Computer Vision Final Project

Sport video analysis for billiard matches

Fresco Eleonora, Girardello Sofia, Morselli Alberto

June-July 2024

1 Introduction

1.1 Project Assignment

The goal of this project is to develop a computer vision system for analyzing video footage of "8-Ball" billiard games. This vision system aims to provide detailed, high-level information about the status of the match, including the positions and trajectories of the balls. These trajectories will be displayed in real-time through a 2D top-view minimap, superimposed on the bottom-left corner of each video frame.

1.2 Requested Tasks

The objectives of this project can be achieved dividing the project into several key tasks:

Playing Field Detection

This task requires detecting the lines that define the billiard table for the first frame. Accurate detection of these boundaries is essential for subsequent analysis, as it establishes the area of interest for all further processing.

Ball Detection

This task focuses on isolating potential ball candidates on the table for every frame of the video. The system has to identify all the balls on the table, focusing on their detection and localization.

Segmentation and Classification of Playing Field Elements

This task involves segmenting and classifying each pixel within the table region into one of the five categories: the cue ball, the 8-ball, solid balls, striped balls, and the playing field itself.

2D Top-View Visualization

This task aims to create a dynamic and real-time graphical representation of the billiard table from a top-view perspective. The system will continuously update a minimap that shows the current positions and trajectories of all balls on the table.

Metrics Evaluation

To evaluate the performance of the developed system for analyzing billiard matches, two metrics are employed:

- Mean Average Precision (mAP): This metric is used for evaluating the ball localization accuracy of the system. It measures how well the system can detect and localize each ball type within the playing field.
- Mean Intersection over Union (mIoU): This metric assesses the segmentation performance of the system. The mIoU is computed as the average IoU across all classes of interest: background, white cue ball, black 8-ball, solid-colored balls, striped balls, and the playing field.

2 Methodologies

2.1 Approach

Our first focus was on the project structure and code hierarchy. We tried to subdivide the code into modules, each one testable on its own and interfacable with the others in a well-defined way decided a priori, before writing any line of code. Since the work proceed not sequentially, but in parallel due to time-contraints, we focused on testing each module using groundtruth matter and easy tricks (e.g. corrupting the groundtruth segmentation masks with a well-defined shape of known pixels to test metrics before having finished detection). The code can either process just first and last frame, as requested, or even every frame for debug purposes if desired.

2.2 System Design

Our computer vision system processes video input through a structured pipeline to accurately detect and track objects within the scene, as depicted in Figure 1.

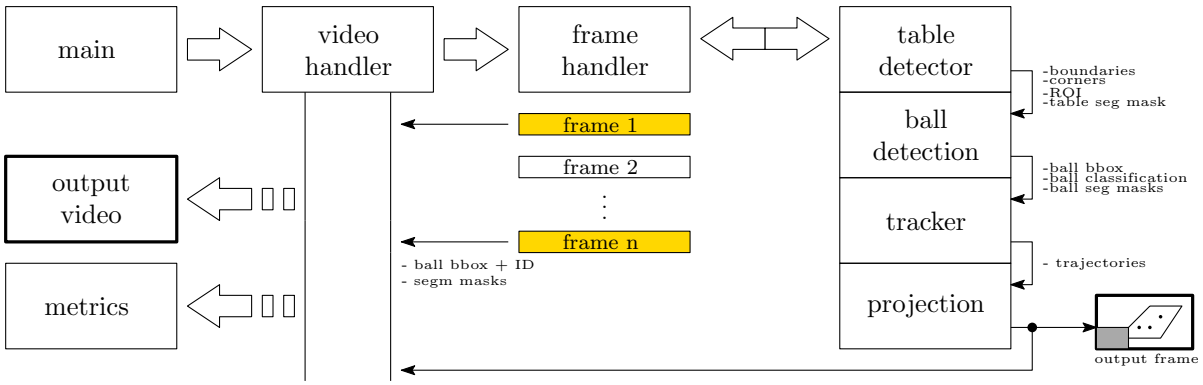


Figure 1: Block-scheme of Project Modules

Initially, the *video handler* receives the raw video feed, which is then divided into individual frames, each one passed to the *frame handler* to be further sequentially processed.

Within the frame handler, for every frame to be processed (first and last or all), the first step involves the *table detector*, which identifies and delineates the table's boundaries, corners, region of interest (ROI) and provides the table segmentation mask. Next, the *ball detector* module can use the ROI to restrict the searching area to identify and classify the balls, providing bounding box data, classification results and ball segmentation mask. Moreover for the first frame, the system initializes *trackers* for each detected ball.

As the video progresses, the tracker updates the position of the balls in each frame, ensuring continuously trajectory updating. Additionally, the trajectory *projector* calculates and overlays the trajectories of the balls onto the frames. Specifically for every processed frame the system superimposes a minimap of the computed 2D top-view which represents the current state of the game.

Finally, the processed frames are returned to the *video handler*, which assembles the new video with the elaborated frames and after computes the metrics.

3 Implementation Details

3.1 VideoHandler and FrameHandler

VideoHandler and *FrameHandler* are fundamental for all the processing and managing of the video data within the system.

The *VideoHandler* class is responsible for managing the entire video processing pipeline. It initializes by loading ground truth data files of segmentation masks

and bounding box annotations which will later be used for metrics computation. The class provides a method, `process_video`, which orchestrates the extraction and processing of frames from a video file, employing the *FrameHandler* class to perform detailed operations on each frame. Specifically, *VideoHandler* controls the flow of processing, including detecting, tracking objects and projection. It displays the intermediate result outputs based on a flag passed on command line by the user, and additionally saves the processed frames into a new reconstructed video file. Moreover, the *VideoHandler* computes and outputs performance metrics: mean Average Precision (mAP) and mean Intersection over Union (mIoU), to evaluate the accuracy of object detection and segmentation.

