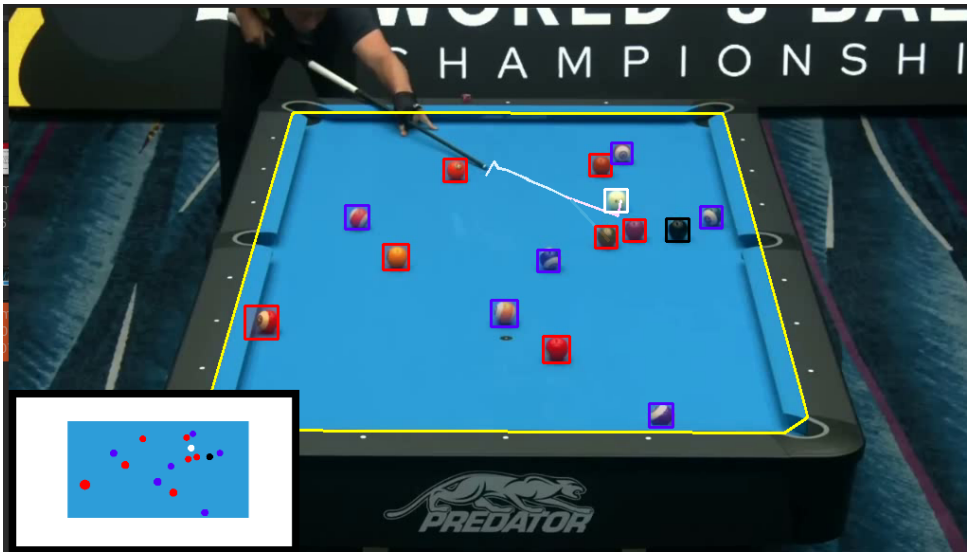
Eight Ball Pool Game Analyzer

An Analysis of Eight Ball Pool Game with Computer Vision System



Name
Alessio

Last Name
Zattoni

ID
2125280

University
Università degli Studi di Padova

Department
Department of Information Engineering

Course
Computer engineering

Date
20/07/2024

Index

1 Computer vision system description

1.1 Presentation

This project concerns the task of analizing the eight ball pool game in order to produce some statistics and some important information about the play.

In the project some assumptions are considered:

1. the pose of the camera does not change among the frames, so the table remains in the same position in the image for all frames of the same video, then the pool table boundaries are computed only at the first iteration, this preserve time

The main steps that was applied consists in:

1. Detect the pool table boundaries
2. Detect the balls
3. Classify the balls
4. Perform the segmenation of the pool table and the balls
5. Calculate the ball's trajectories
6. Draw the 2D minimap that rappresent the current game state

1.2 Detect the Pool Table Boundaries

This task consists of finding the boundaries of the pool table. For each frame, it is necessary to analyze the current frame in input frame to find the boundaries. The main challenge of this task is that the image has many lines and patterns, also the color of the table changes from image to image. In addition, lighting and the difference between the boundaries of the table and those of the playing field make this task even more difficult.

1.2.1 Algorithms

There are two algorithms used to detect the boundaries of the pool table, the first deals with the detection of the pool table, while the second deals with the refinement of the boundaries of the pool table, so with two steps the boundaries of the pool table are detected. The explanation of the algorithm follows:

detectPoolTable

Given an image, it tries to detect the pool table boundaries.

INPUT/OUTPUT

- Input: the current frame, Output
- Output: the set of vertices that compose the pool table boundaries

Preprocessing

1. Convert the frame from BGR to gray scale

Algorithm

1. Apply Otsu threshold
2. Find contours
3. Obtain the largest contour
4. Calculate the convex hull with this contour

Description
The preprocessing phase involves a gray scale conversion to works in a simpler color space. Next the Otsu thresholding is applied to the image resulting in a binary image, then the contours is found by `findContours` function defined in OpenCv library. The next step consists in find the largest contour since it will be likely the pool table, since it's a object with the largest area in the image. The last part consists in finding the convex hull given the contour, it's the minimum polygon that passes through all the points of contour. The pool table boundaries are computed only at the first frame, this thing works well since the camera position doesn't change in the videos, this thing is true by assumption. This algorithm tries to find the pool table, but this task doesn't always end well, so the application involves the use of another method called `refinePoolTable`, which will be presented in the next section.

---

refinePoolTable

Given an image, this algorithm tries to refine the pool table boundaries.

