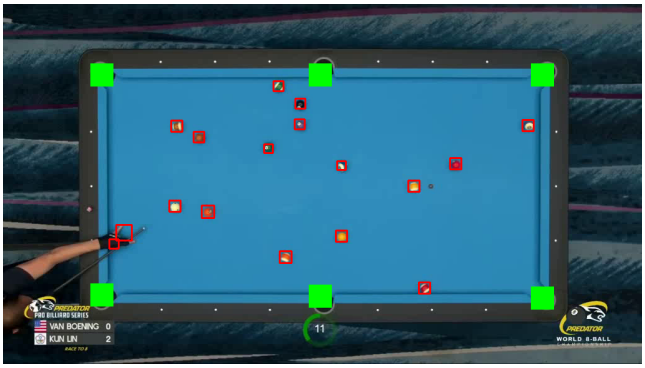


(e) Without hand mask

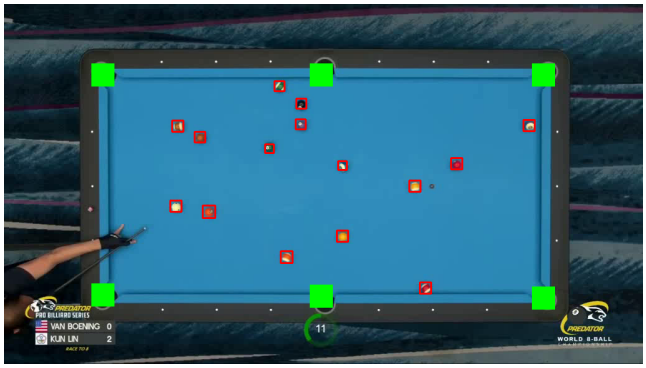


(f) With hand mask

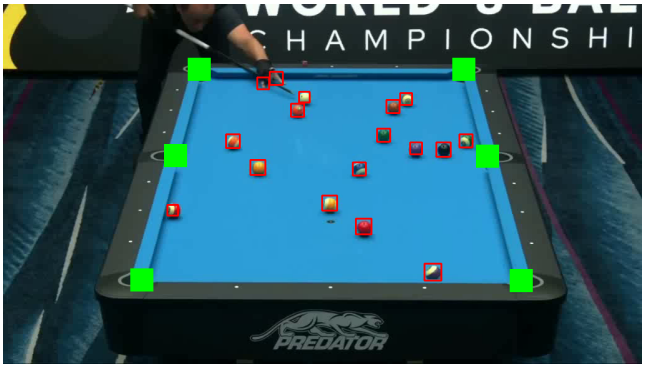
Figure 6: Results of Hand Mask False Positive Suppression



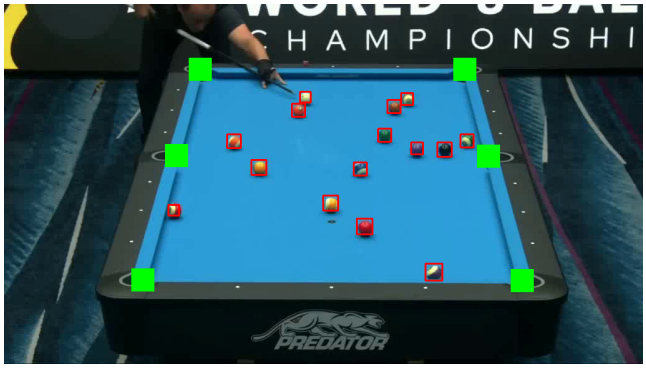
(a) Without hand mask



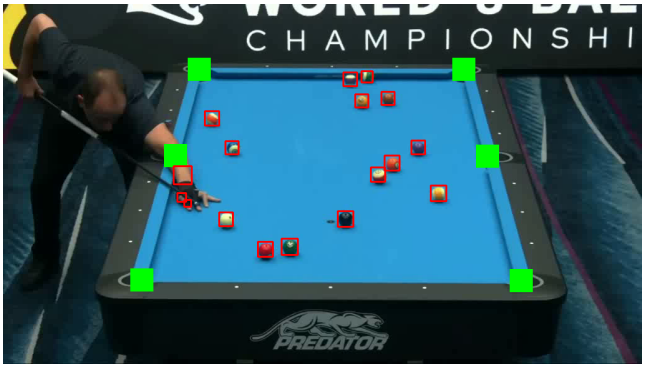
(b) With hand mask



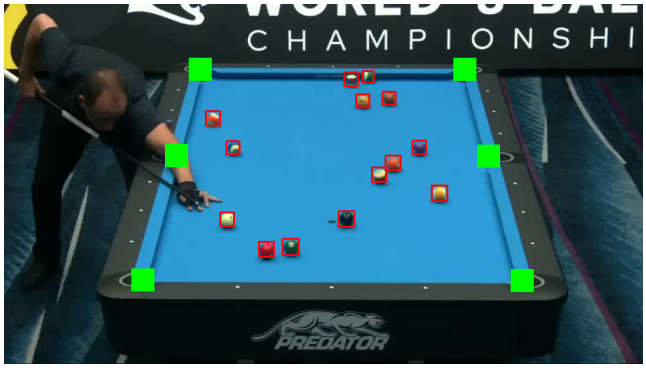
(c) Without hand mask



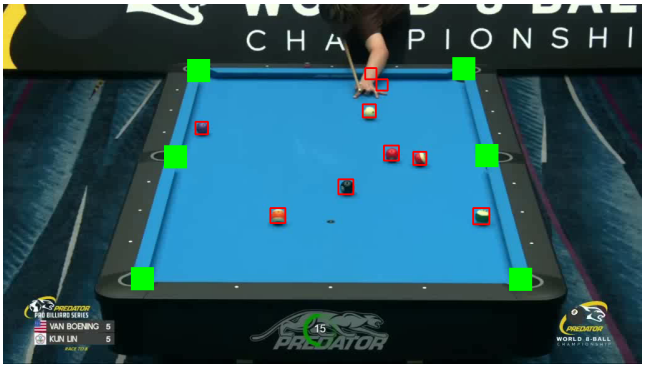
(d) With hand mask



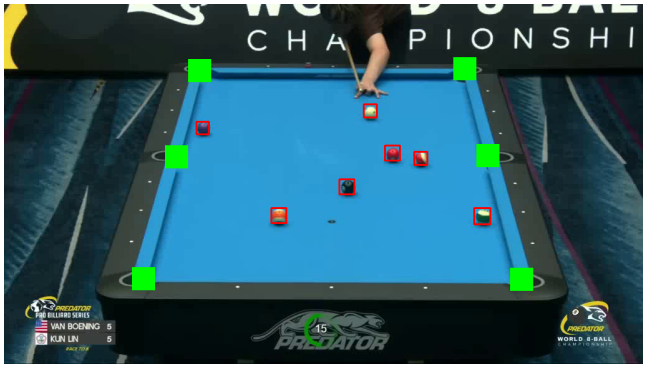
(e) Without hand mask



(f) With hand mask



(g) Without hand mask



(h) With hand mask

Figure 7: Results of Hand Mask False Positive Suppression

Balls Classification (Matteo Dal Nevo)

As mentioned in the previous section, the balls detection task provides the bounding boxes of the detected balls. Given this important information, another fundamental part of the project is to correctly classify the balls based on their category.

The primary idea is to consider the region of interest dictated by the bounding box for each ball and analyze the percentage of black and white pixels inside this region in grayscale. As one can imagine, a white ball or a ball with stripes should have a high percentage of white pixels. Conversely, solid-colored balls should have a lower or zero percentage of white pixels.
To ensure that the bounding boxes entirely include all the balls, their dimensions were enlarged. This adjustment was necessary because the dimensions of the bounding boxes vary due to perspective, and not all the bounding boxes perfectly fit the balls' radii. This has been done by `adjustBoundingBox`.
This method proved to be very effective across all game clips. The logic employed was to classify as many balls as possible using different binary thresholds for black or white pixels. Once classified, these balls were excluded from further analysis, and different thresholds or approaches were applied to classify the remaining balls.
The classification method was divided into several cascade phases:

- The first phase aimed to correctly classify the black and white balls. These balls exhibit the highest percentages of black and white pixels in the image, respectively. One challenge of this approach was that yellow balls, whether solid-colored or with stripes, also presented a high percentage of white pixels. Without a proper threshold, the classification could be incorrect due to changes in illumination.

- In the second phase, some of the solid-colored balls were correctly classified. Data analysis on the white pixel percentage values from all the images enabled the derivation of a threshold. Balls with a very low percentage of white pixels were identified as completely solid-colored.

- The third phase addressed the challenge of classifying some balls with stripes. A preprocessing mask procedure was developed to improve the distinction of striped balls. This phase involved creating a mask for the light ivory color (BGR: 175, 242, 252), which is a common color in striped balls. The inRange function was used to create a binary mask that identified the pixels in the region of interest of the ball that fell within the specified color range. The pixels that match the light ivory color criteria appeared white in the mask, while others were black. This mask was then combined with the grayscale image of the ball using a bitwise AND operation. Based on data analysis, a threshold for the percentage of white pixels was determined, highlighting the stripes and differentiating them from solid colors.

- The fourth phase used the same approach to the third phase but considered a dark ivory color (BGR: 86, 131, 140), but for classify other solid-colored balls. This phase ensured that more solid-colored balls were correctly identified by refining the thresholds for white pixels and adjusting the preprocessing steps accordingly.

- In the fifth phase, the remaining balls were classified based on their white pixel percentage. This phase aimed to resolve any ambiguities left from the previous steps and finalize the classification process.

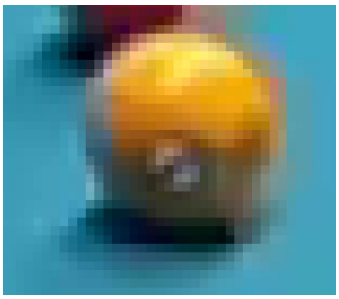
- The final phase involved removing any fake bounding boxes based on the table color. This

step ensured that only actual balls were considered, and any false positives caused by the table color were eliminated.

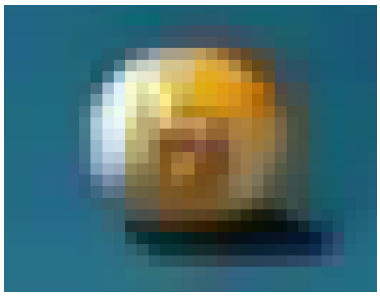
The overall function that handles the process of state representation and tracking is `ballClassification`. It takes as input an image of the game (frame first / frame last) and the relative position of the bounding boxes. Then, it returns the new bounding boxes but with an identification parameter for the class to which it belongs. It is subdivided into smaller functions, where the principal ones are:

- `calculateWhitePixelPercentage`: Compute the white pixel percentage in a grayscale image based on a binary threshold.
- `calculateBlackPixelPercentage`: Compute the black pixel percentage in a grayscale image based on a binary threshold.
- `adjustBoundingBox`: Modify the bounding boxes dimension.
- `computeWhiteBlackBallIndexes`: Find the indexes of the bounding boxes relative to the balls that maximize the white and the black pixels percentage, respectively.
- `processBoundingBox`: Process the bounding box image creating a mask based on a color range and make a bitwise AND with the ball grasycale image. Then return the white pixels percentage.
- `classifySolidStripeBalls`: Classify solid-colored balls and balls with stripes. It is also robust to detect possible false positive bounding boxes if where is not a ball inside based on the color of the table.

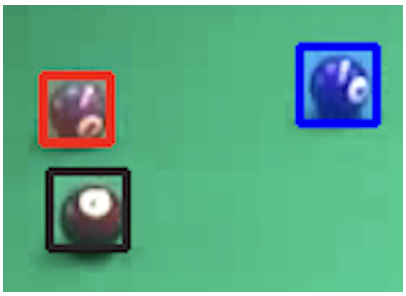
Some challenging examples can be seen in the following images, and will be discussed subsequently.



Solid-colored yellow ball



Striped yellow ball



Indistinguishable dark balls

Overlapped balls

Figure 8: Balls classification challenging problems

It is observed that the yellow balls, both solid-coloured and striped, result very similar. Indeed, the solid ball seems to present a small white component on the left, due probably to a different illumination, making their classification difficult. Similarly, in one of the games, the black ball was difficult to distinguish as the brown ball and the black ball appear very similar to each other. Additionally, there is a situation where the white ball is occluding a brown ball, which further complicates the classification process.

In addition to the primary method, another tentative approach involved using the k-means clustering procedure similar to the table detection part. In this approach, the ball images were clustered into three clusters to separate the white component from the solid-colored component. The goal was to determine which component was more prevalent. The third cluster, which was most similar to the table color, was excluded from the comparison. This method worked very well in the first games, but for the other ones, the difference in color prevalence and clarity diminished, making the classification less reliable.

