Tracking and Simulate Billiards Balls' Movement Using Computer Vision

XIAO WANG, HANG JIANG
University of New South Wales

Abstract

Billiards, sometimes is also called pool, is one of the favorite leisure sports around the world. In this project, we build a computer vision system to track different balls in the video clips (with static camera) of 9-ball billiards/pool game, and project them into a 2D board and simulate the balls' movement. The system recognizes the game table, and then detect, identify and track balls using computer vision technique on object features such as shape and color. In the end, the system projects the ball movement onto the 2D board and show the simulation. The performance of the system is decent on most case, however, the failure rate of the "striped" ball tracking is still high. Moreover, since the training and testing video's resolution is limited, the accuracy of ball detection and identification is not perfect.

Keywords: pool game; computer vision; object reorganization; object tracking; color histogram; blob detection; homography

1. Introduction

We are pool ball fans, so we decided to apply the computer vision technique to this field, which is an important motivation of our project.Besides, pool game seems to be suitable for computer vision analysis, since the visual parameters, such as the color and shape of pool balls, the uniform background of pool table, are highly constrained.

The goal of the project is to recognize, identify and track different pool balls in the pool game video, as well as project the real pool game onto 2-D plain board and simulate the pool ball movement in the pool game. This could be useful for further working such as pool game analysis, automatic scoring, training and recording.

Under the condition of fixed camera and pool game environment, the system would firstly identify the pool table, and then detect and identify the pool balls which is one of the challenges of the project in terms of the accuracy of the pool ball identification. Different light intensity, closely laying of many pool balls, differentiation of stripped balls and non-stripped balls are all issues we are trying to solve. Besides, the system would dynamically track the pool balls in the video as well as project the pool ball and its trajectory onto 2-D plain board after the pool ball identification. The challenge for this part mainly involves with the efficiency of dynamic tracking, which could be always improved by modifying the tracking algorithm and identification accuracy trade-off. The system was written in Python using Opencv3 library.

We also tried to use machine learning algorithm such as support vector machine to improve the accuracy of the pool ball identification. Because of time limitation, we have not completed it due to the main issue of the preparation of the large amount of training dataset. It could be regarded as the further development of the system. Further development of the system could also include more robust dynamic tracking by modifying the tracking logic.

2. Related Work

Object tracking is widely used in many sport games, which could be used for game analysis, training [4], recording and automatic scoring. Billiard game is one of the typical scenario.

There are many approaches for object recognition. Many of them are based on object color and shape features. The specific methods include matching using color/HOG histogram [2][3],Canny detector and Hough transform to find specific shapes [5] and Blob detection [1]etc. For this project, we tried several of these methods, and choose the combination of blob detection, color features with some optimizations based on our context.

For tracking, there are many dynamic models such as Kalman Filtering [6], mean shift [7]. For this project, we use simple model to keep track of balls (mainly based on object recognition combined with basic prediction logic such as distance and direction threshold). The experiment proves it is effective.

For projection part, we calculate the homography from the pool table in the video to 2D plain board. Homography is a common method to realize this type of transformation [8].

3. Context Analysis

3.1. Pool game specification

For the pool table, in 2.1 a standard pool table is seen from top view, which is used for the project.

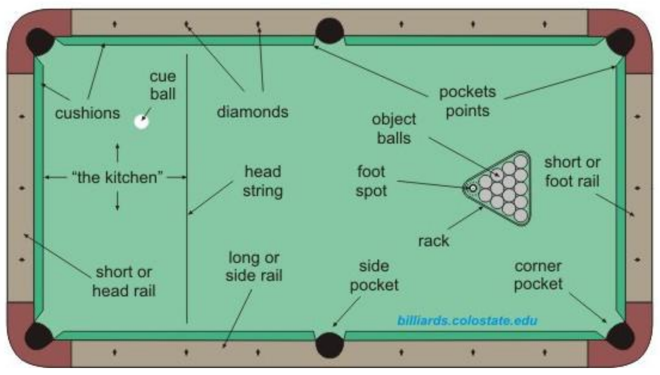


Figure 1.Pool table sample. From https://au.pinterest.com/explore/pool-table-dimensions/