FrameHandler class focuses on frame-level operations and coordinates with multiple components to process each video frame. It's an encapsulating class that owns all the instruments (table, detection, tracker and projector) to coordinate the processing on each frame. Its purpose is to ensure a structured and encapsulated method to process each frame, without accessing the low level details of each of his "tools".

3.2 Field Detection

The field detection process is a critical component of the system, designed to identify the billiard table within a given frame. (From here below we use consider "playing field" the area inside lines belonging to the upper parts of the top rails of the table, that is the parts where the cloth covering the playing field surface is attached to the wooden structure of the table). The process of detection is executed through a series of well-defined steps implemented in the *tableDetector* class. Below is a detailed description of each step involved.

1. **Extracting the Dominant Color:** This is accomplished by converting the input image to HSV. The `get_dominant_color` function is used to compute the hue histogram to determine the dominant one, i.e: the table field color.

2. **Thresholding:** with the dominant color identified, the `treshold_mask` function generates a binary mask that isolates the part of the image having that chromatic dominant from the rest. This is achieved by applying a color threshold in the HSV space, with a bit of hand tuning around the dominant hue to highlight effectively the table area for all the provided dataset. The result is visible in Fig.(2a).

3. **Finding the Largest Connected Component:** the next step isolates the table from the binary mask. The `find_largest_comp` function performs morphological closing to reconnect internal and external raised cloth edges of

the table and then identify the biggest connected component that is assumed to be the table. The result is visible in Fig.(2b).

4. **Detecting the Contour:** once the largest connected component is identified, the `find_contour` function extracts the contour of this area, used in outlining the boundary of the ROI excluding occlusions like humans and billiard rods.

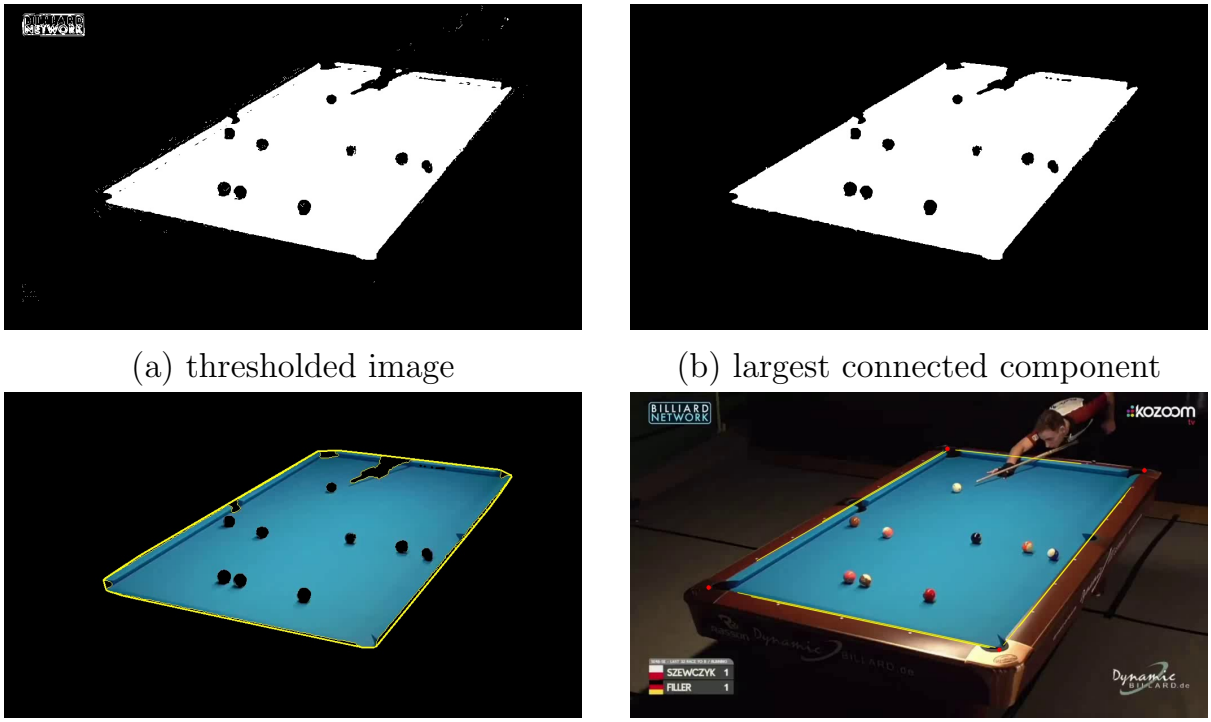
5. **Generating the Segmentation Mask:** is simply done by filling the detected contour.

6. **Computing the Convex Hull:** Since we also want to identify the edges of the table, to address potential occlusions and irregularities in the contour, the `get_hull` function computes the convex hull around the detected contour, providing a more accurate representation of the table's shape. Both internal contour and external convex-hull can be seen highlighted in yellow in Fig.(2c).

7. **HoughLines and corner detection:** Having the convex-hull makes it now easy to apply `houghLines` to get the piecewise lines of the convex hull. Since there's a projection phenomena in the game and camera non-idealities they can (even with perfect code) sometime be slightly curved. By a coarse tuning of houghlines parameters and a computation of all the possible intersections of the detected lines it's possible to discriminate the four corners of the table, represented by 4 red points as can be seen in Fig.(2d).

8. **Drawing Borders:** as last step, to visualize the detected table, the `draw_borders` function overlays the external table contour on a given frame.

Note that the assumption that leads to consider the table as fixed throughout all the frames from the same video, makes it possible to do the table detection process only for the first frame and then maintaining the extracted contours and corners for all subsequent processing. It is however possible to re-identify the borders and corners for each frame analyzing all the frames instead of just first and last ones.