State representation and tracking **(Marco Panizzo)**

The state representation is based on a 2D top-view scheme that shows the evolution of the game. The components needed to perform this task were table detection (corners) and ball detection and classification. The main idea behind this step was to utilize the above information to define the state of the first frame and then pass this information to the tracker provided by the OpenCV Library.
The tracker, `TrackerCSRT`, must be initialized with the first frame of the video and the bounding boxes of the balls to be tracked. By looping through the frames, the tracker will be updated for each frame, tracking the balls. It is not guaranteed that the tracker will always perform accurately, so it was necessary to verify its correct functioning. While looping, the center of the tracker's bounding box, which identifies the center of the ball, was saved into a vector representing the ball's position evolution.
To speed up the process, since the tracker takes some time to complete the operation, only the balls that showed a significantly different position between the initial and final frames within a small tolerance were tracked. The loop cycles through all frames and all moved balls. A check was also implemented to stop updating the tracking if the ball was considered to have exited the playable field.

Till now, the tracker has saved the real footage center positions of the balls. However, to be able to draw them into the scheme, it was necessary to find the homography matrix that describes the transformation between the table corners in the real footage and the corresponding corners in the scheme. First, a scheme was downloaded, and its corners were hand-checked since they will always remain the same. The difficult part was correctly aligning the corners of the real footage with those from the scheme. The scheme corners were saved counterclockwise, and the same was done for the table corners in the footage.
Some ideas were considered, but the final solution was based on theory. The diagonal elements of the homography matrix describe the stretching of the transformation along the components. So, the idea was to perform two homography transformations: one with the given table corners and one with a single step rotation. Then, see which of the resulting matrices had diagonal elements closer to value one. This solution worked well; however, during testing, it was noticed that in some cases, the two matrices were not significantly different. To make the process more robust, four

different matrices were computed instead of just two, ensuring that even if noise/error was higher, the check would yield the correct one matrix.
The switch from two to four different homography matrices could result in a 180° rotation of the table in successive clips. This outcome is still correct but not optimal. However, it was a trade-off accepted to achieve greater robustness.

Once the transformation matrix was found, it was used to transform the centers of the balls in the footage into the centers of the balls in the scheme. Then, the balls were drawn as filled circles using the color provided by the classification stored in the bounding box ID, and the trajectories were also drawn with the same color. Instead the white ball's trajectories were drawn in green.
This part was computed on the full-size image of the scheme table and then resized and attached to the original frames.
To help the tracker correctly track the balls, preprocessing was applied to the frames: they were converted into the HSV color space and the value component was doubled. Although the saturation was also modified, it did not improve the tracker's performance, so the original value was kept. Then, the BGR modified version was utilized. Another helpful modification was enlarging the bounding box to capture a more complete scene for tracking. This helped significantly in some particular clips where the tracker confused overlapping balls that were not well-defined.

The overall function that handles the process of state representation and tracking is `homography_track_balls`. It takes as input the clip frames, table corners, and the bounding boxes with classifications from the first and last frames. Then, it returns the new frames with the 2D top-view scheme added over the bottom left corner.
The function it is clearly subdivided into smaller functions, the principal ones being:

- `HSV_preprocessing`: Doubles the value channel of the HSV color space.
- `identifyMovedBalls`: Identifies and separates static balls from moving balls.
- `best_homog`: Computes the four homography matrices and selects the one with the least stretch component.
- `resizeAndCopyToFrame`: Resizes the drawn 2D top-view scheme and attaches it to the bottom left corner.
- Various `draw` functions: Draws static balls, moved balls, and trajectories.

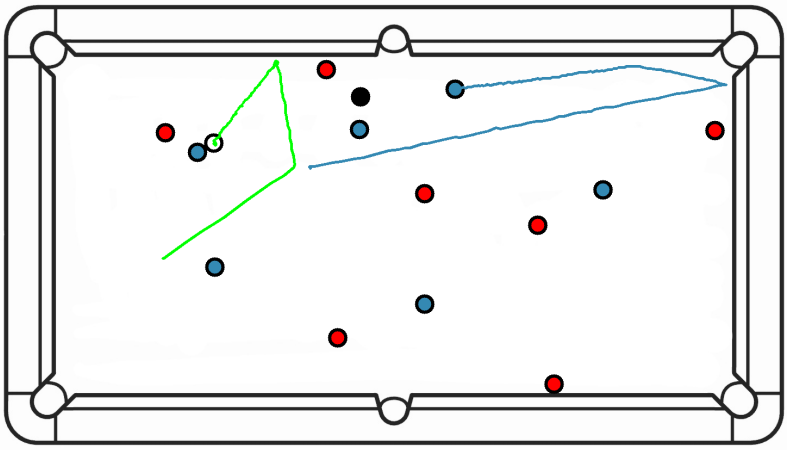


Figure 9: Trajectory scheme

Segmentation (Marco Panizzo)

The final segmented image was created by integrating the results from several preceding operations, each of which delivered satisfactory outcomes. The procedure involved multiple stages, utilizing detailed information from table detection, ball detection and classification, and hand segmentation.

To begin, a grayscale image with six distinct gray levels was generated. This grayscale image served as the foundation for further processing. A function was then defined to map this grayscale image to a BGR color image. This function was designed to be reusable, allowing it to map both the grayscale image and the provided ground truth.

An empty image, matching the dimensions of the original input, was created to facilitate the segmentation. Using the coordinates of the table corners, a mask was generated to delineate the table area. The OpenCV function `fillPoly` was employed to color this masked region in green.

For placing the balls in the image, the bounding boxes from the detected and classified balls were analyzed. The center of each bounding box was calculated using its width and height information. Since it is not possible to assume that the bounding box are always squares the radius of the drawn circles is equal to the smaller side. Each circle was colored according to the ball's classification ID. In the final step, the hand mask was applied to the image and pixels corresponding to the hand region, as indicated by the mask, were set to zero to exclude the hand from the final segmented image.

Overall, the quality of the final segmentation is heavily dependent on the accuracy and effectiveness of the preceding steps. Each operation, from detecting and classifying the table and balls to applying the hand mask, played a crucial role in achieving the final result.

Pipeline

Here the diagram for the final Pipelines:

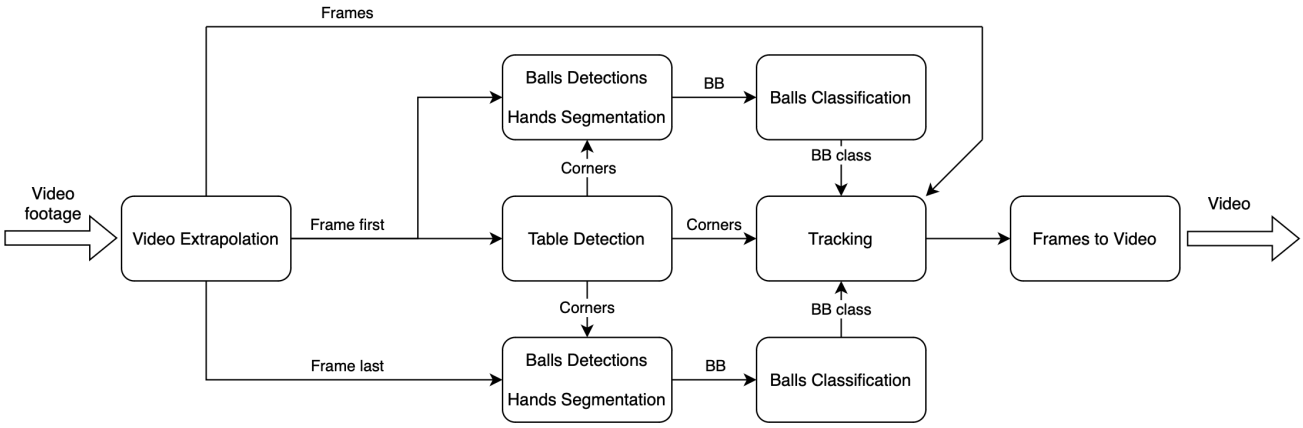


Figure 10: Flowchart of the Video computation

Starting from the input video footage of a game, we perform an extrapolation of the individual frames. Once the frames are extracted, the table detection module identifies the playing table's

corners using the first frame.
With the table corners detected, the next step involves detecting the balls and segmenting hands within the first and last frames. This step outputs bounding boxes (BB) around detected balls and segmented hands.
The bounding boxes generated in the previous step are passed to the balls classification module. Here, each detected ball is classified as either a solid-colored ball or a ball with stripes. Additionally, false positive bounding boxes (regions without any balls) are identified and discarded.
The classified bounding boxes, along with their classes (solid-colored or striped), are then fed into the tracking module. This module tracks the movement of the balls across frames creating a state representation of the game.
Finally, the tracked frames are reassembled into a video format and saved in a .mp4 file.

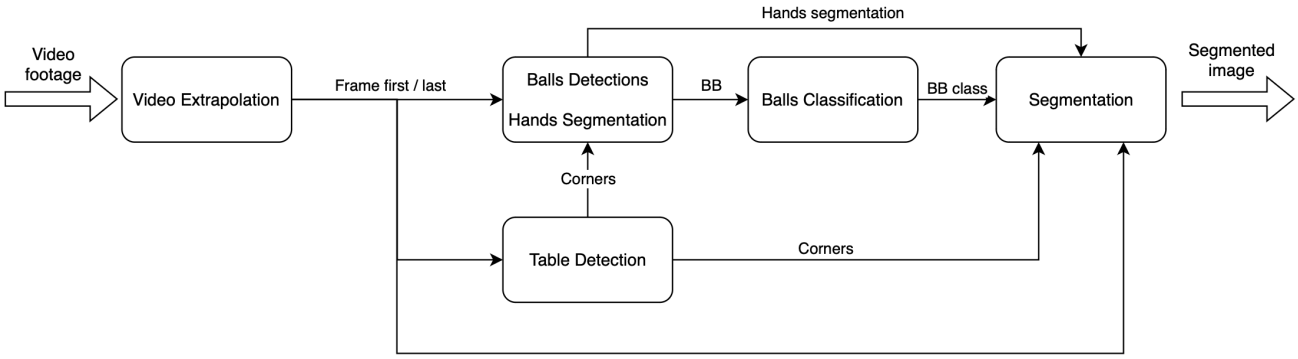


Figure 11: Flowchart of the Segmentation

The segmentation module integrates the results from the hands segmentation, balls detection, and balls classification stages, that works exactly as described before. It produces a segmented image where the classes of interest (i.e., background, pool table field and classified balls) are accurately identified and labeled.

The evaluation process is included in a separate function of the Pipeline. It simply relies on the implementation of the Metrics (mIoU and mAP), which are computed between the Pipeline's output and the ground truth.

Metrics Computations **(Aaron Prevedello)**

mIoU

Mean Intersection over Union (mIoU) is a crucial metric for evaluating the performance of image segmentation algorithms, particularly in the context of multi-class segmentation tasks, quantifying the accuracy of the predicted segmentation by comparing it to the ground truth.

The Intersection over Union (IoU) for a single class is defined as the ratio of the area of overlap between the predicted segmentation mask and the ground truth mask to the area of their union. To compute the mean IoU (mIoU) across a dataset, the IoU is first calculated for each class present in each image. These individual IoU values are then averaged over all the classes and images in the dataset. This averaging process ensures that each class, regardless of its size or frequency within

the images, contributes equally to the final performance metric. This is particularly important for our dataset where there are important differences between the size of the different classes in the segmented images, for example the difference of the area covered by the table class compared to that of the white ball class.

mAP

The Mean Average Precision (mAP) is a performance metric for evaluating the localization and classification of objects in a dataset. The computation of mAP is intricate, involving several steps to assess the effectiveness of object detection algorithms. Specifically, mAP measures the Average Precision (AP) for each class across the entire dataset, and then averages these AP values to provide an overall performance score.

The first step in calculating mAP involves determining the Intersection over Union (IoU) for each detected object compared to the ground truth for a specific class. A threshold is applied to classify each prediction as either a True Positive (TP) or False Positive (FP).

Following this, a Precision-Recall (PR) curve is computed. Ideally, this involves ordering predictions by confidence scores, however, in our implementation, all detection were assigned the same confidence level due to the fact that our detection process does not output any confidence level related to each detection, and having low rates of false positives and false negatives gives us a good confidence on all the detected balls in this dataset, despite some particular cases.
A possible idea was to utilize the confidence parameter obtained by the Hough Circle function, but it resulted difficult to keep trace of a solid and reliably estimation of that parameter when doing operations like merging and filtering of close detection.