Input/Output

- Input: the current frame
- Output: the set of vertices that compose the pool table boundaries

Preprocessing

1. Apply the Gaussian filter
2. Convert the image to the HSV color space

Algorithm

1. Find the mode of the pixel in the image
2. Convert the BGR mode to the corresponding HSV value
3. Apply morphological closure
4. Apply a threshold with the mode and a tolerance
5. Find the contours in the resulting image
6. Obtain the second largest contour
7. Compute the convex hull with these points

Description

The preprocessing phase involves the Gaussian filter to smooth the image in order to remove the noise, next the image is converted in the HSV color to get more straight regard the illumination changes. Firstly starting from the pool table boundaries the mode is computed, this phase try to search the most present color i the table in agreement to the idea that the most present color in the image represents the color of the pool table. Once the value is retrieved you have to convert this color from BGR color space to HSV one, then a morphological operator is applied to the mask in order to join the thin contours, then a threshold is applied whit a custom tolerance that represent join to the mode a lower and upper bound in the color space. The thresholding phase produce an image where the table is separated by any other component not in the range, so at this point a function to find the contours is applied to the binary image in order to found the pool table boundaries. Next you have to find the second largest contour, because the first one represent the window's boundaries and we expected that the second one represent the table due to the size of the pool table. Finally a convex hull is computed with the points obtained by the previous phase, this polyhedron depicts the pool table boundaries enhanced starting from the boundaries found by the `detectPoolTable` call.

1.3 Detect the Balls

This operation is not so simple, because the balls have different color and different brightness and if the ball is next to the pool table boundaries the ball shape is not so clear to detect. These problems make for instance the Hough circle transformations not so suitable for this task, so you have to take into account the fact that not perfect circle could be present into the images indeed the balls will move frame by frame and the shape could deformed. All these issues make this task hard to implements in perfect way, I have purpose a version that exploits the segmentation of the image and try to find the ball's contours as precise as possible.

1.3.1 Algorithms

There algorithm try to detects the balls present in the image, it uses a similar technique adopted in the refine pool table boundaries function. Following the algorithm explanation:

detectBalls

Given an image, it tries to detect the balls.

Preprocessing

1. Apply the Gaussian filter
2. Convert the image to the HSV color space

Algorithm

1. Find the mode of the pixel in the image
2. Convert the BGR mode to the corresponding HSV value
3. Apply morphological closure
4. Apply a threshold with the mode and a tolerance
5. Find the contours in the resulting image
6. filter the contours
7. calculate the minimum enclosing circle given a filtered contour
8. Create the ball and add it to the set of balls

Description
The preprocessing phase involves the Gaussian filter to smooth the image in order to remove the noise, next the image is converted in the HSV color to get more straight regard the illumination changes. Firstly starting from the pool table boundaries the mode is computed, this phase try to search the most present color in the table in agreement to the idea that the most present color in the image represent the color of the pool table. Once the value is retrieved you have to convert this color from BGR color space to HSV one, then a morphological operator is applied to the mask in order to join the thin contours, then a threshold is applied whit a custom tolerance that represent join to the mode a lower and upper bound in the color space. The thresholding phase produce an image where the table is separated by any other component not in the range, so at this point a function to find the contours is applied to the binary image in order to found the balls contours. Next a filter process is applied in order to keep only the best candidate, this filter is based on a restriction on the width and height and another restriction by area. The following step consists in iterate through the remain contours and find the minimum enclosing circle by mean of the OpenCv function `minEnclosingCircle`, this function tries to find the minimum enclosing circle from the given contours, when the circle is find then the new ball is created and the ball' parameters is set up. When the iteration finished then a set with all the balls is returned.

1.4 Classify the Balls

This task consists in classify the balls given them a category, to do so we have to analyze the ROI(region of interest) to find some useful information that let us to label the ball. The problems related to this task regard the position and the gradient of the ball, indeed if the ball is not positioned in the best way, then a stripped ball will may looks like to a cue ball and the task will become quite hard or for instance if the frame is underexposed, the green solid ball will may appear as the eight one. In this phase the final result could contains a series of false positive and true negative samples, these predictions could affected in negative way the performance of the system.

1.4.1 Algorithms

There algorithm try to classify the ball by means of the colors present in the region of interest centered on the ball's center. Following the algorithm explanation:

classify

Given an image, it tries to classify the balls.