(a) thresholded image

(b) largest connected component

(c) contour and convex-hull

(d) houghlines and corners

Figure 2: Field Detection algorithm on game3_clip1

3.3 Ball Localization

The *ballDetector* class encapsulates the ball detection functionality and, subsequently, their classification. The primary method in this class is `detectBalls`, which performs the detection, localization and classification of balls in the given frame. A detailed description of each step involved in the detection procedure is now presented.

1. **Masking with the ROI:** the Region of Interest (ROI) is applied to the current frame to focus only on the pool table area , thereby eliminating any extraneous background information that could interfere with the detection process. The resulting image is then passed to `applyColourDetection` method for further analysis of the pooling table.

2. **Enhancing Contrast:** to improve the visibility of the balls, the `enhanceContrast` function is employed. This function firstly converts the masked image to the LAB color space and separates its channels. Then the L-channel, which represents lightness, undergoes Contrast Limited Adaptive Histogram Equalization (CLAHE), which enhances the local contrast of the image, making the balls stand out more clearly. Finally the overall image is converted back to the BGR color space for further processing. The result of this contrast enhancement is reported in Fig.(3a).

3. **Calculating Average Color:** within the `averageColourThresholding` function, again called by `applyColourDetection` ,the average color of a small central area of the frame is calculated aiming to estimate the color of the playing field, which helps in distinguishing the balls from it. The average color is converted to the HSV color space, which is more suitable for color-based thresholding. Although this process might seem similar to the color detection used in the *tableDetector* class for identifying the pool table, there is a crucial difference. The previous thresholding method, which relied on the most frequent hue value in the image, did not yield satisfactory results for all the cases. Consequently, an alternative color thresholding approach, employing a different color analysis method, has been implemented to improve accuracy.

4. **Color-Based Thresholding:** a thresholding based on the average center color previously found is done to isolate potential balls. This is again handled by the `averageColourThresholding` function, that returns a mask of the type reported in Fig.(3b).

5. **Detecting Balls:** to detect the balls, the Hough Circle Transform is employed due to its effectiveness in identifying circular shapes. This method is applied to the binary mask obtained as output from the thresholding, which serve as potential candidates for the balls. The parameters of the Hough Circle Transform have been meticulously hand-tuned on the dataset to optimize performance and achieve the best possible results, as shown in Fig.(3c).

6. **Validating Detected Circles:** each detected circle undergoes further analysis to ensure it accurately represents a ball. The `selectBalls` function filters out the acceptable balls using this criteria: if a circle sufficiently overlaps with the ROI and the inverse threshold mask, it is confirmed as a detected ball. Looking back at this last two steps, it is clear that a dual approach based on both the shape of the balls (Hough Transform) and their color (colour threshold mask) is necessary to obtain the best performance out of the project.

7. **Saving Detected Balls:** the validated balls are then saved using the `saveInfo` function, which stores each ball's position and bounding box. In particular two types of bounding boxes are stored: the first one representing the accurate ball dimensions used for localization of the balls, and the second one representing a wider area around the balls used for tracking their trajectories.

3.4 Classification and Segmentation

The procedures of segmenting and classifying the balls are integrated into the same ball detection process, facilitating a smooth transition from detection to

classification. This step is vital for distinguishing between different types of balls on the pool table. The process involves the following steps:

1. **Extracting the Ball ROI:** for each detected ball, a circular mask is created to isolate the ball from the rest of the frame. This mask is used to extract the Region of Interest (ROI) corresponding to the detected ball.

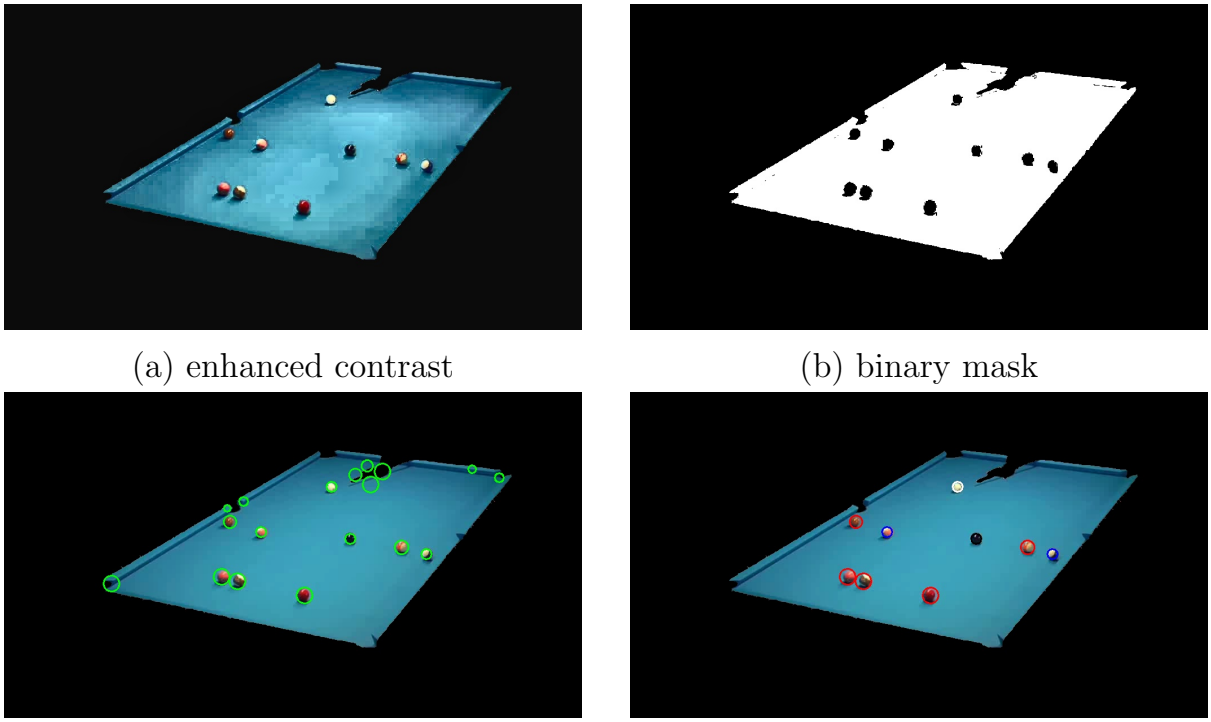
2. **Converting to Grayscale:** the `analyzeBallPattern` function converts the frame to grayscale in order to desaturate the image and simplify the analysis by reducing the image to a single channel.