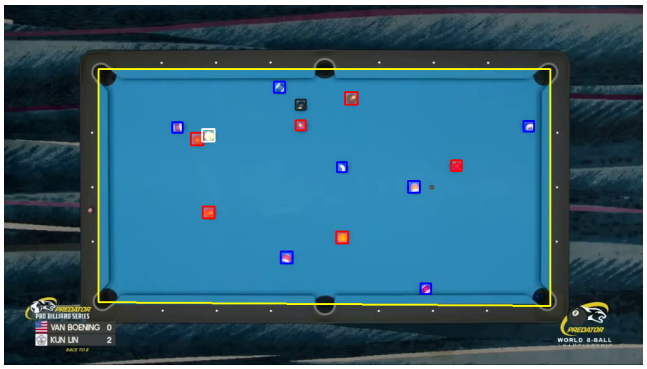
The Average Precision (AP) is then derived from the PR curve by integrating it implementing a function for the PASCAL VOC 11 interpolation method.
Initially, we considered using a function similar to that employed for the first metric to compute IoU values after drawing ground truth and predicted bounding boxes on empty images. But, this approach would yield a less precise performance measure as it would provide only a single IoU value per class per image.
To enhance precision, we adopted an approach that pairs each detected bounding box with the closest ground truth box (if available) and calculates the IoU for each pair. This method offers a more granular and accurate assessment of performance for each individual ball detected.

To pair predicted balls with ground truth boxes, we compute a distance matrix for each box pair and match each detection with the nearest ground truth. We have been very careful to deal with particular cases, for instance if no ground truth objects for a class is present, and also the algorithm does not detect any objects of that class, the situation is considered as a true negative and positively rewarded. Conversely, if the algorithm does not detect any object of a class but ground truth objects are present, the AP for that class will be zero. Those cases happened particularly in the $game4\_clip1$.

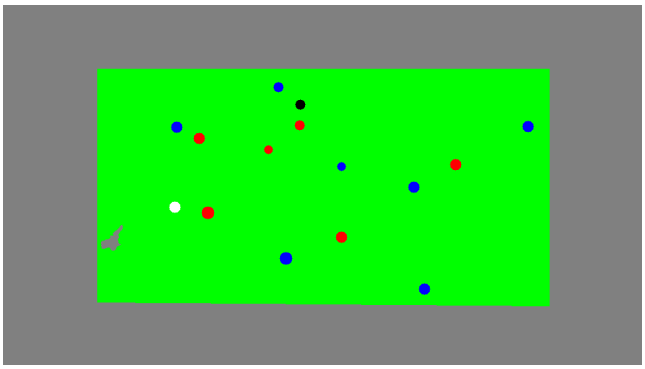
Upon analyzing the performance for each class individually, it is evident that the classification accuracy is lower for solid and striped balls. This is primarily due to the variety of conditions in the different images of the dataset. We aimed to achieve near-perfect classification for the white and black balls, as they are critical for gameplay. Misclassifying these balls would result in a zero Average Precision for those classes and the errors would propagate to the final tracking video.