Preprocessing

1. Convert the image to the HSV color space

Algorithm

1. Set the lower and upper ranges for the category
2. iterate through the ROI and set the correspondent mask to 255 (active) if the pixel is in the range of the mask
3. count the number of non zero elements in each masks
4. classify the ball by mean of some percentage of pixel related to a mask present in the ROI

Description
The preprocessing consists in converts the input image from BRG to HSV color space to get more straight regard the illumination changes. Firstly three mask is declared: white mask, black mask, color mask; next the relative upper and lower bounds ranges is declared, afterwards some iteration is performed in the ROI so this iterations regards an enviroment twice the radius, at each iteration a pixel is extracted and next the pixel is compared to the ranges previously defined in order to figure out if the

pixel is in the range, if the pixel correspondent at given range so the relative mask is set to 255 at same position as to advice that the correspondent pixel is in the given range, once the iteration finished then the non zero pixel in every mask is counted so to obtain the number of the pixel in the white mask, black mask and colored mask. These values give us some information about the color present in the ROI centered in the ball. Next the counts of the pixel in each mask is compare by percentage relative to the total pixel in the image, so if the ROI presents a percentage of pixel above a threshold then the ball is classify same as the correspondent mask, for instance if the black mask has 40% of pixel set to 255 then the ROI has 40% percent of pixel in the range of darkest pixel so it will be classify as eight ball.

1.5 Perform the segmentation of the pool table and the balls

The segmentation consist in divide the image in different regions in agreement to a condictions as the color of the pixel or the distance between the pixel. Once obtained the pool table boundaries, the balls position and size you could segment the pool table and the balls in easy way, indeed once decide the color you could draw the different region just with a few function call.

1.5.1 Algorithms

There algorithm try to classify the ball by means of the colors present in the region of interest centered on the ball's center. Following the algorithm explanation:

segmentsImage

Given an image, it segments the pool table in four regions: pool table, eight ball, cue ball, white stripped ball and solid color ball.

Preprocessing

1. None

Algorithm

1. Get the pool table color computed by the previous phase
2. fill the pool table with the right color
3. fill the balls with the corresponding color by category
4. draw the pool table boundaries

Description
The pool table color is computed in the previous phases, so with the right color the table is filled. Next the balls is filled by the right color computed in the classification task and finally the pool table boundaries is drew. The final result present the inner area of the pool table segmented in four category: pool table, eight ball, cue ball, white stripped ball, solid color ball. Moreover the pool table boundaries are highlighted.

1.6 Calculate the ball's trajectories

This section is dedicated to the calculation of the trajectory in the game use the Lucas-Kanade algorithm to calculate new ball positions from their previous positions. There are a lot of issue due to the flow calculation, for instance a threshold is required in order to exclude the false positive balls that aren't move from one frame to another and moreover the parameters are not easy to tune in order to calculate the flow in the right way.

Preprocessing

1. Convert the image to the gray scale color space

Algorithm

1. Calculate the previous balls positions
2. Calculate the optical flow
3. filter the trajectory and associate the closest ball compare the centers of the balls
4. draw the line that represent the trajectory between the previous position and the current position of the ball

Description
The preprocessing phase expects the conversion of the color space from BGR to gray scale, this step is required because the Lucas-Kanade algorithm works only on gray scale images. The next phase concerns the calculations of the previous balls positions, this is made by find the center of the current detected balls. The following step concerns the optical flow calculation, for this task the Lucas-Kanade algorithm is applied, it takes as parameters the previous frame, the current frame, the previous balls positions, a window size, a value used as max level and a termination criteria, then it tries to find the flow vector for each main points that best explain the position of the pixel in the next frame minimizing the difference of intensity between two points, this process is iterated in agreement of the criteria passed as argument in order to obtain best results. Once the new balls position is calculated then if the algorithm has calculated in right way the low (status[i] == 1), then the distance between the previous point and the current point is calculated and if that distance is relevant (this exclude some false positive positions) then the closest center is calculated in order to obtain the right center of the ball, next if the closest center is found a line that represent the trajectory is drew starting from the previous position to the new position, at the last step the previous previous positions are updated as the previous frame is updated.

1.7 Draw the 2D minimap that rappresent the current state

This phase consists in draw a minimap in the left bottom corner that represent the current game state, the minimap is composed by the pool table, the balls and the trajectory of the balls.

Preprocessing

1. None

Algorithm

1. Create an empty image with all pixel set to 255
2. call drawPoolTableAndBalls function
3. draw a rectangular black border
4. resize the image by 30 percent
5. position the mini map in the right place
6. add the mask to show the trajectory

Description
The first step consists in create an empty white image so to highlight the minimap, next the function `drawPoolTableAndBalls` is called; this function draw the pool table surface and the balls inside it. Following some black border are drew in order to separate the minimap from the image, next the image is resized by 30% so to represents the mini map and it's placed in the right bottom corner. The final step consists in adding the mask calculated in the previous phase in order to draw the trajectory of the balls.

2 Classes description and program workflow

2.1 Introduction

In this chapter will be explained the classes of the application in order to let understand the structure of the application and more clarify the design. Moreover will be show a description of main method of the application.