3. **Applying Binary Thresholding:** Two binary thresholds are applied to the grayscale image to create binary representations of the balls. The first threshold masks the dark pixels, while the second masks the light ones. This step is crucial for distinguishing potential stripes on the balls and accurately identifying the white cue ball and the black 8-ball.

4. **Creating a Circular Mask:** the circular mask created previously is applied on the binary images to ensure that only the relevant part of the frame related to each ball is analyzed, excluding any background or noise.

5. **Calculating White Pixel Percentage:** the percentage of white pixels and black pixels within the circular mask area is calculated in order to determine if the ball is striped, solid, black or white.

6. **Classifying the Ball:** based on the calculated white pixel percentage and black pixel percentage, the ball is classified by the `classifyBalls` method. Just the balls with the maximum percentage of light or dark area are classified as the white or black balls. On the rest of them a distinction between striped or solid balls is developed by making use of a threshold on the percentage of white pixels. The balls are then highlighted with different colors depending on their class, as shown in Fig.(3d).



(a) enhanced contrast

(b) binary mask

(c) hough circles

(d) detected and classified balls

Figure 3: Balls Detection and Classification algorithm on game3_clip1

NOTE: An additional method is present in the *ballDetector* class, but it is not utilized: `detectBallsFinalFrame`. This method was initially designed to use the tracked centers, along with the detected balls, to provide more accurate detection and classification for the last frame. However, the results were found to be less accurate compared to the existing approach, and thus it was not adopted.

3.5 Trajectory Tracking

The trajectory tracking component is fundamental for monitoring the movement of objects through video sequences. This is accomplished by the series of two fundamental steps, encapsulated within the *trajectoryTracker* class.

1. **Initializing Trackers:** the process begins with the `initializeTrackers` method, which establishes the framework for tracking objects within the video frames. Initially, the method sets the parameters for the CSRT (Channel and Spatial Reliability Tracker). This involves configuring various settings, such as the use of HOG (Histogram of Oriented Gradients), color, grayscale information, and different window functions and learning rates. These configurations are essential for the tracker to handle several tracking challenges and object appearances effectively. In particular, it was necessary to higher the filter and weights learning rates in order to be able to keep up with the fastest moving balls.

After the configuration part, the function creates and initializes a tracker for each object based on the initial provided bounding boxes which specify the initial positions in the first frame of the video, as shown in the first frame of Fig.(4). These trackers are added to a list for ongoing updates, and a separate list is prepared to keep track of each object's trajectory over time. It is worth underlying a fact already cited above: the bounding boxes used for tracking the balls need to be bigger than the actual bounding dimensions of the balls, in order to give the tracker a sufficiently big area to perform their detection. This causes a slight imprecision on the projected trajectories.

2. **Updating Trackers:** as the video progresses, the `updateTrackers` function updates each tracker with the latest frame, determining the new positions of the tracked objects, as shown in the sequence of Fig.(4).
   If the tracker successfully updates, a bounding box is drawn around the object to indicate its current position and its center is stored in order to later visualize object's trajectories path across frames, connecting successive center points.
   If instead a tracker fails to locate an object, a message is printed to indicate that the tracker has lost the object, which may occur due to the object leaving the frame or being temporarily occluded.

Figure 4: Tracking algorithm on game3_clip1

3.6 2D Top-View Visualization

The *trajectoryProjecter* class provides a bird's-eye view of the billiard balls and their trajectories, visualizing their positions and movement on a table minimap. This process involves several key steps:

1. **Placing the Table Minimap:** the `projectBalls` function begins by loading the table minimap image from the predisposed directory. This image is then resized to fit correctly the designated area on the bottom-left corner of the frame.

2. **Assigning Corners:** to ensure accurate correspondence between the minimap corners and the previously retrieved table's corners, the function `sortCornersClockwise` stabilizes a clockwise order. To decide the order the function sorts the corners based on the angle they make with the centroid computed by the `computeCentroid` function.

3. **Computing Perspective Transform:** the perspective transformation matrix is computed for mapping the coordinates of the balls and their trajectories from the original frame to the 2D projection.

4. **Determine Table Orientation:** to determine the table's orientation, the code checks the diagonal lengths of the transformed corners. If the table in the frame is vertical, the code rotates the corners and recalculates the perspective matrix to align them to the bird's-eye view of the horizontal minimap.

5. **Transforming Ball Positions and Trajectories:** using the perspective matrix obtained in the previous step, the positions of the balls and their trajectories are transformed from the original frame to the bird's-eye view on the minimap.

6. **Drawing Trajectories and Balls:** then the transformed trajectories and balls are drawn onto the resized minimap image. Different colors are used for each ball based on their IDs, and trajectories are represented as lines connecting successive points.

7. **Overlaying the Minimap onto the Frame:** finally, the updated 2D top-view image, now completed with the projected ball and trajectories, is placed onto the bottom-left corner of the considered frame.

3.7 Metrics

3.7.1 Mean Average Precision (mAP)

The mean Average Precision (mAP) metric is essential for evaluating the performance of object detection models, as it combines both precision and recall indexes to provide a thorough assessment of the model's capability to classify and localize objects considering both localization and classification accuracy.

To compute the mean Average Precision (mAP) metric, several key steps are followed. The next steps are done for every detection of every of the classes of interests.

Let's restrict for now of having a single detection for a single class. Having the predicted and groundtruth bounding box (bbox) the Intersection over Union (IoU) for detection (rect by rect) is calculated using the `compute_IoU` function which computes the ratio of their intersection area over union areas.