Results and Discussions

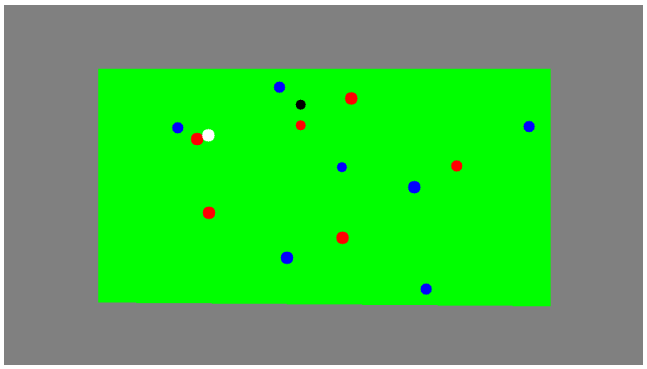
(a) Detection and Classification first



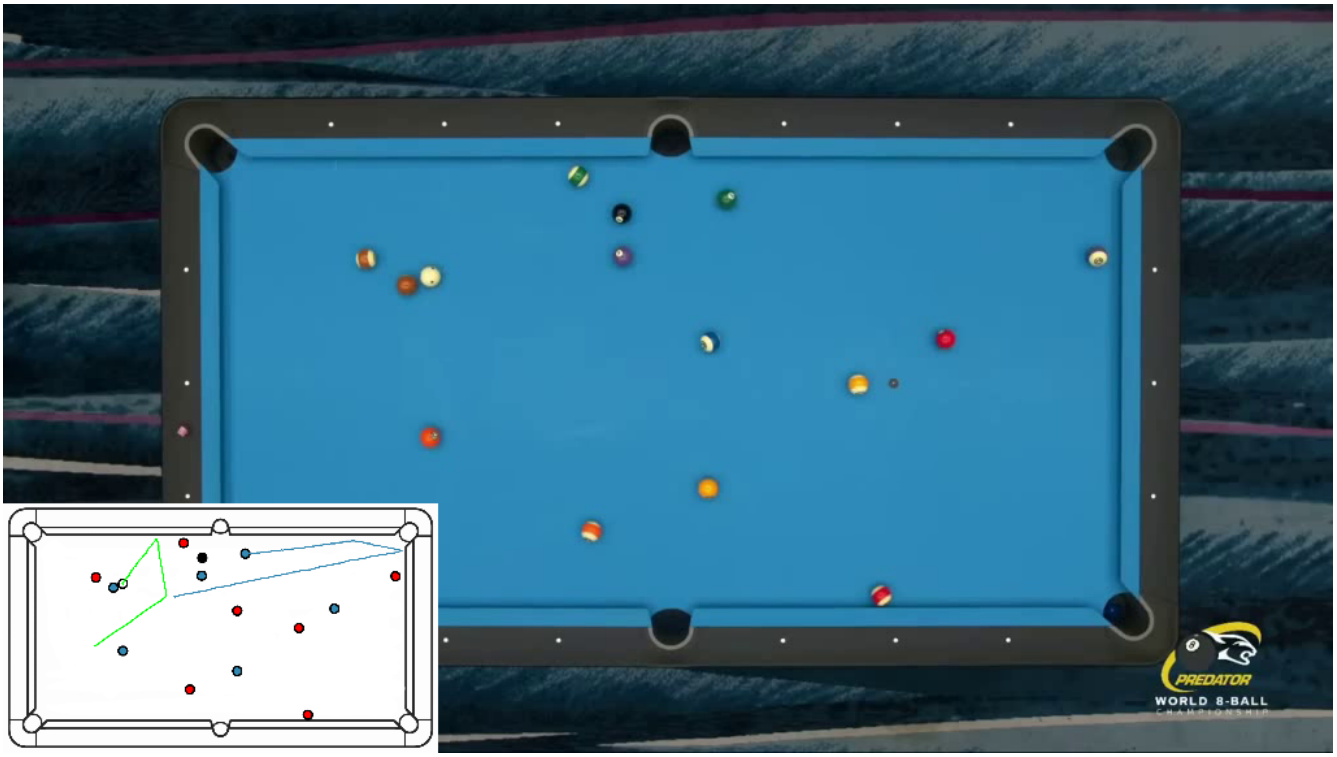
(b) Detection and Classification last



(c) Segmentation first frame

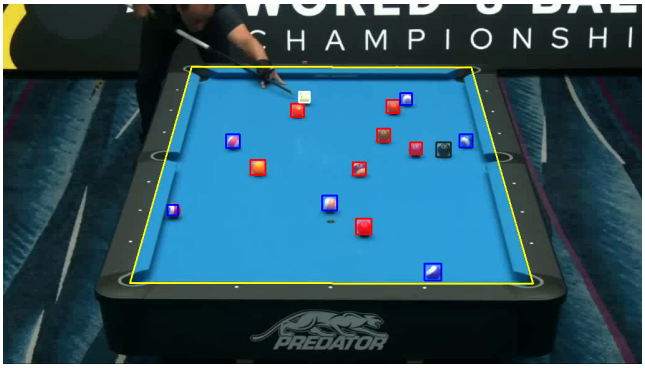


(d) Segmentation last frame

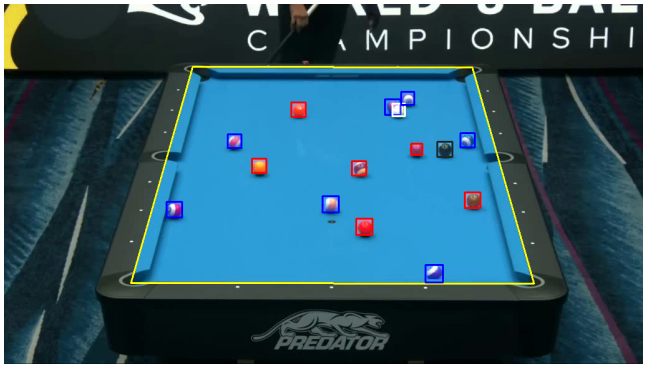


(e) Trajectory last frame

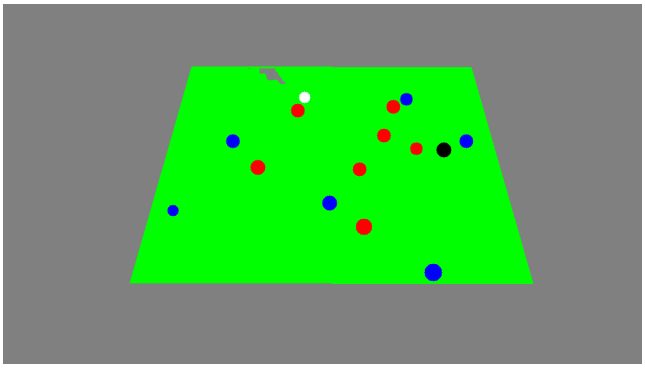
Figure 12: Game 1 Clip 1



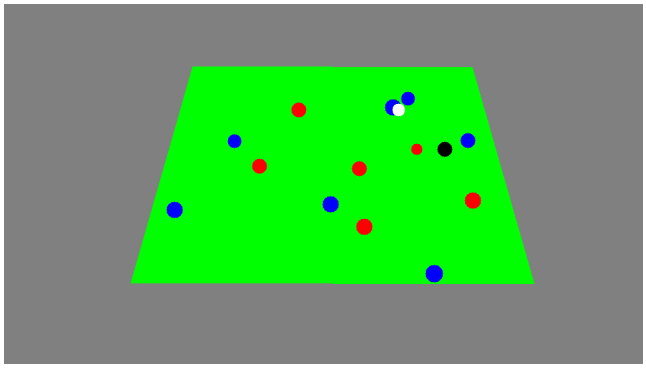
(a) Detection and Classification first



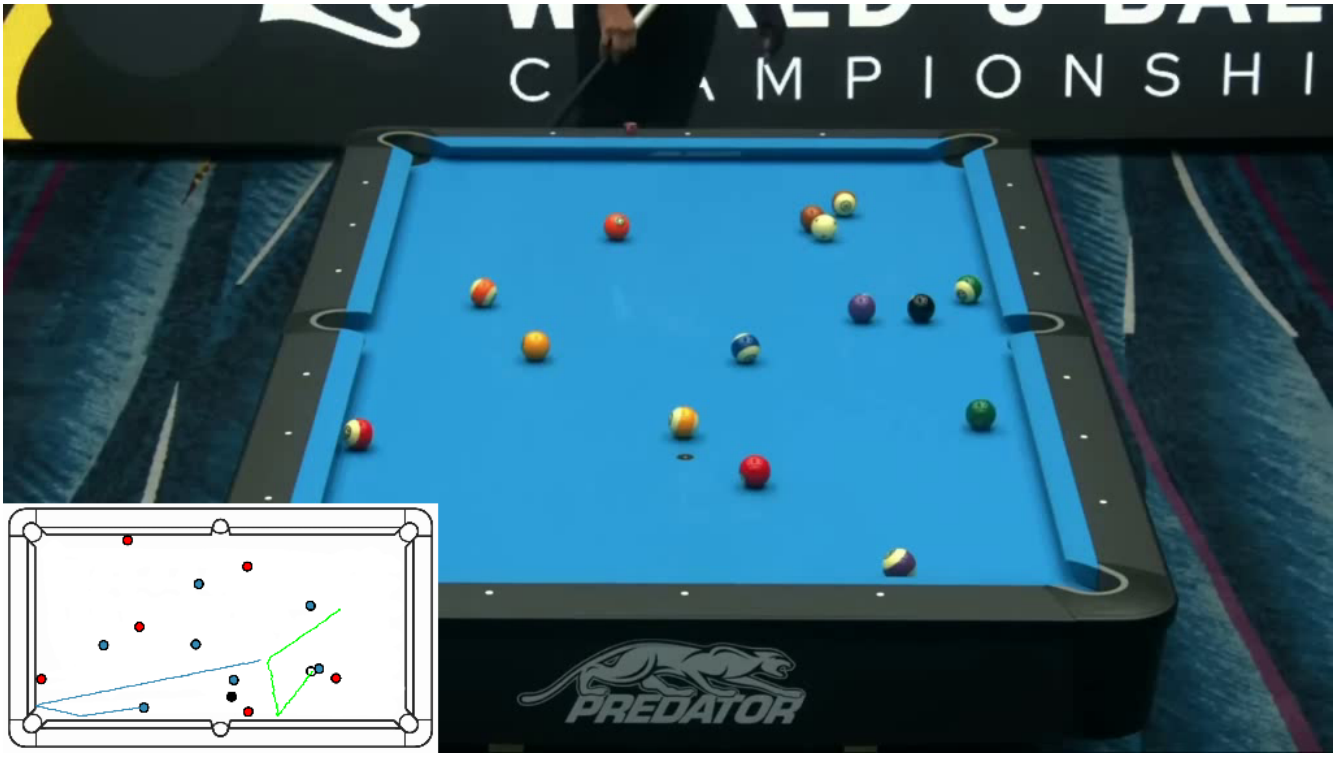
(b) Detection and Classification last



(c) Segmentation first frame



(d) Segmentation last frame

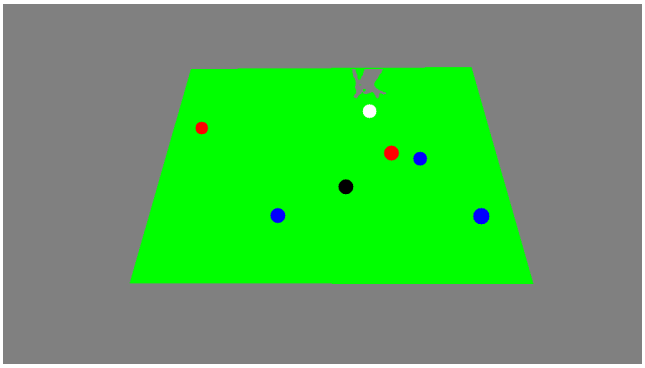


(e) Trajectory last frame

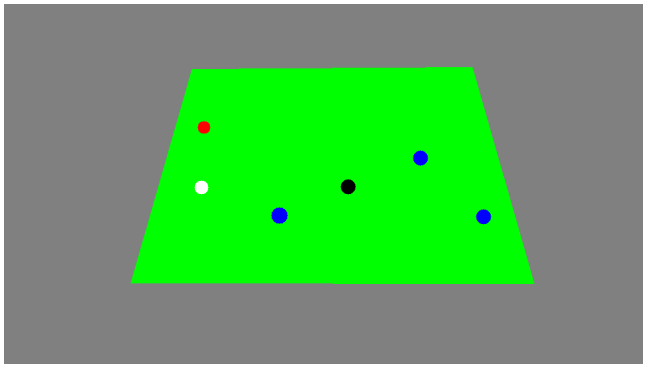
Figure 13: Game 1 Clip 2

(a) Detection and Classification first

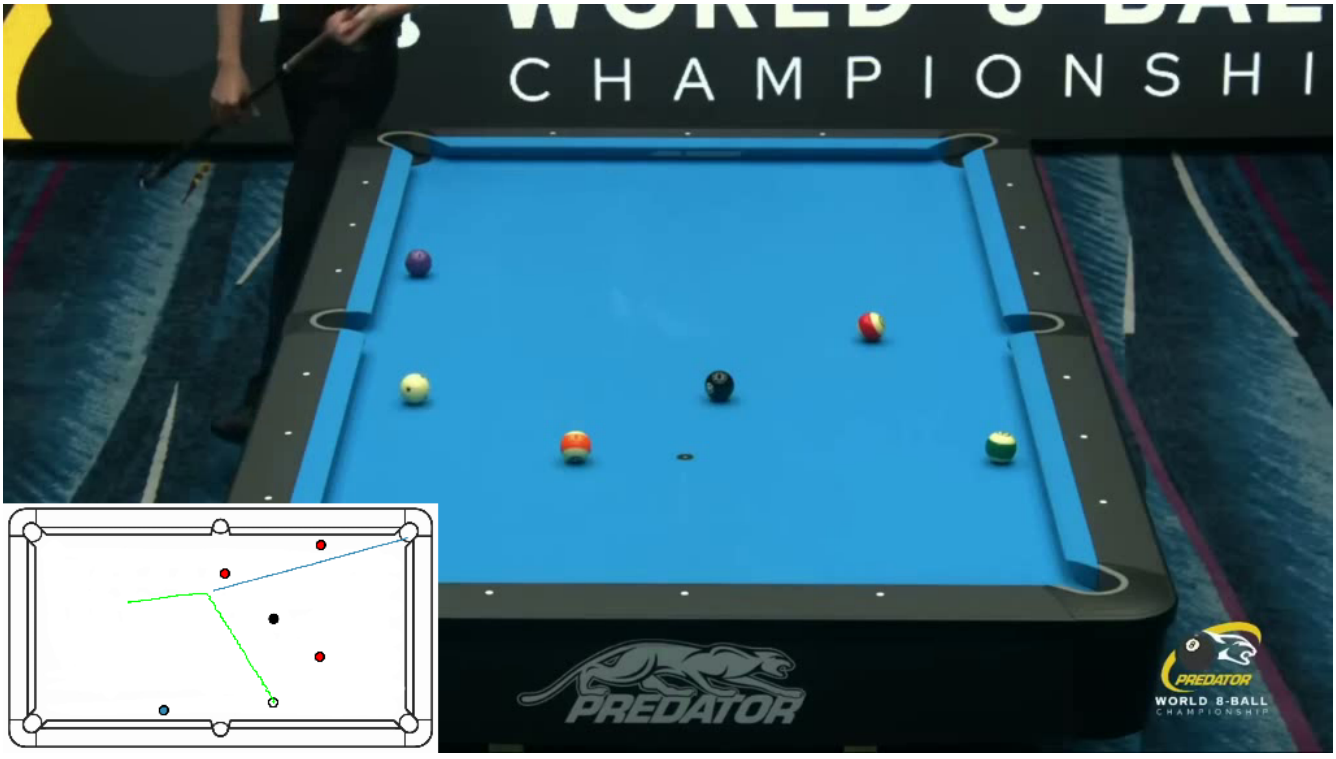
(b) Detection and Classification last



(c) Segmentation first frame

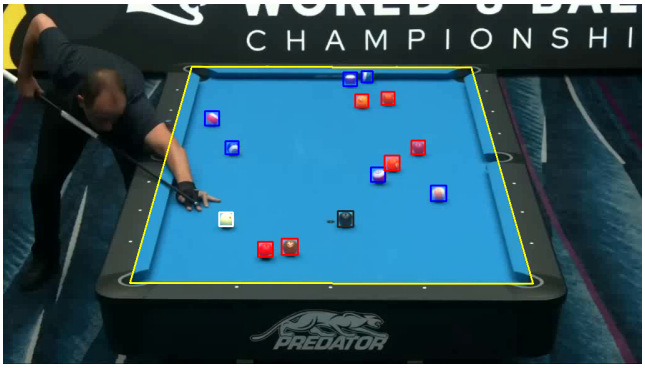


(d) Segmentation last frame

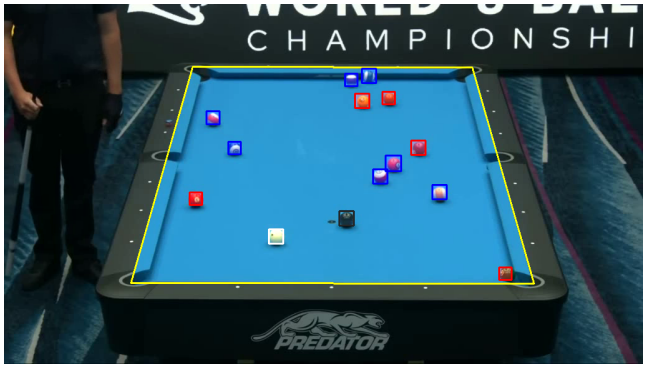


(e) Trajectory last frame

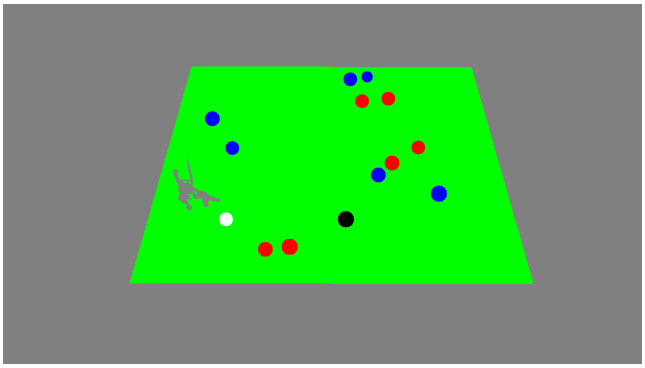
Figure 14: Game 1 Clip 3



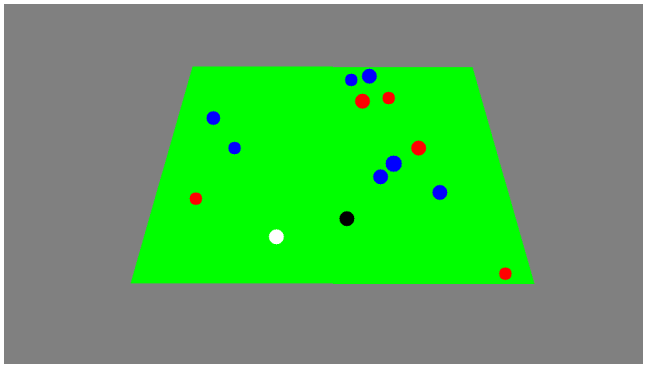
(a) Detection and Classification first



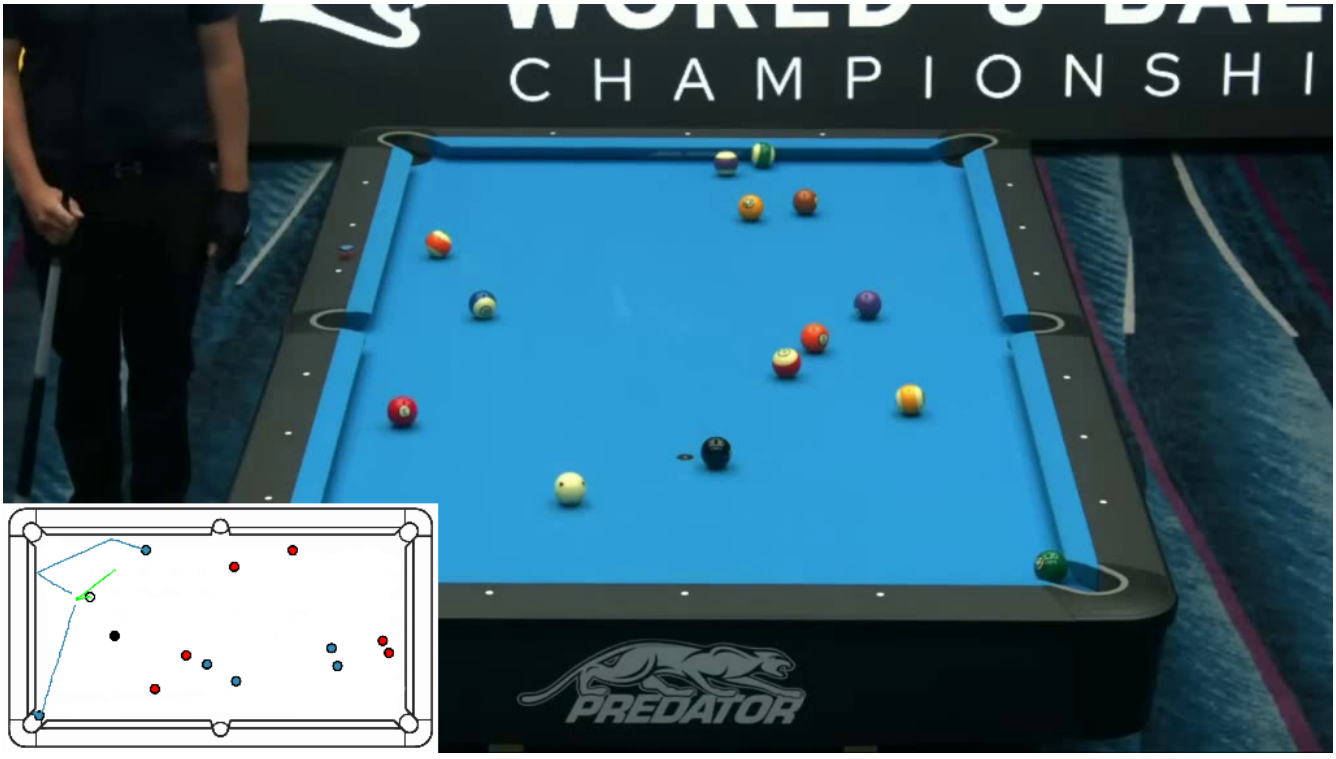
(b) Detection and Classification last



(c) Segmentation first frame

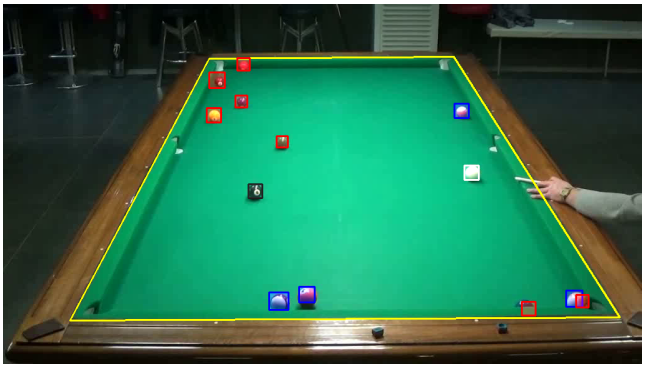


(d) Segmentation last frame

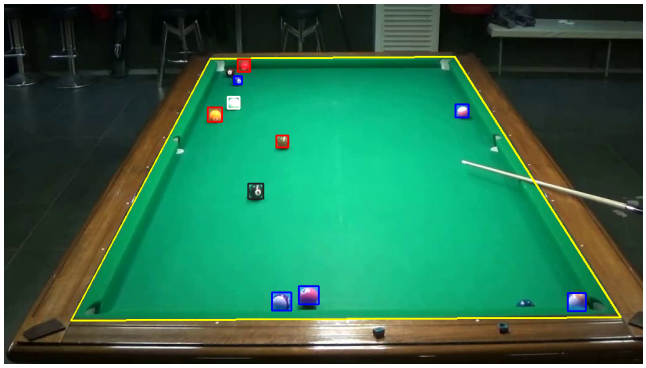


(e) Trajectory last frame

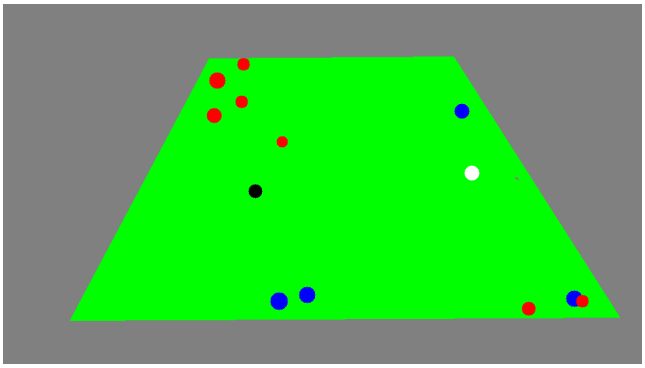
Figure 15: Game 1 Clip 4



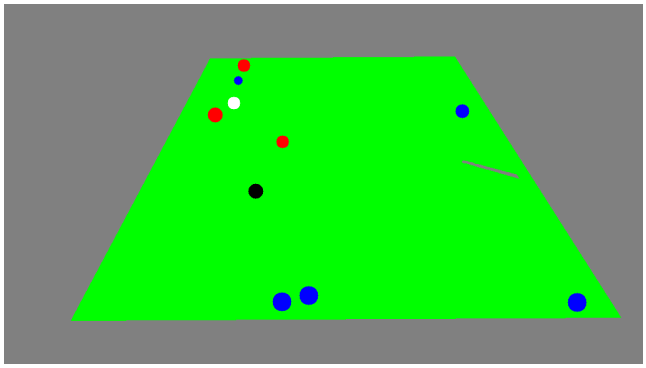
(a) Detection and Classification first



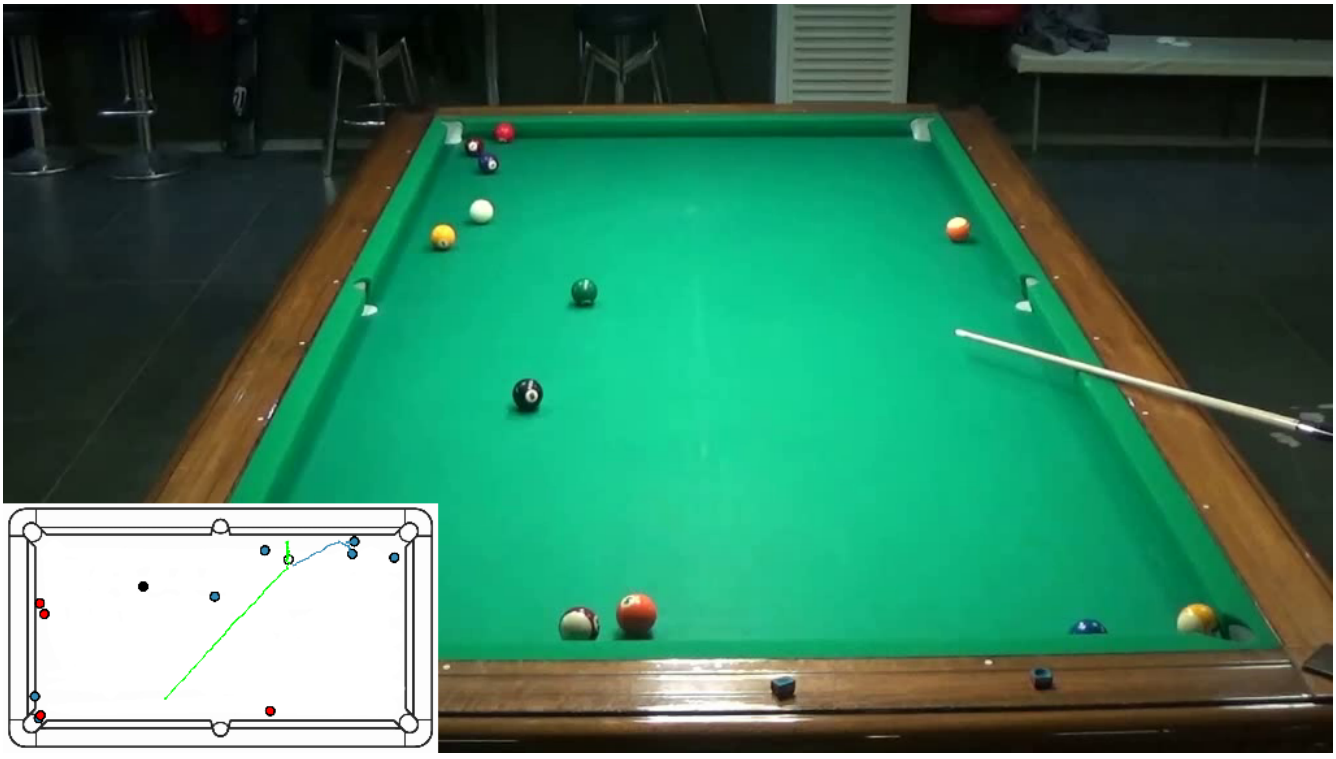
(b) Detection and Classification last



(c) Segmentation first frame

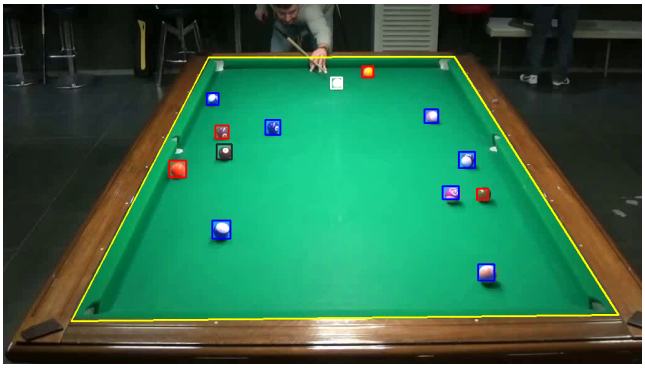


(d) Segmentation last frame

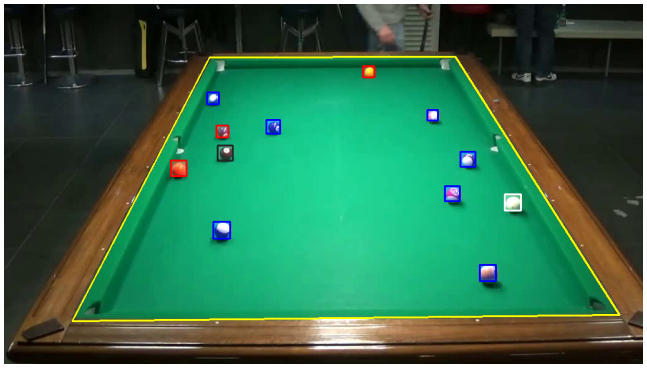


(e) Trajectory last frame

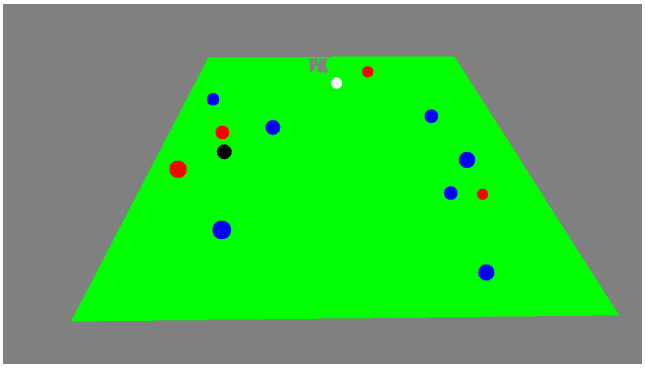
Figure 16: Game 2 Clip 1



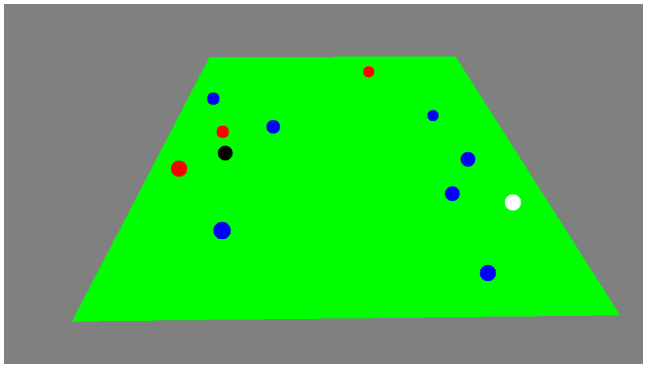
(a) Detection and Classification first



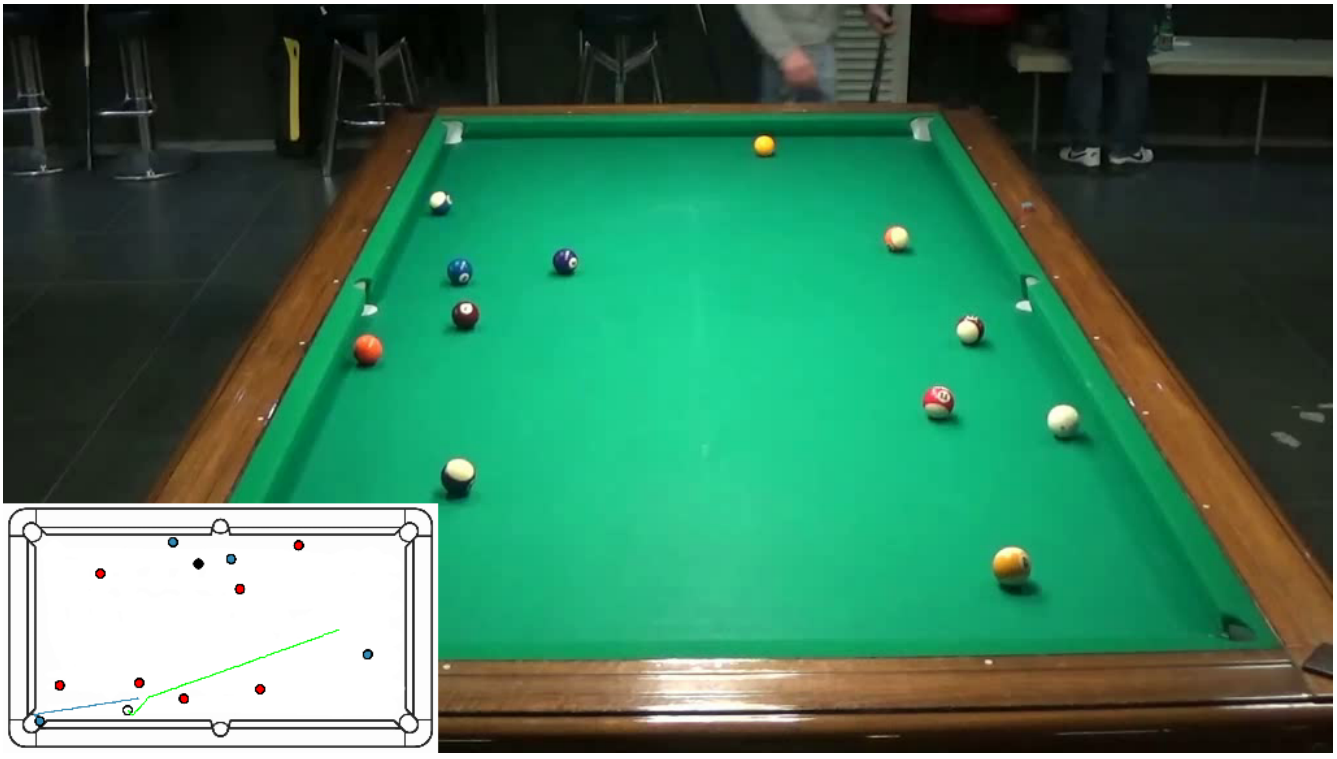
(b) Detection and Classification last



(c) Segmentation first frame

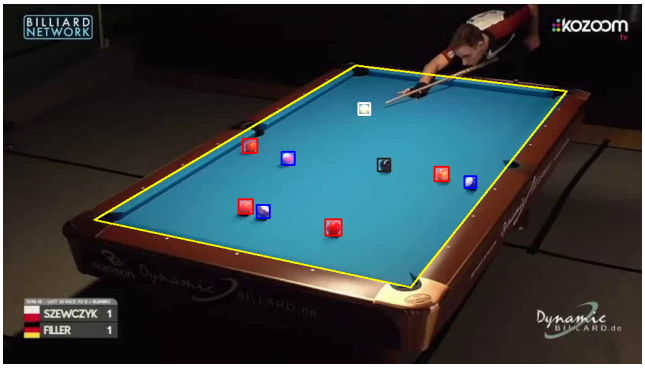


(d) Segmentation last frame

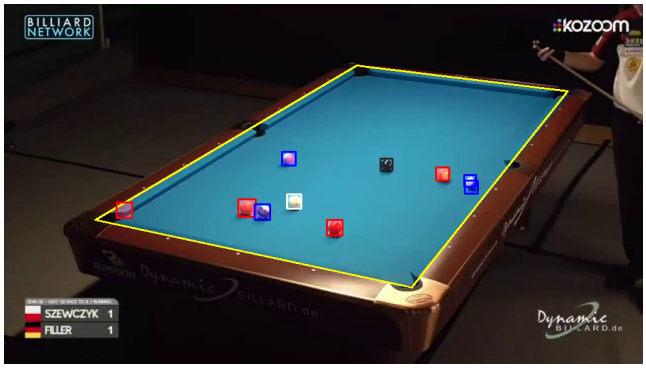


(e) Trajectory last frame

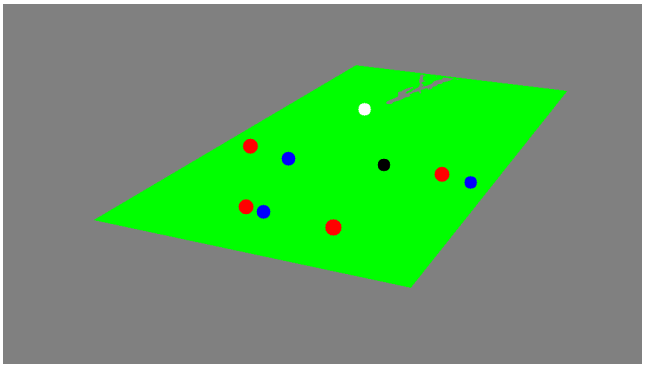
Figure 17: Game 2 Clip 2



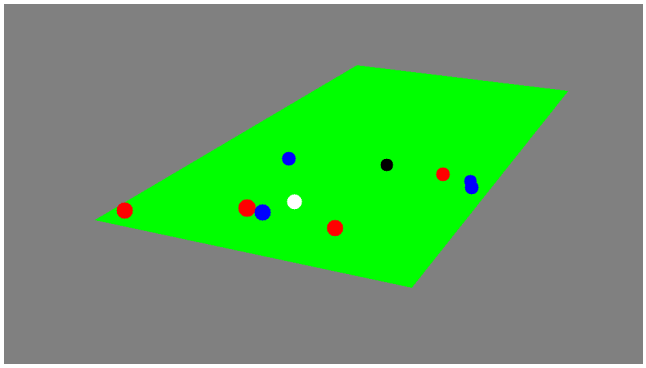
(a) Detection and Classification first



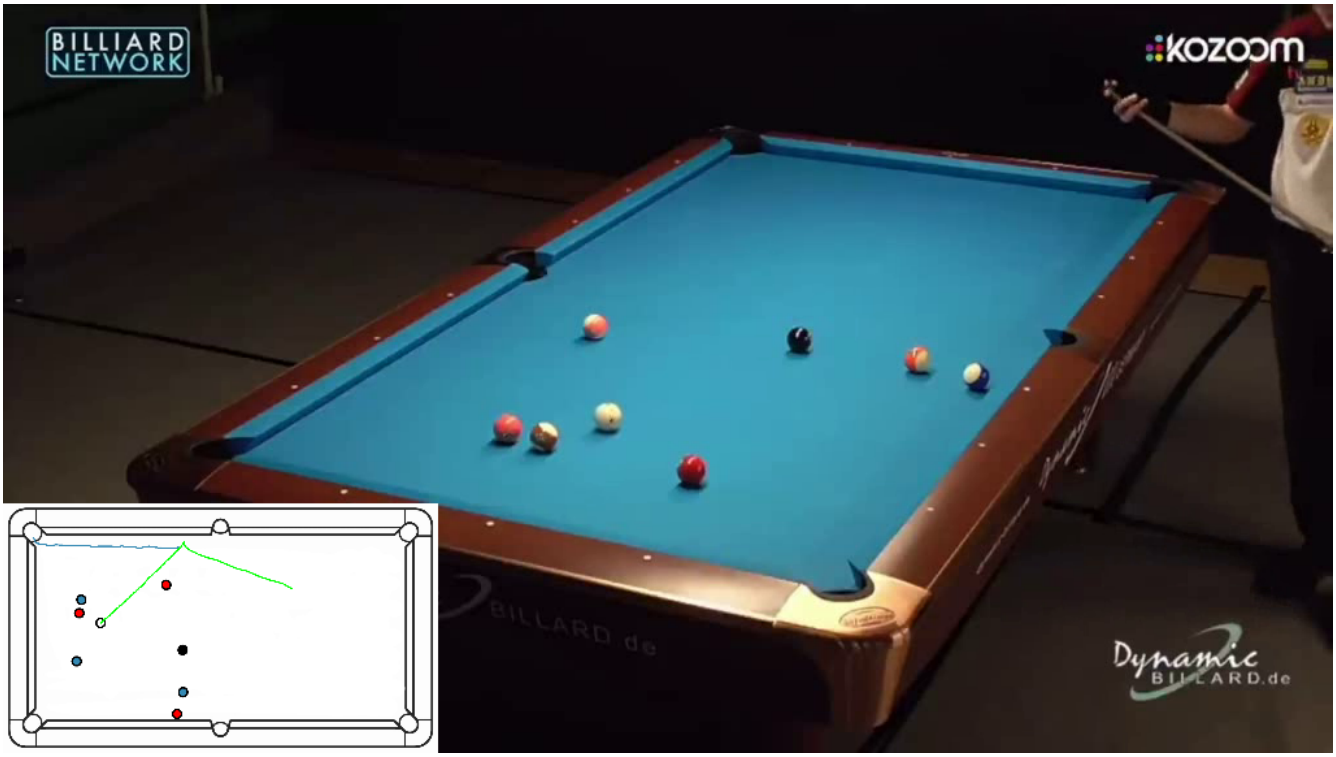
(b) Detection and Classification last



(c) Segmentation first frame

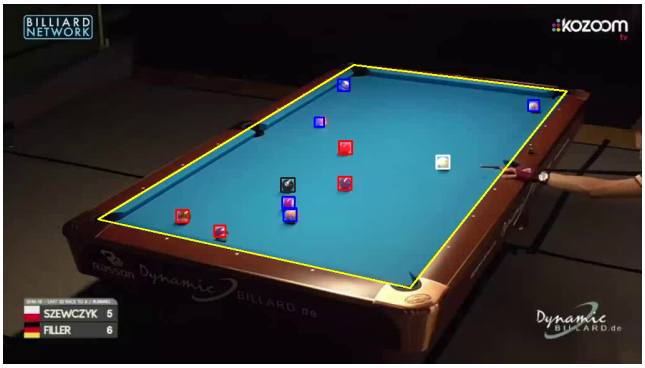


(d) Segmentation last frame

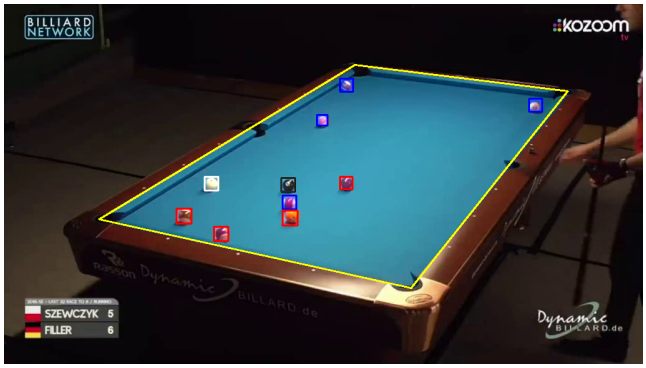


(e) Trajectory last frame