2.2 Classes

2.2.1 Ball

This class represent a pool ball, it contains the methods to classify the balls and to draw the balls boxes once the balls are classified.

Attributes

- `center`: the center of the ball
- `radius`: the radius of the ball
- `classification`: the category of the ball

2.2.2 Pool table

This class represent the pool table, it contains the methods to draw the pool table and to get the pool table image.

Attributes

- `vertices`: the boundaries of the pool table
- `color`: the color of the pool table

2.2.3 EightBallPoolGame

This class represents the eight ball pool game as current instance of the play, it contains main method such: balls detection, pool table detection, analysis execution and information printing.

Attributes

- `balls`: the balls in the game
- `poolTable`: the pool table in the game

2.2.4 Detection

This class is used to compute some statistics of the game, it fits the ground truths structure so it allows to compute in convenient way the IOU for the instance.

Attributes

- `x`: ordinate of the top left corner of the box
- `y`: abscissa of the top left corner
- `width`: width of the box
- `height`: height of the box
- `category`: category of the detection
- `confidence`: the value of confidence in the detection

2.2.5 Utility

This class contains many utility methods used in the application.

Attributes

- None

2.3 Workflow

This section aims to explains the workflow of the application in therms of main methods that could be used to make the eight ball pool analysis.

2.3.1 Methods descriptions

The library makes available two methods to do the analysis of the play, the first is defined to works with command line arguments instead the second one is designed to be called in the program. These two methods work in the same way, by means of the path passed by parameter the methods understand if the path refers to an image rather than a a video, so internally call an appropriate method than process in the right way the relative image/video. These last two methods call the same function that perform the true analysis with the given parameters. Following a brief descriptions of these methods:

analyzeSystem

The method accept the following parameters:

1. `path`: this parameters is referred to the path of image/video that should be processed, this parameter is required by the program
2. `showTableDection`: this parameter is use to show or hide the pool table boundaries, its default value is false
3. `showBallDection`: this parameter is use to show or hide the ball detection, its default value is false
4. `showSegmentation`: this parameter is use to show or hide the segmentation of the area relative to the pool table, its default value is false
5. `showMinimap`: this parameter is use to show or hide the mini map, its default value is false
6. `pathToGtmAP`: the option to calculate the mAp whit given gt file
7. `pathToMaskmIoU`: the option to calculate mIoU with give mask image

analyzeSystem

These method accept a list of command line argument as a program options. Following the flag that are accepted:

1. `-path`: this parameters is referred to the path of image/video that should be processed, this parameter is required by the program
2. `-showTableDection`: this parameter is use to show or hide the pool table boundaries, its default value is false
3. `-showBallDection`: this parameter is use to show or hide the ball detection, its default value is false
4. `-showSegmentation`: this parameter is use to show or hide the segmentation of the area relative to the pool table, its default value is false
5. `-showMinimap`: this parameter is use to show or hide the mini map, its default value is false
6. `-mAP`: the option to calculate the mAp whit given gt file
7. `-mIoU`: the option to calculate mIoU with give mask image

The method parse the command line argument to find the flags depicted above, then call the method `analyzeSystem` defined previously.

---

processVideo

This method open the camera by OpenCv api and next create an eightBallPoolGame object that depicted the current game, next occurs as iterations as the length of the video. At each iteration is called the method `executeAnalysis` that performs the true analysis.

---

processImage

This method read the image given by the path and next create an eightBallPoolGame object that depicted the current game, next is called the method `executeAnalysis` that performs the true analysis.

---

executeAnalysis

This method is the core of the library, it performs the main operation on the image/video. At the beginning it tries to detect the table, next it tries to detect the balls and in the next phase in agreement to flag it shows the relative result (table detection, ball detection, game segmentation, mini map).

2.4 Tree workflow

Now will be present a tree view of the workflow so to clarify the flow of the program. You could call one of the `analyzeSystem` in agreement to the fact that you call this method in you program or you would like to call this method in console, so to test the library.

