Detection, Identification, and Tracking of the Billiards Balls

Chauhan, Dhaval
dmc8686@rit.edu

April 21, 2018

Abstract

In this report, I present a method for detecting, identifying, and tracking of the balls present on the Billiards Pool table. I also present and explain alternative approaches for the purpose of identification and tracking of the balls, with the advantages and disadvantages of them. The intent behind this project idea was to automate the process of on or off-screen digital score updating through segmentation, classification, parsing of the balls. The challenges, failure scenarios, improvements on the current system, and extension of this idea on other similar applications are also discussed in brief towards the end.

1 Introduction

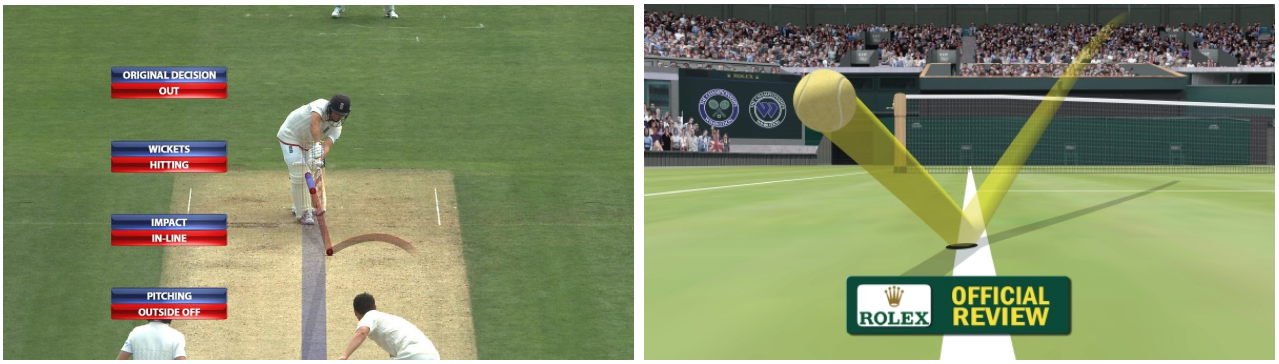
Billiards is one of the most popular indoor games played by a number of different types of people, as both, a leisurely activity, and as part of a professional pool player's career. While there isn't much need of an automated system when this game is played by a group of friends just as a pass time activity during lunch break or something, there is a need for a system like this in professional games that get broad-casted on the live Television, to help automate and simplify the process of analysis, expert discussion, and score or status related information extraction. Mainly because when things are live on Television, there isn't much time for the technicians to produce accurate visuals for the experts who talk and make comments on the shots based on the physics of it for the entertainment of the viewers.

Image 1 shows a frame from the live broadcast of a professional game in which the expert commentators use what seems like a set of manually controlled tools, to add a physically inaccurate movement path visual of a ball on the screen. Had there been a system that automatically made predictions and generated such visuals as the game progressed in real time, there would have been a much better and an easier discussion for the experts to make. Such systems are already there in place for some very popular worldwide outdoor sports like Football, Cricket, Tennis and more. See Image 2(a) and Image 2(b). The technology used in these examples is called "Hawkeye", which is very often used in games like Cricket, Tennis, Badminton and Squash. These graphics are automatically generated by the system and it also helps with decision making by using an appropriate parsing methodology as shown in Image 2(a). These tracking and motion prediction systems are used in various other situations as

well in the real time to generate many different types of statistics. The difference between such systems in games like Billiards, and other games like Cricket and Tennis is the necessary hardwares required. While games like Cricket and Tennis needs several powerful cameras in different orientations, Billiards can be aided by just one relative less powerful camera.