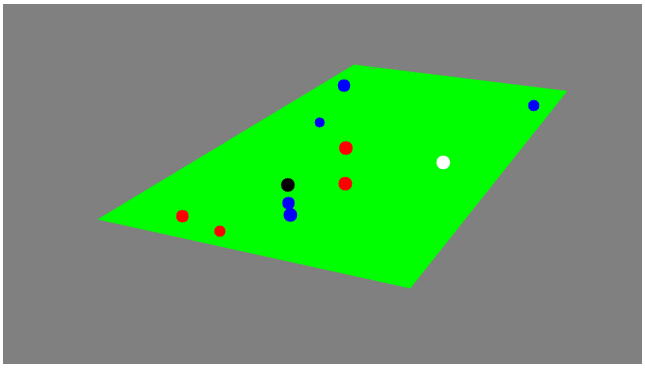
Figure 18: Game 3 Clip 1



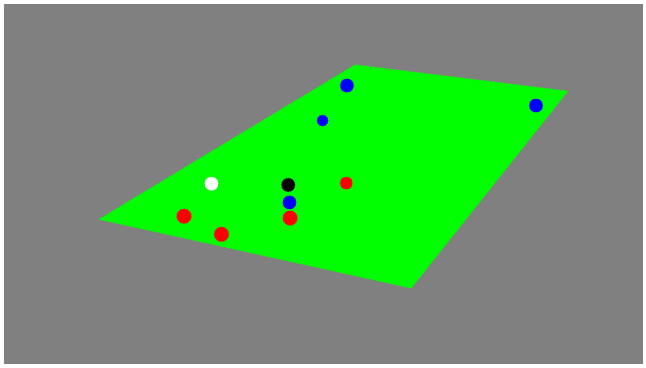
(a) Detection and Classification first



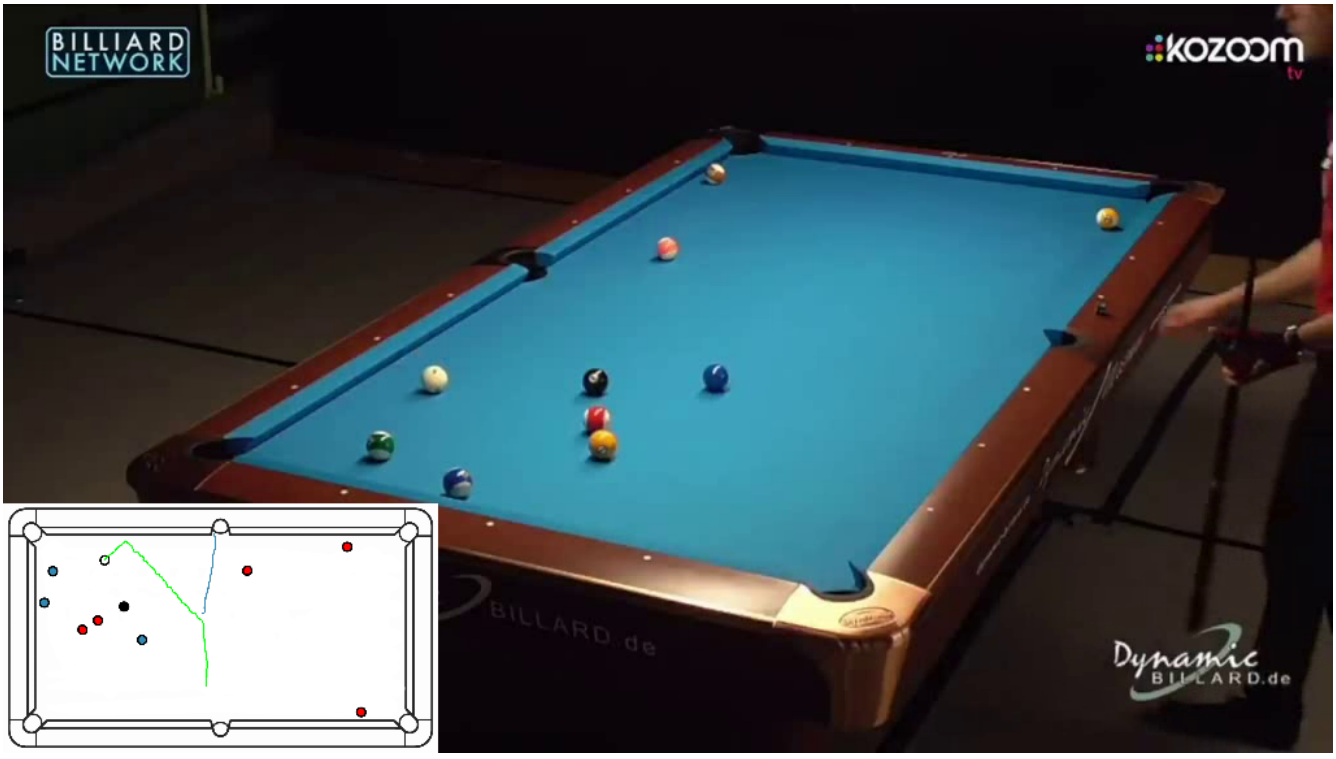
(b) Detection and Classification last



(c) Segmentation first frame

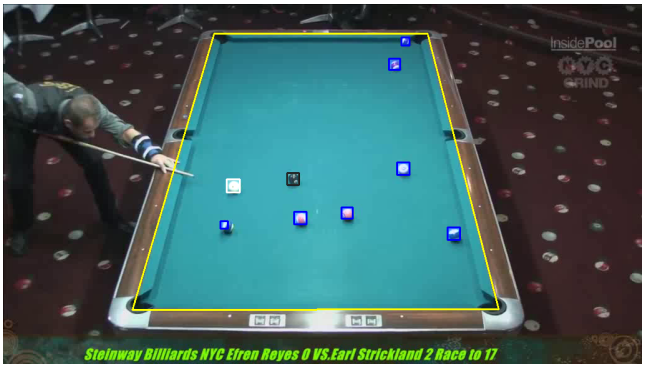


(d) Segmentation last frame

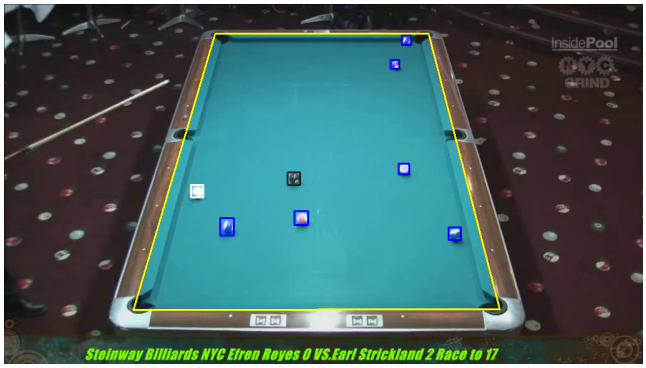


(e) Trajectory last frame

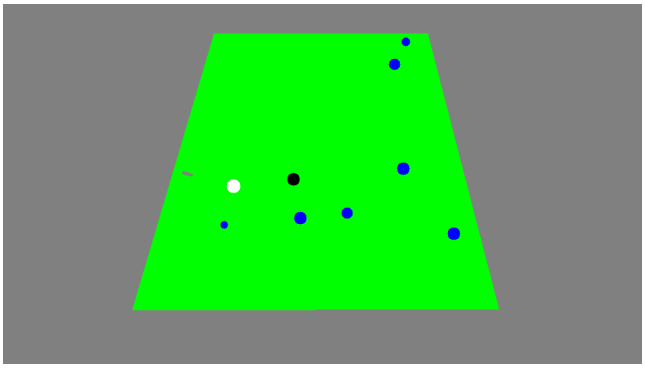
Figure 19: Game 3 Clip 2



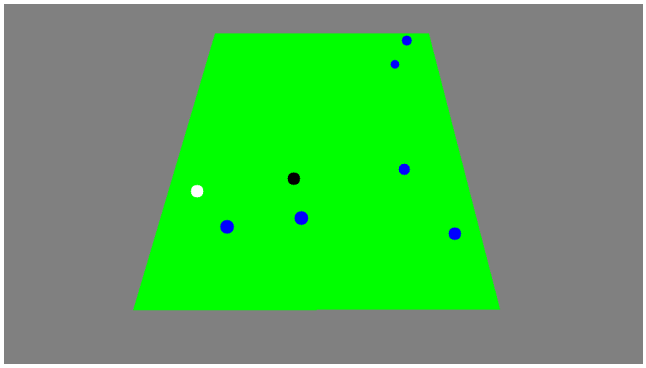
(a) Detection and Classification first



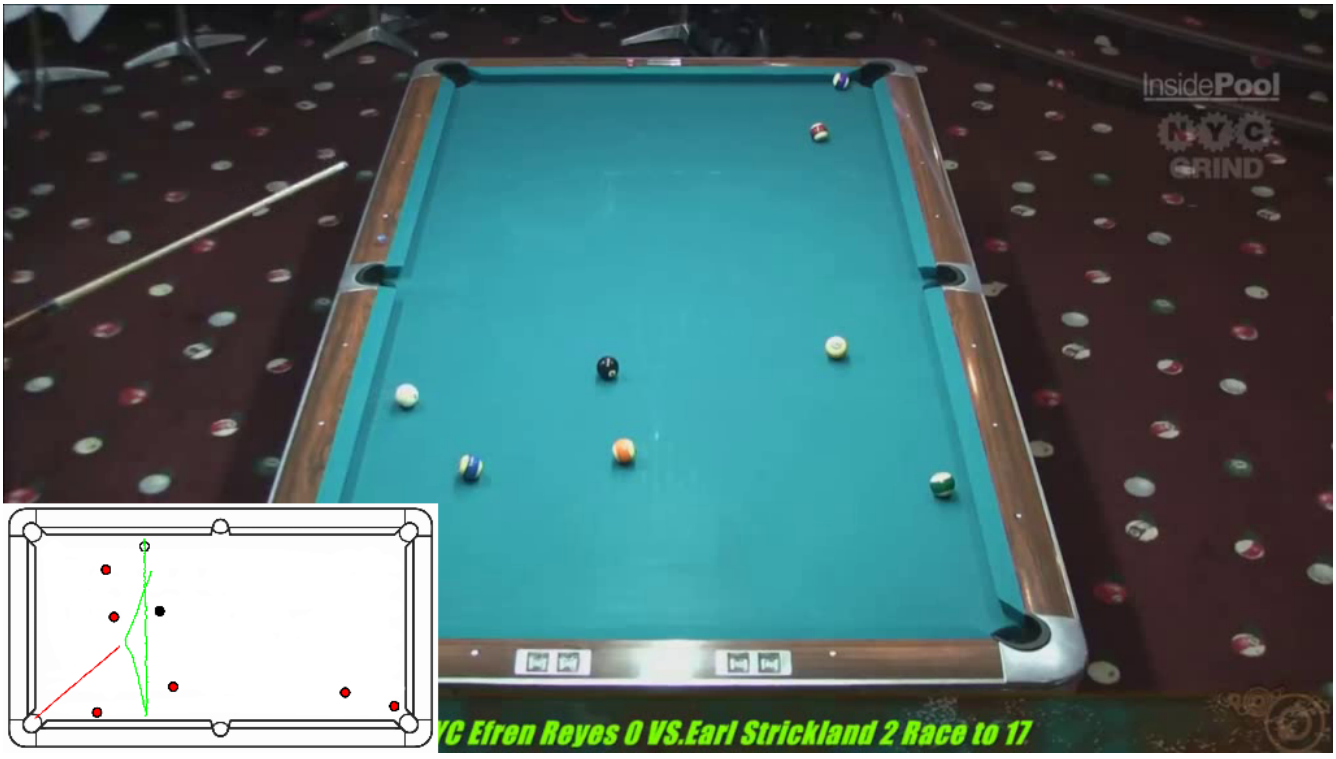
(b) Detection and Classification last



(c) Segmentation first frame

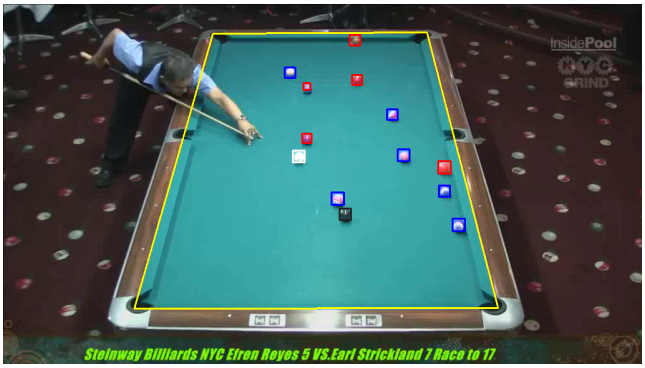


(d) Segmentation last frame

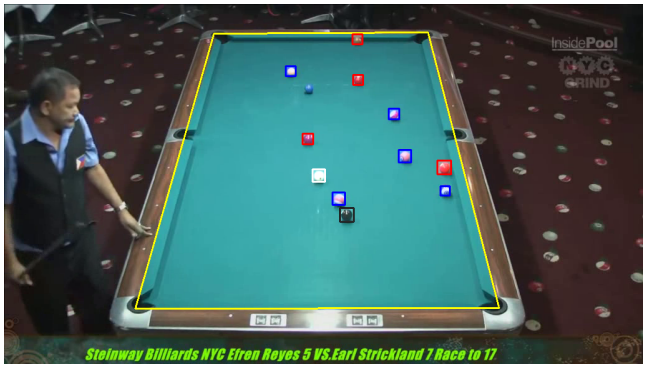


(e) Trajectory last frame

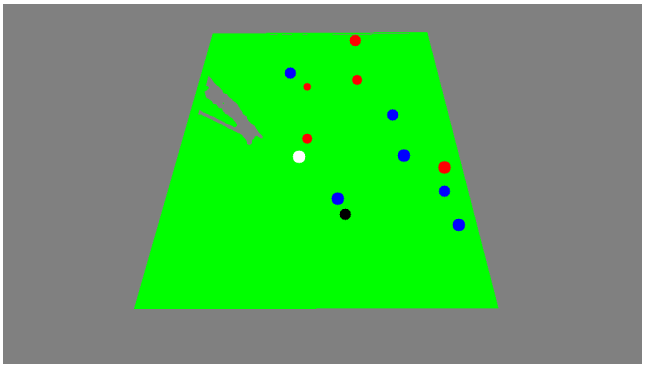
Figure 20: Game 4 Clip 1



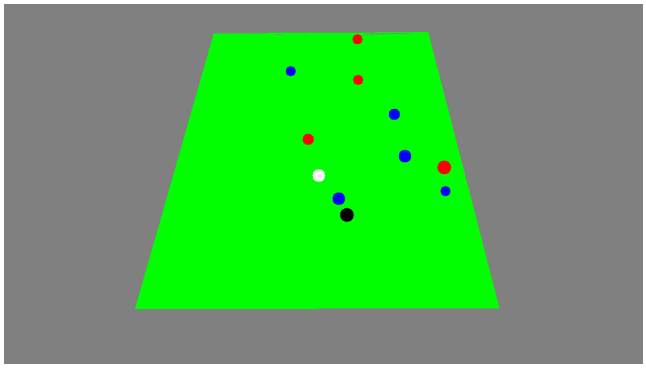
(a) Detection and Classification first



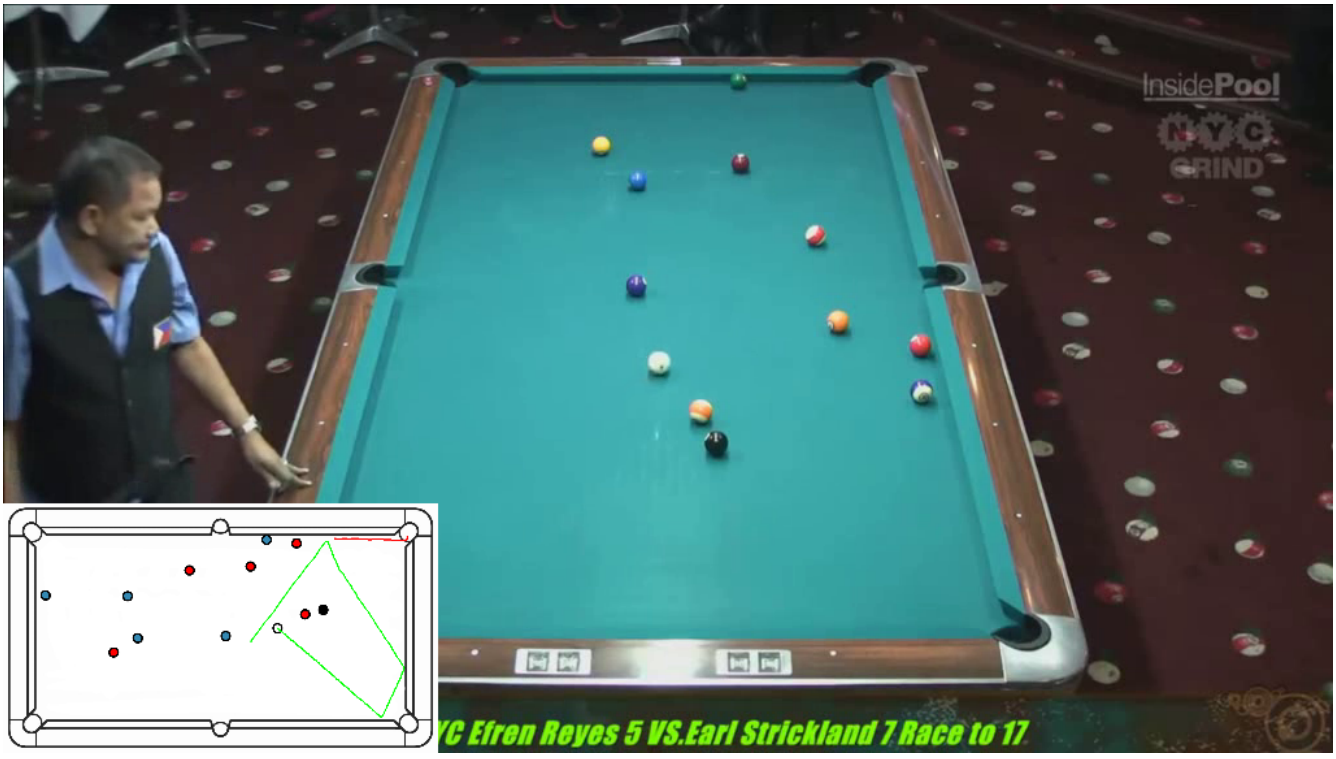
(b) Detection and Classification last



(c) Segmentation first frame



(d) Segmentation last frame



(e) Trajectory last frame

Figure 21: Game 4 Clip 2

Performance measures

Below are represented two tables that show the final performance results both on the whole dataset and on each videoclip:

Class	**mIoU**	**AP**
Background	0.98	-
White ball	0.75	1
Black ball	0.75	0.89
Solid ball	0.52	0.77
Striped ball	0.61	0.85
Playing field	0.96	-
Mean over all classes	0.76	0.88

Table 1: Performance results on the whole dataset, for each single class

Clip	**mIoU**	**mAP**
Game 1 Clip 1	0.80	0.77
Game 1 Clip 2	0.78	0.77
Game 1 Clip 3	0.83	0.79
Game 1 Clip 4	0.82	0.77
Game 2 Clip 1	0.74	0.77
Game 2 Clip 2	0.70	0.77
Game 3 Clip 1	0.76	0.79
Game 3 Clip 2	0.72	0.64
Game 4 Clip 1	0.68	0.54
Game 4 Clip 2	0.74	0.68

Table 2: Performance results on single clip

Even if the computation of averaged results across all the classes are used to overcome the imbalance on the size of the different classes, it can be noticed how the overall results are lowered by the performances on classification solid and striped balls. These values underline the difficulty of the classification task for those balls, and indicates that there is room for improvement on this part. Besides that, it can be said that in some cases the classification of the balls is difficult even by the human eye, for example in the *game2_clip2* as the following image shows:

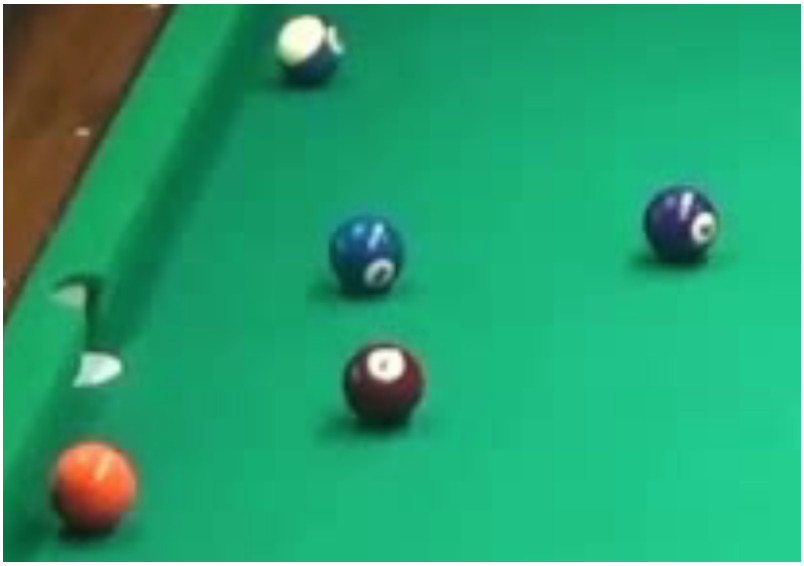


Figure 22: Misclassification of black ball in game 2 clip 2

Indeed, our approach was to first classify a white and a black ball with a quite high confidence, but it was difficult to find a black ball in this clip. Due to this uncertainties we checked the ground truth segmentation and the .txt of the classified boxes, but this didn't give us any further