For the pool ball, this project would focus on the standardnine-ball type. All balls should be 5.715cm in diameter. A complete set of balls consist of:

- Cue ball: White
- Solid balls:1:Yellow, 2:Blue, 3:Red, 4:Purple, 5:Orange, 6:Green, 7: Maroon, 8:Black.
- Striped Balls: 9:Yellow

These specifications will be used in order to know what size and color of table, balls etc. that have to be looked for while doing the image processing. Additionally, the camera used is fixed at the angle of degree 45~90 of the pool table.

3.2. Color Analysis of Table and Balls

To detect the pool table and balls it is important to understand how they appear in different color spaces. In this section the colors of the table and the pool balls are measured for further analysis[2][3][4].

3.2.1. The pool table

To decide which color space (RGB/HSB) is better to be used to detect pool table, different images of the pool table are analyzed in RGB color space and HSB color space respectively. The analysis would be usefulfor pool table identification.

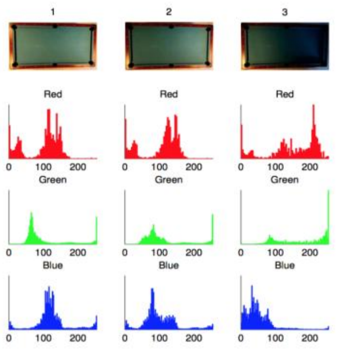


Figure 2. RGB Images to show the pool table in different light situation.

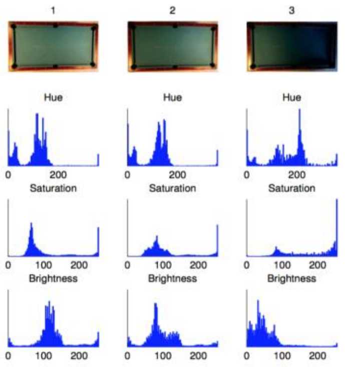
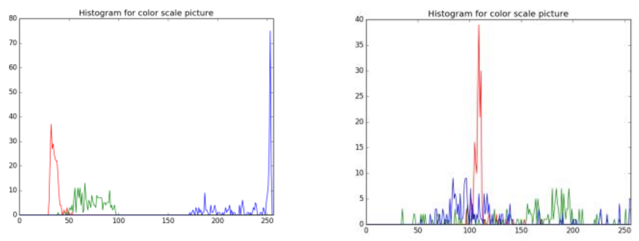


Figure 3.HSV Images to show the pool table in different lightsituation

From the above figures, we know that using HSB color space is easier to identify the pool table than using RGB color space. This could be the result of brightness invariant property of hue space channel.

3.2.2. The pool balls

One of the most challenges of this project is to successfully detect and identify the pool balls in different situations. And the pool ball color analysis is the basis for the success of the identification. For the project, the system would compute the HSV histogram for all pool balls as the preprocessing. We use matplotlib library to compute and visualize the HSV histograms for all pool balls, and define the benchmark HSV color ranges for each pool ball to identify each specific pool ball in the implantation part. Some HSVhistogram samples are shown below:



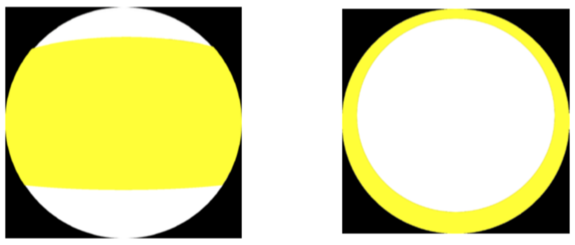
a.cue ball HSV histogram b. ball #2 HSV histogram

Figure 4. Sample pool ball HSV histograms

Some issues are generated here. For example, from the figures, we know that the color of 3, 5 pool balls have similar hue value, and are mostly distinguished by the saturation channel value. Besides, some HSV histograms,such as those of 5, 7 pool balls, show that the hue value is distributed into two remote areas (maximum and minimum value areas), which makes it more difficult to define color ranges for target area during the implementation. Thus, some optimizations or improvements are tried to increase the identification accuracy, which would be explained in part 3.