The system developed by me isn't capable of doing some of the things I mentioned above, yet, due to the constraint of time. My system doesn't predict the motion of the ball before the ball is hit. It can however, trace the path of the ball as it is in motion since the balls are being tracking in every frame. Pre-shoot motion can also be added as a feature to this tool by the untested approach I will explain later on. This prototype of the tool serves as a proof that such full feature-rich tools can be designed and put into use for such applications, given proper hardware support. Hardware is a crucial component of such systems. My system is not capable of real time application support due to limited computing power of my computer. However, given proper hardware support for raw computational power, even my naive methodology can give us a tool that will be suitable of real time usage. This can be said so because some work on this has already been done some vision researchers and practitioners.

Figure 1: Manual addition of graphics to a live Billiards game during an expert analysis session. Frame from broadcast of World Pool Masters 2018, Match between Niels Feijen of Netherlands and Shane van Boening of the USA. Image courtesy: Genipool youtube channel.



(a) Example of how Vision is used in Cricket to make a decision of ruling a Batsmen out or not out in a game of cricket. Image courtesy: ESPN

(b) Example of how Vision is used in Tennis to make a decision of ruling a ball inside or outside of the court. Image courtesy: ESPN

Figure 2: Hawkeye Technology: a fully automated AR Vision system

2 Related Work

There has been plenty of work done on vision based systems mainly on pedestrians, urban city semantics, traffic and roads and so on. There has been some work done on sports related applications, but, not much on Pool table game specifically. Researching for this topic was a challenge in itself, let alone researching for the specifics in such a system. I found only a handful of works that was on the exact same topic as mine.

Bauza and team [1] from University of California, San Diego designed POOL-AID, a tool that guides beginner players by displaying shot and ball trajectories during a game of standard 8 ball pool, for their Embedded Systems course project. Most of my work is based on the strategies used and experimented by them during their own work. Their work however, includes the use of an Embedded device like FPGA that has the advantage of accelerating vision algorithms while being power efficient and portable enough to become a standalone embedded device in the future.

Lars Bo Larsen and co [2] did a similar thing where they developed an Automatic Pool Trainer (APT) with the help of Augmented Reality. The APT serves a double purpose as a platform for research of multi modal user interaction using computer vision and speech recognition, agent technologies, etc. and for the development of a specific application, namely a computer aided learning system for the game of pool. Image 3(a) and 3(b) shows their AR based Pool trainer. Their pool training system is based on a platform developed some years ago within the Chameleon project [3] carried out at Aalborg University.

While these works have been excellent, they have heavily relied on the use of additional hardwares. Making use of these hardwares in my own work seemed like an overkill and it also felt unnecessary for the scope of this course' project. I adapted most of the strategies from [1], while coming up with my own ideas and using them which may or may not have been used before by others for a similar or a different application.

(a) The Pool Trainer at the Danish Science Fair 2004

(b) Example of the menu, displayed on the pool table.

Figure 3: Automatic Pool Trainer: Aalborg University, Denmark

3 Implementation & Design

I carried out my implementation in C++ using most of the methods and techniques from the free OpenCV library. Highlevel algorithm is shown below in Algorithm 1. I will explain each and every part of the algorithm with my way of implementation, and few other alternatives if any, with their advantages and disadvantages. Details of the specific tasks of the algorithm are explained below.

3.1 Technical Details

The algorithm only works on the video of the pool table taken from the top, such that the camera is perpendicular to the plane of the table and the frame centers the pool table. That is usually how it is taken in the professional games. See Image 1 again. Color values are hardcoded into the system, therefore, care must be taken that color values match the color of the table and the balls. Or change the values to match them. This is a challenge as well, because, there are no standards for the color of the table and the balls. Also, the table should be a regular sized table with no markings on it. Markings may interfere with the segmentation and detection process.

3.2 Segmenting Pool Table

There are a couple of ways in which the top of the pool table could be segmented. I used the third method to get the pool table.