confidence since there is some discrepancy also there.
Furthermore, looking at the results on the single images, it can be assessed that the system yields better results on the first clips, as they seem visibly clearer, while the performances decay when analyzing the last clips that show more noise and light reflections. This suggest that robustness to different light conditions must be enforced. Regarding the detection task, it can be seen that the general performance is quite satisfying, even though there are some issues, for example in *game*1_*clip*2, where the ball on the lower part of the table that is almost totally covered is detected only on one of the two frames, while the brown ball that has been hit by the player, is not detected in the last frame due to the long distance from the camera and the fact that the image is darker in that area. In this clip another problem has been encountered, as one ball in the lower right part of the table is detected twice.

Possible improvements

Table detection

Besides the good results in the detection of the table on all the frames, all the code could be improved to gain more efficiency. Regarding the procedure, some work could be done to obtain more robustness against different luminosity conditions, relying on operations on different color spaces like HSV or HSL, which give more intuitive information on brightness of the image.

Balls Detection and Hands Segmentation

There are some functions and processes that could have been implemented better from the start. First of all, the hand segmentation should have been done on the original image, not on the modified one with blocks hiding the holes. The region of interest could have been the table corners area instead of the playable field area. This approach would have allowed more of the arm to be detected and not considered part of the table in the segmentation process. Concerning ball detection, a more uniform and unique filter could have been implemented. As can be seen in Fig.7(e) and Fig.15(c).

State representation Tracking

For the tracking part, a possible improvement could be noise reduction and suppression to address the oscillation of moving balls in the 2D top-view scheme. Additionally, improving the efficiency of nested loops could be another improvement.

Miscellanea

Notes about Project Filesystem and Code

The project folder structure has been created using CMake, organizing code files into *include* and *src* directories. We have generated a source code and header file containing all the functions related to each phase of the pipeline and also a `utils.cpp` source file for the implementation of general purpose functions. The executable is generated in the *build* folder.