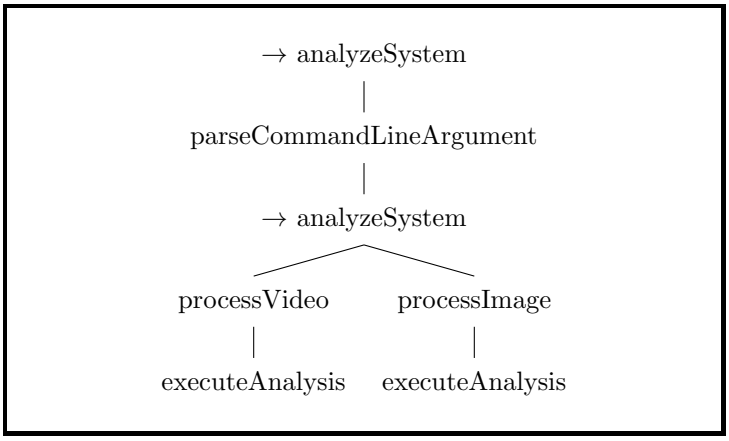


Figure 2.1: Workflow diagram

The entry point are the two functions `analyzeSistem`, if you call the version that accepts command line arguments then the `parseCommandLineArgument` is called in order to parse the command line arguments, instead if you call the other version the flow continue. Next the flow is separated in two ways in agree to the fact that is passed a path to a video or to an image, but in both cases the `executeAnalysis` function is called to perform the true analysis.

3 Benchmark

3.1 Introduction

This section will display some tests performed as benchmark in order to understand the accuracy of the designed computer vision system. The first metrics is `mAP` these metrics will measure the precision of the ball detection task, the second one is the `mIoU` and it will calculate the precision in detect: background, white ball, black ball, solid color, striped and playing field.

3.2 mAP

For ball localization, the mean Average Precision (mAP) calculated at IoU threshold 0.5, so it says that the ball is considered as true positive only if its IoU is greater than 0.5. In the analysis the confidence value is set constant to 0 so in the sorting phase the values don't change the position, this choice is made by simplicity but anyway the system is predisposed to sorting by confidence so if a new method that calculate the confidence will be implemented, so the system will work at the same way without any changes. The manual process to calculates the mAP could be described in a series of steps, to evaluate an object detection model Average Precision (AP) is calculated for each class of detected objects:

1. `Record Predictions`: First, collect all predictions for a specific class, along with their confidence scores. This data is crucial for subsequent analysis.
2. `Calculate Precision and Recall`: Sort the predictions in descending order based on confidence scores. For each prediction, determine cumulative True Positives (TP) and False Positives (FP). Using these cumulative counts, calculate precision and recall. Precision is the ratio of cumulative TP to the sum of cumulative TP and cumulative FP. Recall is the ratio of cumulative TP to the total number of ground truth objects for that class.
3. `Plot Precision-Recall Curve`: After calculating precision and recall for all predictions, plot the Precision-Recall curve. It's essential to ensure that if multiple precision values correspond to the same recall value, only the highest precision is retained for simplicity.
4. `11-Point Interpolation`: To smooth the Precision-Recall curve and minimize fluctuations, apply the PASCAL VOC 11-point interpolation method. This involves finding maximum precision values at specific recall thresholds (0.0, 0.1, ..., 1.0) and averaging these values to compute AP.
5. `Calculate Average Precision (AP)`: The final AP for a class is computed as the average of the interpolated precision values across the 11 points.
6. `Compute Mean Average Precision (mAP)`: Once AP has been calculated for each class, mAP is obtained by averaging these AP values across all classes. This metric provides an overall assessment of model performance.

3.3 mIoU

For the calculation of MIoU we need the labelled matrix of both predicted result and expected one (ground truth), the two images will have a pixel value between 0 and 5 in agree to the category:

1. `background`

2. `cue ball`
3. `eight black ball`
4. `solid color ball`
5. `stripped ball`
6. `pool table`

So this results in a couple of gray scale images. Now you have to calculate the intersection and the union among the categories, this has to do for each category, in the next step the `intersection over union` (IoU) is calculated by the following formula:

$$IoU = \frac{Intersection\_area}{Union\_area} \tag{3.1}$$

Once the IoU is computed for each calsses, then the `mean intersecion over union` (mIoU) is computed as the mean of the IoU for each classes:

$$mIoU = \frac{1}{class\_number} \sum_{n=1}^{\text{class\_number}} IoU(n) \tag{3.2}$$

3.4 Dataset description

The data are represent in tabular format, the first column represent the ordinal, the second column the predicted values, the third one represent the ground truth values and the last one represent the matches. The predicted value e the ground truth one have several column:

- `category`: the category of the ball (1-4)
- `x`: ordinate of the top left corner of the box
- `y`: abscissa of the top left corner
- `width`: width of the box
- `height`: height of the box

The last column represent the resulting matches, the possible values are:

- `TP`: true positive
- `FN`: false negative
- `FP`: false positive
- `MC`: miss classified

3.5 benchmark

The benchmark is performed on two sets of data, the sets contain the first and the last frame. These frame is characterized by a different camera position, distance and pool table color; following a list of the images used in the benchmark:

1. game1_clip1 first frame
2. game1_clip1 last frame
3. game1_clip2 first frame
4. game1_clip2 last frame

3.5.1 game1_clip1 first frame

ID	Predicted	x	y	width	height	Ground truth	x	y	width	height	Match
1	cue ball (1)	268	316	20	20	cue ball (1)	268	317	17	17	TP
2	eight ball (2)	475	494	46	46	eight ball (2)	469	154	15	15	TP
3	eight ball (2)	468	153	18	18	eight ball (2)	-	-	-	-	FP
4	solid color (3)	535	364	18	18	solid color (3)	534	363	18	18	TP
5	solid color (3)	320	324	20	20	solid color (3)	321	324	18	18	TP
6	solid color (3)	716	248	20	20	solid color (3)	716	249	18	17	TP
7	solid color (3)	417	221	18	18	solid color (3)	417	221	18	17	TP
8	solid color (3)	307	206	18	18	solid color (3)	307	205	17	18	TP
9	solid color (3)	470	187	16	16	solid color (3)	469	186	17	17	TP
10	solid color (3)	666	446	22	22	white stripes (4)	668	447	17	16	MC
11	white stripes (4)	445	397	18	18	white stripes (4)	444	396	18	18	TP
12	white stripes (4)	650	284	18	18	white stripes (4)	647	283	21	22	TP
13	white stripes (4)	537	252	16	16	white stripes (4)	535	251	18	19	TP
14	white stripes (4)	270	187	20	20	white stripes (4)	271	187	18	18	TP
15	white stripes (4)	833	185	20	20	white stripes (4)	834	186	18	18	TP
16	white stripes (4)	435	125	18	18	white stripes (4)	434	123	18	19	TP

Table 3.1: Predicted and ground truth for the given dataset

mAP

Class	AP (%)
cue ball	90.91
eight ball	45.45
solid color	77.92
white stripes	81.82
Mean Average Precision (mAP)	74.03

Table 3.2: Average Precision (AP) per Class and Mean Average Precision (mAP)

mIoU

Class	IoU (%)
background	96.70
cue ball	64.69
eight ball	10.02
solid color	61.24
white stripes	60.86
pool table	96.19
Mean IoU	64.95

Table 3.3: Intersection over Union (IoU) per Class and Mean IoU

3.5.2 qualitative result

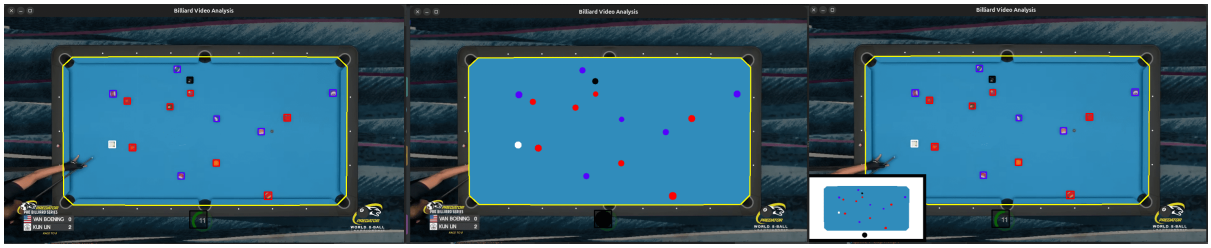


Figure 3.1: From left to right: ball and pool table detected, pool table segmentation, mini map visualization

3.5.3 game1_clip1 last frame

ID	Predicted	x	y	width	height	Ground truth	x	y	width	height	Match
1	none	-	-	-	-	cue ball (2)	321	201	17	17	FN
2	eight ball	468	153	18	18	cue ball (2)	469	153	15	16	TP
3	solid color (3)	535	364	18	18	solid color (3)	535	364	16	16	TP
4	solid color (3)	321	325	18	18	solid color (3)	321	325	18	17	TP
5	solid color (3)	716	248	20	20	solid color (3)	717	249	17	17	TP
6	none	-	-	-	-	solid color (3)	302	209	18	15	FN
7	solid color (3)	470	186	16	16	solid color (3)	469	186	17	17	TP
8	solid color (3)	549	142	18	18	solid color (3)	549	142	17	17	TP
9	white stripes (4)	666	446	22	22	white stripes (4)	668	446	16	17	TP
10	white stripes (4)	445	397	18	18	white stripes (4)	444	397	18	17	TP
11	white stripes (4)	651	285	18	18	white stripes (4)	650	284	17	17	TP
12	white stripes (4)	270	187	20	20	white stripes (4)	272	187	16	17	TP
13	white stripes (4)	303	195	38	38	white stripes (4)	321	325	18	18	MC
14	white stripes (4)	833	185	20	20	white stripes (4)	834	185	18	18	TP
15	white stripes (4)	435	125	18	18	white stripes (4)	434	123	18	19	TP