Next, the `get_PR_table` function computes the Precision-Recall (PR) table along the class we're analyzing iterating over predicted bounding boxes and comparing them with the ground truth boxes using the IoU function before defined. True Positives (TP) and False Positives (FP) are determined based on a threshold set to 0.5 on the obtained IoU value: when above is a TP otherwise is a FP. Precision (P) and recall (R) cumulative indexes are computed respectively as $P = \frac{TP}{TP+FP}$ and $R = \frac{TP}{TP+FN}$ using cumulative TP and FP as we iterate trough the class, with the recall values rounded to the nearest ± 0.1 between 0 and 1 for the after explained 11-pt interpolation method.

The PR table is then refined using in the function `refine_PR_table` that removes duplicate points on the same recall value, discarding the lowest precision-value from the colliding ones (the reason of this is explained right after). As last step, we're interested to compute an approximation of the AUC (area-under-curve) below this zig-zag curve. Since we already fixed the colliding point issue happening when the PR curve falls vertically the remaining cases are only the ones of increasing P-value for subsequent R-values. Since we're now interested in approximating this zig-zag curve with a flat descending staircase this is the reason why the lowest point had been discarded before from the options available. Iterating from the end (1) to the beginning (0) of the PR-array it's easily possible to flatten it and compute the AUC as a piece-wise integral sum using the array points, obtaining finally the AP for the single class.

Finally, in the `compute_mAP` function the mean Average Precision (mAP) is calculated computing the PR curves and the AP value for each class and averag-

ing the result among the total number of classes.

PLEASE NOTE A note on this design choice has to be done. Since the AP index is not based on TN (true negatives) in the scenario where a class has no groundtruth items (e.g. a the black ball is already in a hole) and no prediction is detected (correctly) would result in a failure for the index cost, returning a value of 0. We decided to instead return 1 in this case, as an indicator of success, since, if there's no ball to detect, not detecting it is a good thing.

3.7.2 Mean Intersection Over Union (mIoU)

To compute the mean Intersection over Union (mIoU) for evaluating the segmentation performances , the function `calculateIoU` starts by calculating the Intersection over Union (IoU) for each class, this time pixel by pixel. This is done for each one of the six classes of interest and after averaged among the total number of classes (6).

4 Results

In Table 1 are reported the mAP and mIoU values obtained for each video clip provided in the dataset.
The other requested results are reported in the following. The bounding boxes and segmentation masks extracted from the first and last frame are depicted respectively in Figure 5 and 6. The 2D top-view minimap attached on the last frame for each video clip in the given dataset is shown in Figure 7.

Video	**mAP**	**mIoU**
game1_clip1	0.724026	0.666036
game1_clip2	0.60625	0.639594
game1_clip3	0.590909	0.646451
game1_clip4	0.712987	0.681554
game2_clip1	0.857955	0.715493
game2_clip2	0.556791	0.581031
game3_clip1	0.900568	0.710962
game3_clip2	0.545455	0.589396
game4_clip1	0.496212	0.57283
game4_clip2	0.857576	0.752839