We have produced two executable files:

- *main*: Manages the generation of the output video, images, and the whole processing.
- *performance*: Used for the computation of performance measures.

Notice that the *main* program can be executed in two ways:

- Without any arguments: Produces the output images and video for all games in the dataset.
- With a specific video clip path as an argument: Processes only the specified video.

The *performance* program should be executed without any arguments, providing the requested metrics for the entire dataset. For detailed instructions on running the code, please refer to the README.md file in the project folder.

In the *data* folder, there is an *eight_ball_table* directory containing a .jpg file of the table minimap and a .txt file with the pixel coordinates of the table corners. These information is useful for computing the homography and will be retrieved by the homography functions.
In the project folder, there is also an *Additional_material* folder containing the created dataset for the Haar cascade, which was not the final adopted approach. This folder includes an explanation of how it was created and related videos and images.

The project was initially divided into three primary tasks: homography and ball tracking, ball detection and classification, and table detection, with each task developed independently. During the integration phase, several issues arose concerning the compatibility between the inputs and outputs of various functions. It became apparent that the efficiency of function concatenation and overall code structure could be improved.
During the execution of the `main` some warnings raised when showing images, but should not affect the execution of the programs.
Each file has the author's name at the top, usually representing the author's ideas. Additional considerations and approaches were discussed but were straightforward ideas that all group members contributed. For development and version tracking, we have used GitHub, with each task developed in a separate branch.

Number of hours worked

- **Marco Panizzo:**
  - Group meetings together (ideas, updates, discussions): 5h;
  - Tracking: scheme, hand labeling of scheme and dataset corner: 1.5h;
  - Tracking: Homography part (basic and best homography): 5h;
  - Tracking: Video handling, scheme over imposed: 5h;
  - Tracking: Tracker, balls and trajectories drawing: 10h;
  - Tracking: Speed up the existing code (static and moving balls): 3.5h;
  - Tracking: HSV preprocessing: 1.5h;