Table 3.4: Predicted and ground truth for the given dataset

mAp

Class	AP (%)
cue ball	0.00
eight ball	90.91
solid color	81.82
white stripes	86.36
Mean Average Precision (mAP)	64.77

Table 3.5: Average Precision (AP) per Class and Mean Average Precision (mAP)

mIoU

Class	IoU (%)
background	97.95
cue ball	0.00
eight ball	78.73
solid color	63.19
white stripes	44.72
pool table	97.00
Mean IoU	63.60

Table 3.6: Intersection over Union (IoU) per Class and Mean IoU

3.5.4 qualitative result

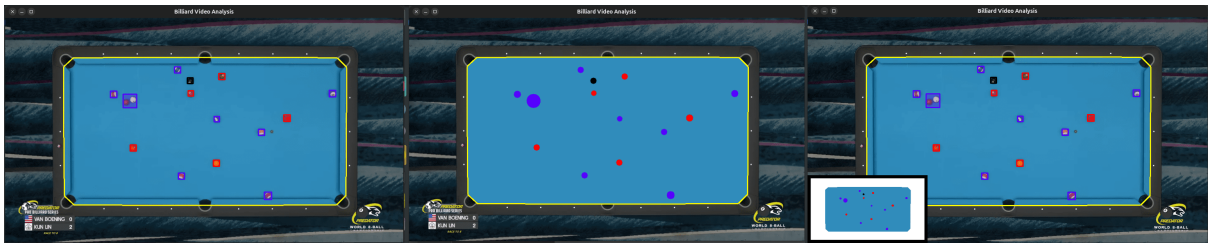


Figure 3.2: From left to right: ball and pool table detected, pool table segmentation, mini map visualization

3.5.5 game1_clip2 first frame

ID	Predicted	x	y	width	height	Ground truth	x	y	width	height	Match
1	cue ball (1)	473	141	22	22	cue ball (1)	475	141	19	21	TP
2	eight ball (2)	694	224	24	24	eight ball (2)	696	223	22	23	TP
3	none	-	-	-	-	solid color (3)	567	347	24	26	FN
4	solid color (3)	397	253	26	26	solid color (3)	399	252	22	23	TP
5	solid color (3)	649	224	24	24	solid color (3)	650	222	22	24	TP
6	solid color (3)	599	203	22	22	solid color (3)	600	200	21	23	TP
7	solid color (3)	460	158	26	26	solid color (3)	462	160	21	22	TP
8	solid color (3)	614	156	24	24	solid color (3)	617	156	19	22	TP
9	white stripes (4)	677	418	26	26	white stripes (4)	677	418	26	22	TP
10	white stripes (4)	511	311	26	26	white stripes (4)	513	309	23	26	TP
11	white stripes (4)	561	257	22	22	white stripes (4)	560	252	23	26	TP
12	white stripes (4)	733	211	22	22	white stripes (4)	733	209	21	22	TP
13	white stripes (4)	357	210	26	26	white stripes (4)	360	210	21	24	TP
14	white stripes (4)	636	142	24	24	white stripes (4)	638	143	20	22	TP
15	solid color (3)	252	315	34	34	white stripes (4)	262	318	24	26	MC

Table 3.7: Predicted and ground truth for game1_clip2 first frame

mAp

Class	AP (%)
cue ball	90.91
eight ball	90.91
solid color	68.18
white stripes	81.82
Mean Average Precision (mAP)	82.95

Table 3.8: Average Precision (AP) per Class and Mean Average Precision (mAP)

mIoU

Class	IoU (%)
background	98.32
cue ball	84.64
eight ball	76.92
solid color	47.86
white stripes	64.20
pool table	95.08
Mean IoU	77.84

Table 3.9: Intersection over Union (IoU) per Class and Mean IoU

3.5.6 qualitative result

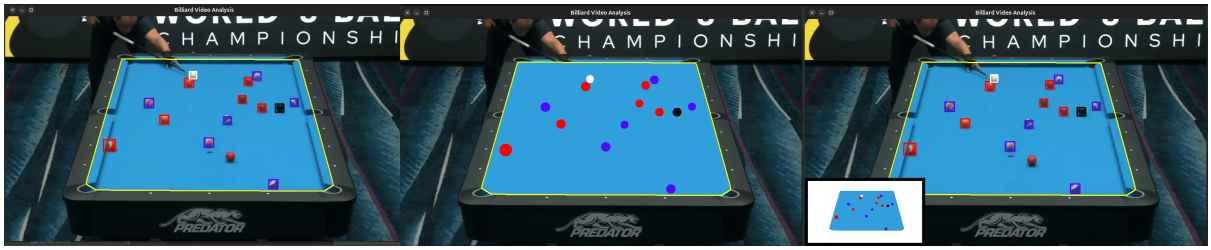


Figure 3.3: From left to right: ball and pool table detected, pool table segmentation, mini map visualization