1. One way to get the pool table would have been to use Hough Line Detector to detect the strongest of the lines of the frame and get 4 edge lines of the pool table. Although this method sounds straight forward and simple, it has a couple of downsides. First, this method wouldn't be full proof. Cue-stick, On screen embedded score boards, additional

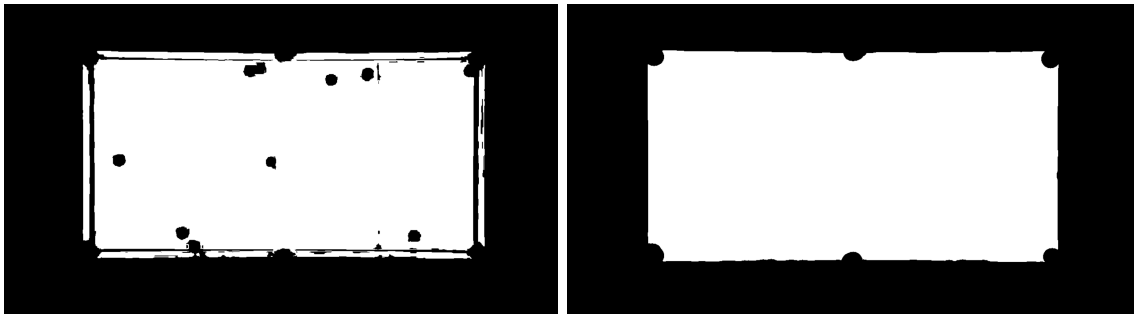
furnitures in the frames etc. all could have gotten in the way of detecting the correct edges of the pool table. A similar approach would be to use a Hough Rectangle Detector. But the disadvantage of getting False-positives would still exist considering the surroundings of the pool table. Yes, by defining some standards we could have made this approach work. But that still wouldn't fix the disadvantage that inherently comes with using a Hough Detection method of any kind. Hough Detection is expensive since it is voting based and it needs random number of iterations to the detections right. The wise thing would be to not use this method.

2. Other technique would be to use template matching using a template of the pool table across multiple scale spaces of the frames. This technique would work pretty if the template and pool table have similar standards like aspect ratio, size, and colors, but, it would still be a very expensive process to run a template through every pixel of every scale space of the frame.

3. This last option that was described in [1] is the one that I felt was the best and the simplest, both, computationally for the system, and cognitively for the developer. In this method, we get the color of the pool table by either of the following methods:

   a) Hardcoding the table color.
   b) Asking for the user to select a part of the table and extracting the color from that.
   c) Computing a histogram of the Colors in the frame and selecting the peak one (assuming the pool table dominates most of the frame space) as the pool table color.

   I used the simplest hardcoding approach. I read the color information of the entire frame, picked the values of the pixel that belonged to the top of the table, and hardcoded this value in the code later. This approach is also used for identification of the balls. Once the Color of the pool table was subtracted, I Go ahead and create a binary image of the subtracted part versus the non subtracted part. This gives a raw segmentation of the pool table. See Figure 4(a). To get a more refined version of the pool table, I perform a morphological opening on the binary image with an elliptical kernel of size 15 pixels. This removes most of the noise from the binary image from the balls and the cue sticks, and brings the table back to its original size. See Figure 4(b).



(a) Raw segmentation of the pool table (b) After morphological opening.

Figure 4: Segmenting Pool Table

Algorithm 1 Ball Detection, Identification, and Tracking

```
1: procedure processVideo(VideoFile)
2:     read VideoFile
3:     for First Frame of VideoFile do
4:         procedure SegmentPoolTable(FirstFrame)
5:             subtract table color from FirstFrame
6:             generate a binary image from the subtracted frame
7:             morphological Opening on the binary image using elliptical kernel of size 15
8:             generate a rectangular contour of the segmented table
9:     for Rest of the Frames of VideoFile do
10:        get circles using Hough Circle Detector on Frame(green)
11:        discard circles not on the pool table
12:        for Every circle on the table do
13:            get a rectangular contour of the circle
14:            get cropped image of that contour
15:            procedure identifyBallNumber(croppedimage)
16:                subtract table color from croppedimage
17:                compute average of Blue in subtractedimage
18:                compute average of Green in subtractedimage
19:                compute average of Red in subtractedimage
20:                return ballnumber based on the average colors
21:            draw a circle around the ballcircle
22:        write new frame to outputVideo
23:    return outputVideo
```

3.3 Detecting & Tracking Balls

There can be multiple ways of detecting Balls inside a frame. However, I could think of only two while thinking of the implementation. I used the second one from the below two methods.