Table 1: mAP and mIoU metrics for the Dataset

(a) game1_clip1 - First Frame

(b) game1_clip1 - Last Frame

(c) game1_clip2 - First Frame

(d) game1_clip2 - Last Frame

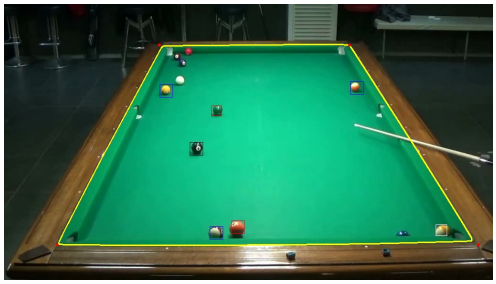
(e) game1_clip3 - First Frame

(f) game1_clip3 - Last Frame

(g) game1_clip4 - First Frame

(h) game1_clip4 - Last Frame

(i) game2_clip1 - First Frame



(j) game2_clip1 - Last Frame

(k) game2_clip2 - First Frame

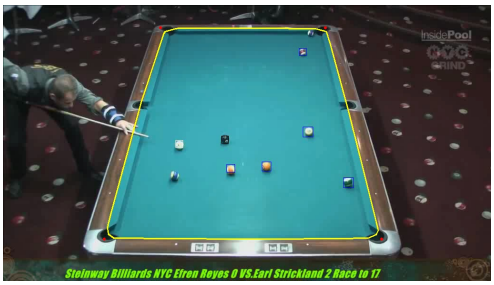
(l) game2_clip2 - Last Frame

(m) game3_clip1 - First Frame

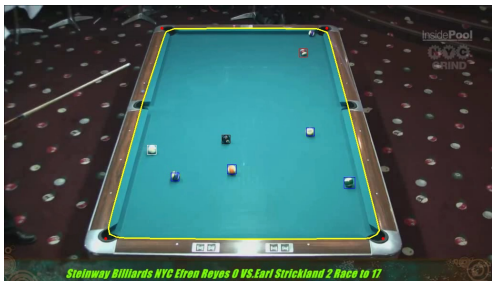
(n) game3_clip1 - Last Frame

(o) game3_clip2 - First Frame

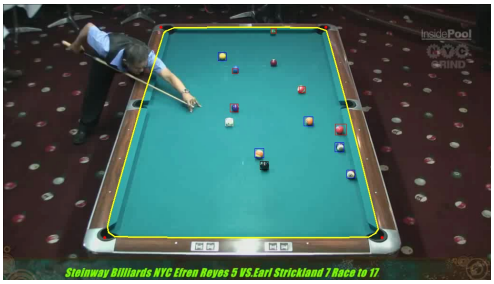
(p) game3_clip2 - Last Frame



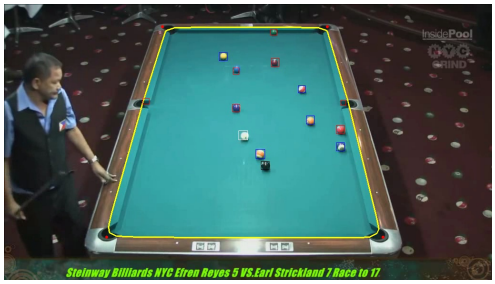
(q) game4_clip1 - First Frame



(r) game4_clip1 - Last Frame

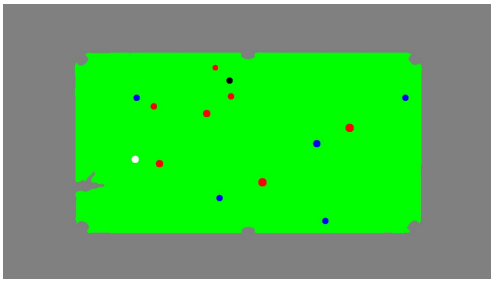


(s) game4_clip2 - First Frame

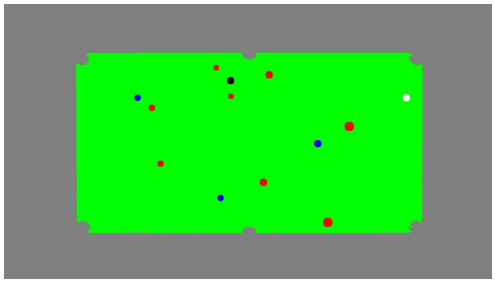


(t) game4_clip2 - Last Frame

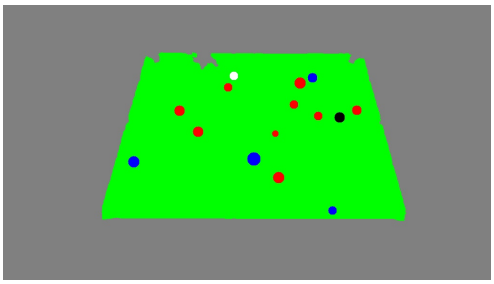
Figure 5: Detection results for each video clip