3.5.7 game1_clip2 last frame

ID	Predicted	x	y	width	height	Ground truth	x	y	width	height	Match
1	white stripes (4)	611	136	48	48	cue ball (1)	620	161	23	22	MC
2	eight ball (2)	693	223	26	26	eight ball (2)	696	222	26	26	TP
3	none	-	-	-	-	solid color (3)	567	346	24	27	FN
4	solid color (3)	738	304	28	28	solid color (3)	741	303	24	25	TP
5	solid color (3)	397	252	26	26	solid color (3)	399	251	22	24	TP
6	solid color (3)	648	224	24	24	solid color (3)	650	222	22	24	TP
7	solid color (3)	459	159	26	26	solid color (3)	462	161	21	21	TP
8	white stripes (4)	611	136	48	48	solid color (3)	614	154	19	21	MC
9	white stripes (4)	677	418	26	26	white stripes (4)	677	418	26	23	TP
10	solid color (3)	252	315	34	34	white stripes (4)	262	318	23	26	MC
11	white stripes (4)	510	311	28	28	white stripes (4)	511	309	27	28	TP
12	white stripes (4)	561	257	22	22	white stripes (4)	560	253	24	24	TP
13	white stripes (4)	357	210	26	26	white stripes (4)	360	210	21	24	TP
14	white stripes (4)	731	210	24	24	white stripes (4)	733	208	21	23	TP
15	white stripes (4)	611	136	48	48	white stripes (4)	638	142	20	22	TP

Table 3.10: Predicted and ground truth for game1_clip2 first frame

mAP

Class	AP (%)
cue ball	0.00
eight ball	90.91
solid color	50.91
white stripes	72.73
Mean Average Precision (mAP)	53.64

Table 3.11: Average Precision (AP) per Class and Mean Average Precision (mAP)

mIoU

Class	IoU (%)
background	98.74
cue ball	0.00
eight ball	75.88
solid color	43.55
white stripes	46.95
pool table	95.69
Mean IoU	60.14

Table 3.12: Intersection over Union (IoU) per Class and Mean IoU

3.5.8 qualitative result

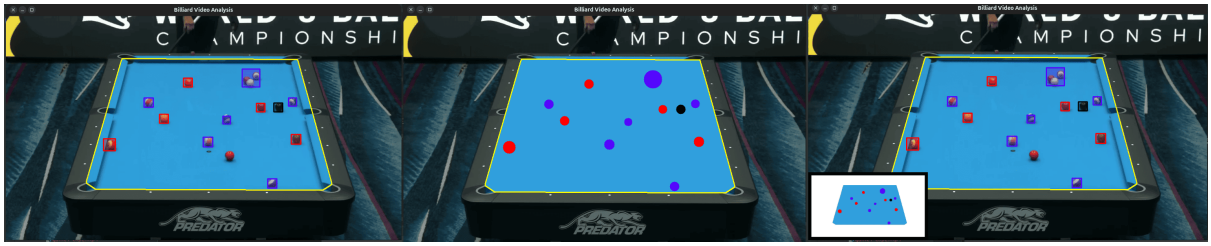


Figure 3.4: From left to right: ball and pool table detected, pool table segmentation, mini map visualization

4 Conclusion

4.1 Conclusion

The algorithm works well in situations where the balls are not too close together; this is a disadvantage because it creates false positives and false negatives. To solve this problem, one could change the strategy or fine-tune the parameters to get better results. All in all, the system always detects the pool table, however, regarding the location and classification of the balls needs to be improved to get better results. Regarding trajectory computation I implemented a first version that turns out to be inaccurate, since it does not filter out false positives, I did not have time to refine the algorithm for trajectory computation so the next step is precisely about filtering trajectories to keep only true trajectories, thus removing false positives. In the benchmark phase the parameters are tuned to obtain the best possible results, so if you want to replicate the results you have to change the parameters of the detectBalls call regarding detection and movementThreshold regarding trajectories.

4.2 Notes

- The hours spent on the project are about 80
- all the ideas and implementation came/made by me
- I used CLion as the IDE to develop this project
- the libraries used in the project are the standard ones and OpenCV