3.2.3. Stripped ball and non-stripped ball

Generally, this is also one of the challenges. The solution we use is to define a threshold for the proportion of the write color the balls contain. If the write proportion is larger than e.g. 15% (exclude cue ball), than we think it is stripped balls, otherwise it is non-stripped balls. The color analysis for specific ball 9 is shown below:



a. Ball 9 with minimum write b. Ball 9 with maximum write

Figure	a	b
Color	548839	175858
Write	173401	542900
Percentage	24.0%	75.5%

Figure 5. Pool ball 9 color compositions

3.3. Overall specifications

The tasks for the project are:

- Read and process the video frames
- Identify the pool table
- Detect and identify pool balls
- Identify pool balls with high accuracy
- Track pool balls in real-time and try to do it efficiently
- Project the video frames onto 2-D plain board
- Draw pool ball movement trajectories on the board

4. Design and Implementation

4.1. System Overall

- **Programming language**

The system is written in python3, using opencv 3.1 lib. We follow the Object-oriented programming principal to design and code.

- **Program Flow**

Here is the Program Flow of our system:

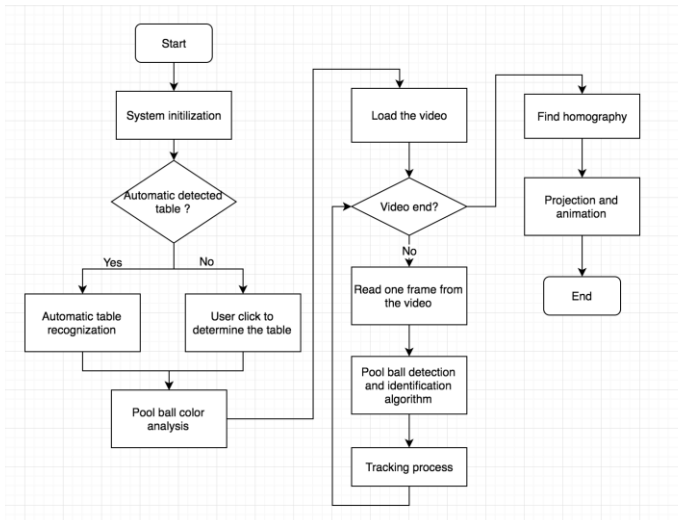

Start
System initilization
Automatic detected table ?
Yes
No
Automatic table recognization
User click to determine the table
Pool ball color analysis
Load the video
Video end?
No
Read one frame from the video
Pool ball detection and identification algorithm
Tracking process
Find homography
Projection and animation
End

Figure 6. Program flow chart

4.2. Table Detection

The First key task for the system is to detect the table area in the video. The purposes of this task is:

✓ The detected table region will serve as the ROI (region of interest) for further detection of balls.

- ✓ The key points of the table (i.e four corner and pockets) will be used to calculate the homography matrix in the projection process.

The outcome of the task is to find:

- ✓ key points (4 corners' coordinate)
- ✓ ROI mask

There are several ideas for finding the table, we need to choose one that is robust and has low complexity to compute.

Idea 1 - User input (One optional backup solution for the project)

This idea is simple, and without any computer vision technique, which just ask user to determine the table area. We actually provide this method as the backup plan. It will show the sample video snapshot and ask user to click four corner of the table as a fix order.