1. First approach is to Get the Table part of the frame and subtracting the table color from the frame again. This would give us a binary image giving us a segmentation of the table from the table, somewhat similar to the Image 4(a). Next, we would find small circular contours on the table top that belongs to balls. Once the contours of all the balls are obtained, we could have obtained concrete circles of the balls by computing the minimum and maximum row and column values of the contours and obtaining the center of the ball by averaging the minimums and the maximums, and radius by halving the range value of the largest of the width, or height of the contour. This approach is a computationally inexpensive approach, but it fails to detect the blue ball, and the balls that are too close to the edge of the table top, or to the pockets, as, a part of these balls would get segmented out as a part of the surrounding of the table.

2. This second approach is the one that I used in my implementation of the system. This is an easier approach to understand, but, computationally heavy on the hardware as it uses Hough Circle Detection on every frame. Computation was kept to the minimum by keeping the range of the radii of the ball as narrow as possible. The Hough was allowed to

detect only those circles that had radii of minimum 8 pixels and maximum 12 pixels. This setting allowed me to capture all the balls on the table top, a 100% of the time See Figure 6 and Figure 7. Also, circles that were outside the contour of the table were discarded. This circle detection step was supposed to be one time thing, only for the first frame and after that an object tracking algorithm was to take care of rest of the annotations. However, I failed to setup the link between my environment and the OpenCV contribute library which hindered my plan of using a genuine tracking algorithm. More on this is discussed in the later sections. However, detection in every frame does give us an illusion of tracking, be it, a very inefficient.

3.4 Identifying Ball Numbers

There are now sophisticated ways of detecting objects in color images. Most of these techniques involve the use of a deep network like a Convolutional Neural Network. Yes, This approach can be used to identify the type of the ball with a very high accuracy. The only problem is the need of training this network, and the volume of the time and the labeled dataset required to train even a simple network. Due to lack of both time, and a dataset of the balls, I used a more conventional method of recognition here which doesn't involve training and which involves manual feature extracting, and a naive approach to decision making.

My idea of identifying the number of ball involved analyzing the color of the ball, and the pattern on it. Once the circles are obtained in the previous detection step, I compute a square bounding box of the ball and crop that part of the frame out for analyzing the color of the ball. These bounding boxes are no more than 20 pixels wide or tall, as most of the circles detected are less than 10 pixels in radius. See image 5.

The cropped bounding boxes of the balls are then sent to a separate procedure that subtracts the part of the pool table from the corners of the boxes. Once this is done, We compute averages of the values in the Blue, Green, and Red channels of these parts. We compare these averages with the hardcoded color template of the balls to identify the type of the ball. One alternative method is do the same thing on the HSV variant of the input cropped image. HSV color space narrows down the color range a bit more because only one channel carries the color information, i.e. HUE channel. HSV worked better than RGB in my experiment because, averaging didn't seem to affect negatively when comparing with the HSV templates.

This approach worked well for identification of the Solid balls. However, it didn't work for the stripped balls. I had to come up with a different mechanism to detect the stripped color balls. I calculated the number of white pixels in the image, and the number of other color of the pixels, after subtracting the pool table color. And I compared the ratio of white to color counts to a predefined threshold of value 0.2. It didn't work as expected because the resolution of the ball is just too low. number of white pixels in stripped balls never exceeded the value of 2, which kept the ratio from crossing the threshold of 0.2. Lowering the threshold value would have confused the system into classifying solid balls as stripped balls because of the specular reflections from the solid balls. I gave up on recognition of the stripped balls for the time being and left it for future work.