  - Tracking: overall restructuring of the code (subfunctions): 7.5h;
  - Tracking: comments: 5h;
  - Dataset: creation of complete dataset for Haar Cascade (not used): 15h;
  - Final Ball detection: Test of Sobel and Laplacian (some tuning was done): 3h;
  - Final Ball detection: Test of Hough circles on Sobel (with tuning): 4h;
  - Final Ball detection: Channels selection and thresholding (and tests): 5h;
  - Final Ball detection: Tuning: 15h;
  - Final Ball detection: Functions to suppress false positives: 20h;
  - Final Ball detection: Tuning (but considering the false positive suppression): 10h;
  - Final Ball detection: Comments: 6h;
  - Final Ball detection: Merging with table detection and homography (for testing): 4h;
  - Segmentation: 4h;
  - Group final adjustments and report together: 20h.
- **Matteo Dal Nevo:**
  - Group meetings together (ideas, updates, discussions): 5h;
  - First Ball detection: Test of Hough circles: 2h;
  - First Ball detection: Creation of the ROI of the detected balls: 2h;
  - First Ball detection: Study of Haar Cascade Classifier: 4h;
  - First Ball detection: Creation of a small dataset for Haar Cascade Classifier: 4h;
  - First Ball detection: Test of Hough circles with more specific parameters tuning: 2h;
  - First Ball detection: Table segmentation using a k-means + threshold approaches: 4h.
  - First Ball detection: Table segmentation test with different color space (HSV): 4h.

  - Table detection: Table segmentation based on previous ball detection part: 2h;
  - Table detection: Tuning for the table segmentation using morphological operator: 2 h;
  - Table detection: Lines detection and tuning: 2h;
  - Ball classification: Data analysis on the white pixels percentage: 10h;
  - Ball classification: Binary threshold tuning for classification: 10h;
  - Ball classification: Color masking approach + threshold tuning: 10h;
  - Ball classification: Creating a cascade process for classification: 10h;
  - Code cleaning: 3h;
  - Comments on code: 4h;
  - Group final adjustment and report together: 20h.
- **Aaron Prevedello:**
  - Group meetings together (ideas, updates, discussions): 5h;
  - Table detection: Segmentation algorithms: 6h;
  - Table detection: Grabcut algorithm: 5h;
  - Table detection: Lines manipulation: 8h;
  - Table detection Hough Lines: 7h;
  - Performance: mIoU: 3h;
  - Performance: mAP: 15h;
  - Performance on the whole dataset: 5h;
  - Ball classification: 6h;
  - Code merging and compatibility issues: 6h;
  - Compatibility of the code with the virtual lab: 2h;
  - Comment on code: 4h;
  - Group final adjustment and report together: 20h.

	Marco Panizzo	Aaron Prevedello	Matteo Dal Nevo
Time [h]	$\sim$ 150h	$\sim$ 90h	$\sim$ 100h