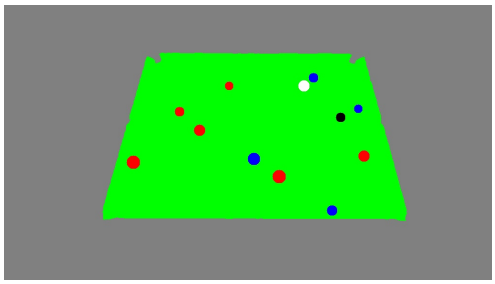
(a) game1_clip1 - First Frame



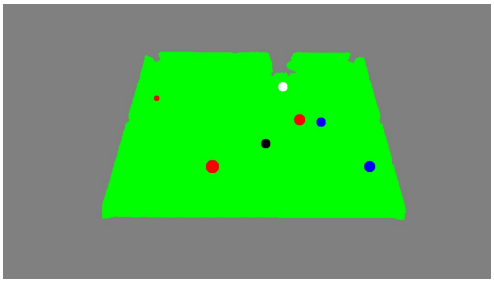
(b) game1_clip1 - Last Frame



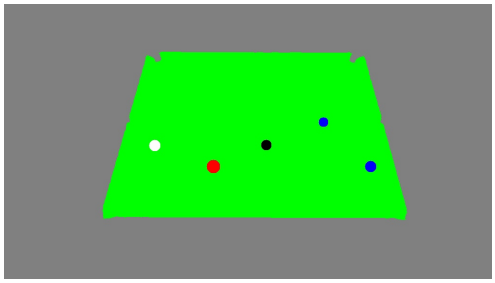
(c) game1_clip2 - First Frame



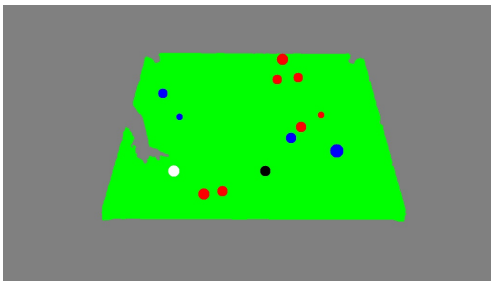
(d) game1_clip2 - Last Frame



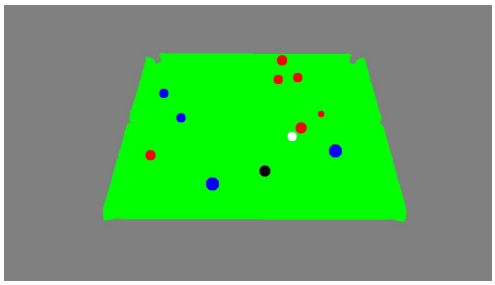
(e) game1_clip3 - First Frame



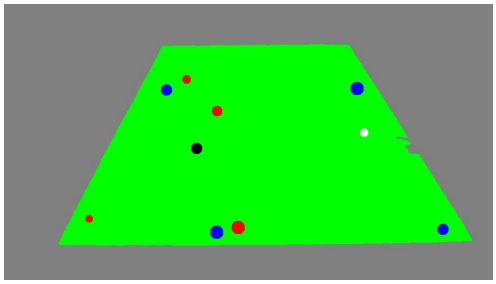
(f) game1_clip3 - Last Frame



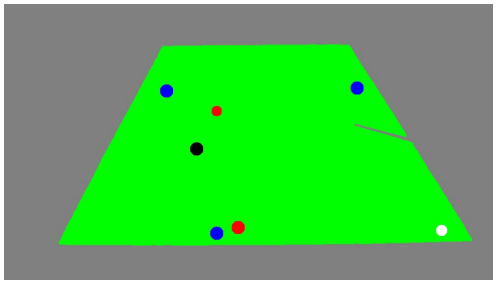
(g) game1_clip4 - First Frame



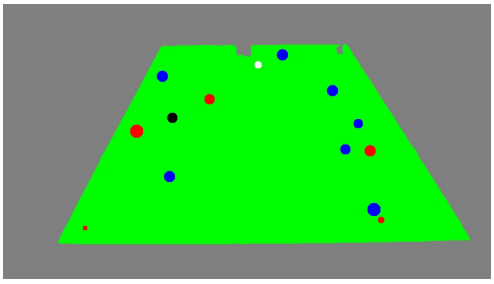
(h) game1_clip4 - Last Frame



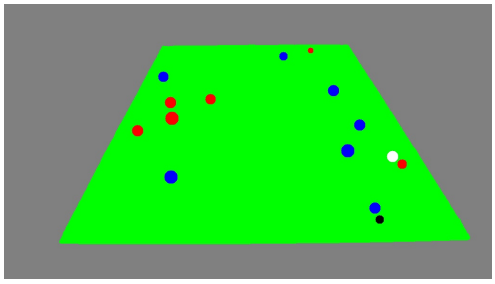
(i) game2_clip1 - First Frame



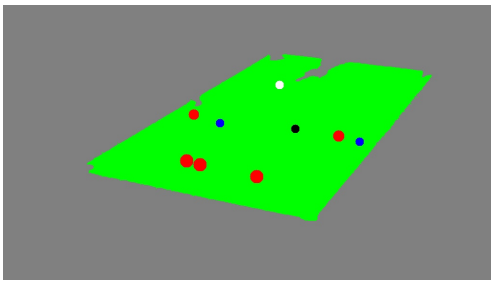
(j) game2_clip1 - Last Frame



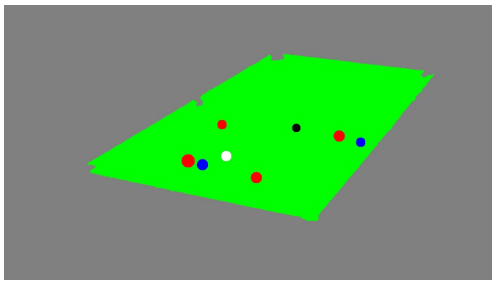
(k) game2_clip2 - First Frame



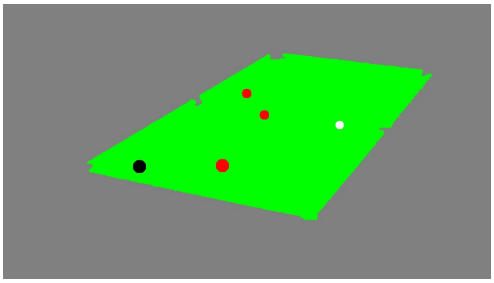
(l) game2_clip2 - Last Frame



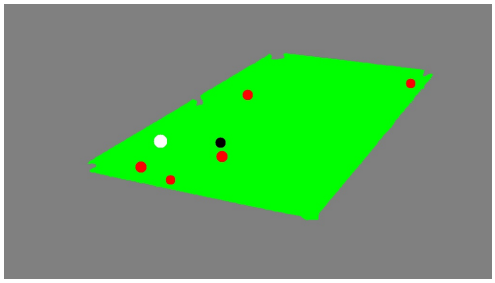
(m) game3_clip1 - First Frame



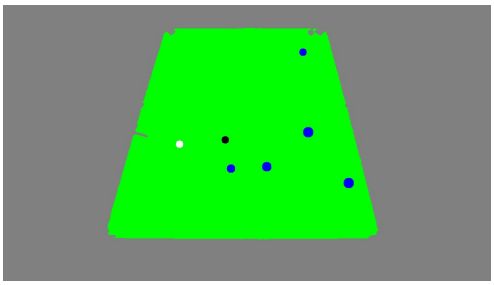
(n) game3_clip1 - Last Frame



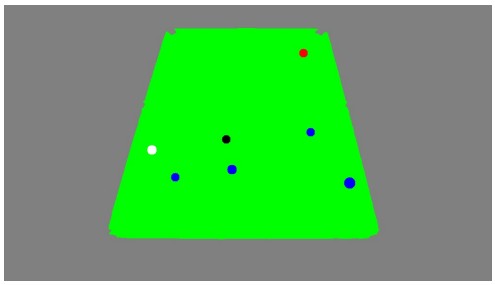
(o) game3_clip2 - First Frame



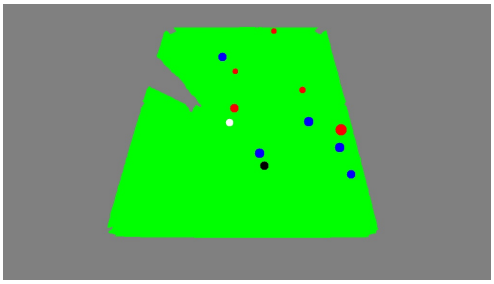
(p) game3_clip2 - Last Frame



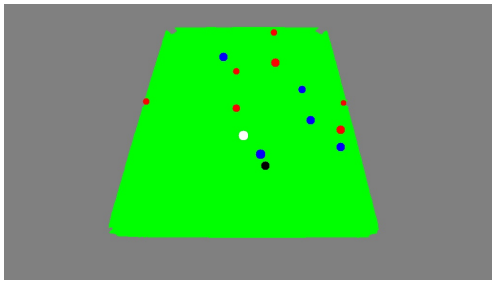
(q) game4_clip1 - First Frame



(r) game4_clip1 - Last Frame



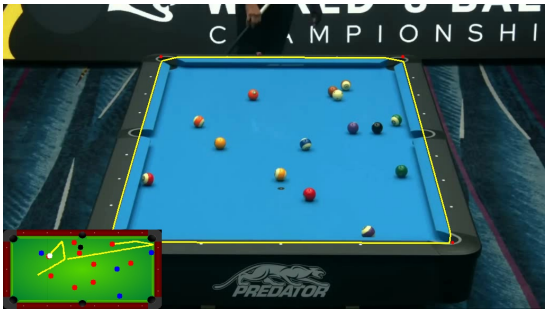
(s) game4_clip2 - First Frame



(t) game4_clip2 - Last Frame

Figure 6: Segmentation results for each video clip

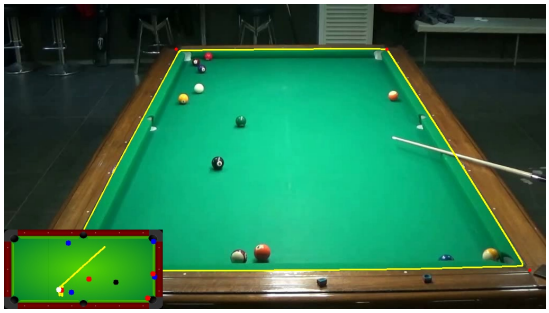
(a) game1_clip1



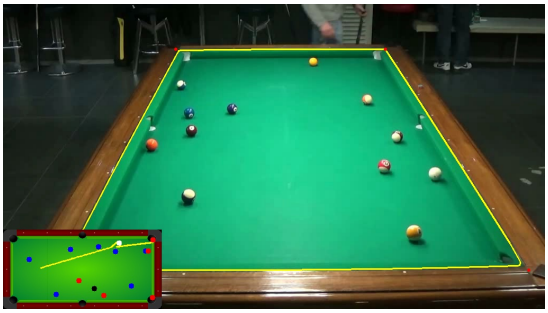
(b) game1_clip2

(c) game1_clip3

(d) game1_clip4



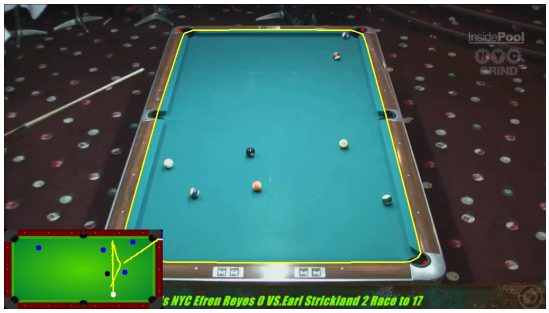
(e) game2_clip1



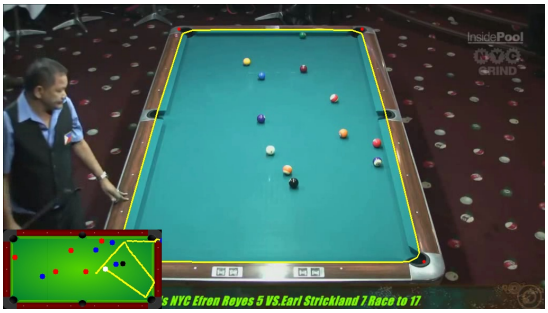
(f) game2_clip2

(g) game3_clip1

(h) game3_clip2



(i) game4_clip1



(j) game4_clip2

Figure 7: 2D Top-Views on final frame for each video clip

5 Conclusions

5.0.1 Insight on the Results of the Project

In conclusion, this project successfully developed a computer vision system for analyzing "Eight Ball" billiard games. The system demonstrated capabilities in real-time monitoring, automated scoring, and trajectory reconstruction. Key accomplishments include accurate identification of the billiard table, reliable localization of balls, effective segmentation and classification of playing field elements, and the creation of a dynamic 2D top-view minimap showing ball positions and trajectories. The system's performance metrics were satisfactory overall, though occasional inaccuracies were observed due to occlusions and lighting effects.

Improvements and Future Work

While the project has successfully achieved its primary objectives, several areas for improvement and future work have been identified:

- Improved Classification and Segmentation of Balls: Improving segmentation algorithms to more precisely differentiate balls from the background and other objects.
- Enhanced Ball Tracking: Improving the robustness of the ball tracking algorithm to handle occlusions and rapid movements more effectively.
- Additional Features: Integrating advanced features such as shot prediction and player behavior analysis to provide deeper insights into the game.
- User Interface: Enhancing the user interface to facilitate more intuitive interaction and visualization of game analysis is essential. Currently, the minimap lacks clarity when a ball is pocketed, which can hinder understanding. Improvements are needed to ensure that the minimap accurately and clearly reflects changes in ball positions and game status.
- Generalization: Expanding the system's capabilities to analyze other types of billiard games and potentially other sports involving ball tracking.

These enhancements will make the system more versatile and valuable for a broader range of applications in sports analysis.

6 Team Contributions

In our project, we effectively divided the tasks among team members to leverage our individual strengths. Each member took responsibility for specific aspects of the development, ensuring that we addressed all necessary components efficiently.

- Fresco Eleonora: was primarily responsible for the video handler and projector algorithms, in addition set the sequence of operation in the frame handler. This task required approximately 40 hours of work.

- Girardello Sofia: took charge of the ball detection, classification and segmentation tasks together with the trajectory tracking part.
  This segment of the project required around 50 hours of effort.

- Morselli Alberto: focused on the table detection and the metrics computation part, additionally prepared the frame handler class.
  Approximately 40 hours were dedicated to these activities.

Moreover Sofia and Alberto were responsible for integrating and testing the overall system: combining different modules, ensuring their compatibility, and conducting system-wide testing. Instead Eleonora collected all necessary data for the report, and searched the starting information regarding the OpenCV functions and the state-of-the-art of the algorithms used in the project.
However, collaboration was a cornerstone of our process; whenever a team member encountered challenges, the entire group came together to brainstorm solutions and provide support. This collaborative approach not only helped overcome obstacles but also ensured that everyone remained informed about each other's progress and contributions. By maintaining open communication and a shared understanding of the project's various elements, we were able to work cohesively and achieve our objectives successfully.