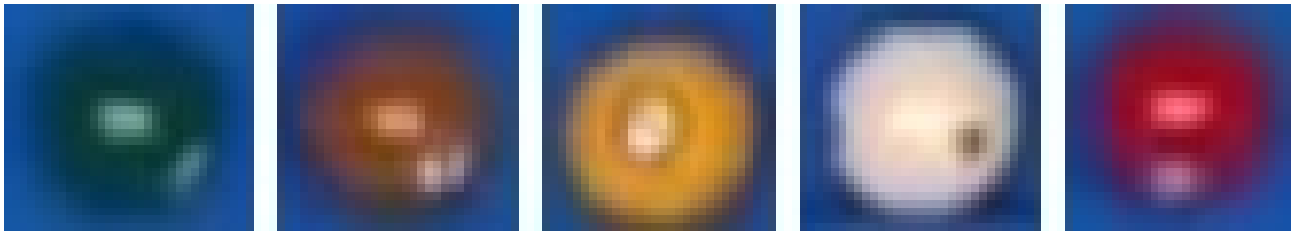


Figure 5: Cropped square bounding boxes of the detected balls

Once the type of ball was identified, however accurately, the number of the ball was printed on the frame, on the top left corner of the ball's bounding box. See figure 6 and 7.

4 Results

Figure 6 and 7 show the results from the system implemented as described above. Balls get detected a 100% of the time and get identified correctly a decent amount I'd say. There is no concrete accuracy on the identification because the dataset is not labeled. Labeling the videos with almost 500 plus frames would not have been feasible in the give time frame. Tracking was achieved by running detection algorithm on every frame as opposed to running it on just the first frame of the video.

5 Challenges, Failures, & Future Work

Identification of the type of ball still remains a challenge due to the low resolution image of the balls. Perhaps, making use of a CNN in future will fix this, or at least give us some true-positives of the stripped balls. Inability to use the contribute library was a huge blow to me, as a part of the result, and as part of the learning for me. I will be putting some more time and effort on getting a working contribute library and trying to get at least one type of tracking algorithm work on this application. I had Kernelized Correlation Filters tracking algorithm in my mind that gave good accuracy with decent speed. I think, given the abundance of time later on, trying other algorithms won't be an issue. A motivation to make use of these sophisticated tracking algorithms was to get a real time output FPS from the system. Running inefficient Hough Circle Detectors on every frame of the video takes too much of time and disables the system to output processed frames at a faster rate. The average output FPS from the current system is around 9 to 11 frames per second.

No work was done on the bonus part of the motion prediction because of the lack of time from above mentioned hindrances. However, I still think that this feature can be implemented by detecting the presence of the cue stick, or a part of it, on top of pool table. Once the stick is detected, an approximate line along the orientation of the cue stick could be computed by skeletonizing of the cue stick's blob. Given the line, the direction, the location of the ball in the direction of the cue-stick, and edge lines of the table, a motion trajectory can be computed assuming zero friction for a fixed amount of displacement.

Figure 6: Output-1: Ball Detection Using Hough Circles and Ball Identification using Colors.

Figure 7: Output-2: Ball Detection Using Hough Circles and Ball Identification using Colors.

6 References

[1] Samuel Bauza, Brian Choi, Chuanquiao Huang and Olga Souverneva, "POOL-AID", University of San Diego, 2014.

[2] Lars Bo Larsen, Rene B. Jensen, Kasper L. Jensen, and Søren Larsen. Development of an automatic pool trainer. In Proceedings of the 2005 ACM SIGCHI International Conference on Advances in computer entertainment technology, ACE '05, pages 83–87, New York, NY, USA, 2005. ACM.

[3] Lars Bo Larsen,Peter M. Jensen, Kenneth Kammersgaard, and Lars Kromann. "The Automated Pool Trainer - a Multi Modal System for Learning the Game of Pool." Intelligent Multi Media, Computing and Communications : Technologies and Applications of the Future. IEEE Computer Society Press. 2001, pages 90-96. accessed online June 2 2016.