Idea 2 - Recognition of Pocket

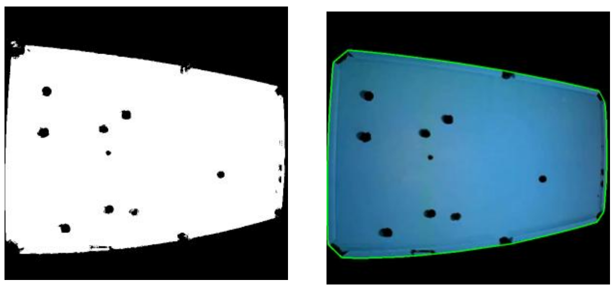
This idea assume pocket has a strong unique feature in the video. If we can find all 6 pockets, then we can determine the table area. We tried this one, but the result is poor. The reason is even though the pockets has the low brightness, something outside the pool table (the background), there are also quite a few objects and regions that are dark as well. It makes the method make some significant miss match.

Idea3 - Finding the "Most Common Color" and apply the color range mask. (Final solution for the project)

It's obvious that even for the worst case, the table cloth color makes the most frequent color in any pool game videos, so we can use image color histogram (we use HSV color space, as its advantage has been stated in the analysis part) to find this primary color range, and generate a image mask. Then, we apply the mask to the original image, and find the largest contours which indicate the table region.

Now, another problem is how to find the 4 corner key points based on the contours. Our basic idea is: first, we find convex hull of the contours, which helps to remove the effect of the occlusions of pockets. Then, we find the approximate polygon to cover the convex hull (use opencv's approxPolyDP function). Adjust the parameters, to find the suitable quadrilateral[5]. Make the four corners of the quadrilateral as corners of table in the video.

Here are some intermediate output image of process:



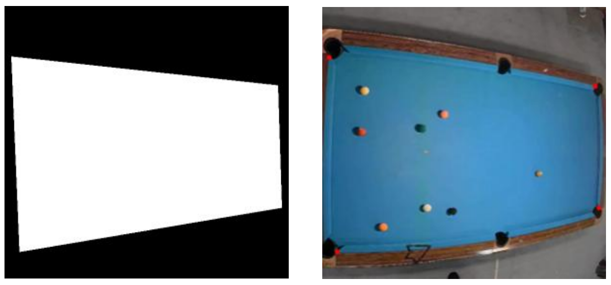


Figure 7. Table detection

We test several images with different color of table cloth, the performance is pretty consistent. So we make it as our final solution.

4.3. Ball Detection and Identification

This is the most critical part of our project, and also the most difficult part. There are many optimization and trade-off when implementing it.

First, there are two general approaches:

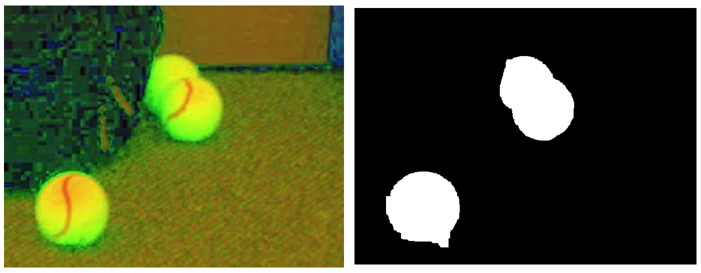
1. Detect balls first, then identify them to specific ball number.
2. Detect and identify each ball in one process.

In our project, we use the later one, as for either detection or identification, we use similar approach, that is heavily rely to color analysis.

There are quite a few ideas to detect/identify balls.

Idea1 – Hough Transform and Circle Detection

The idea[5] is to find the ball contours using ball color mask, and then use Hough transform to find the circle (for OpenCV, it will apply Canny edge detector before transform). For example, the detection of tennis balls:



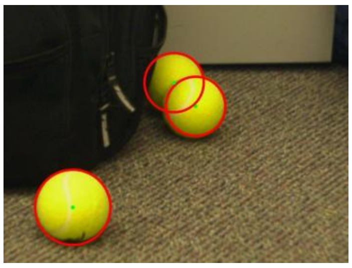


Figure 8. Hough Transform to find circle

One of the pro of this idea is it doesn't require us to find the table region (ROI).

We tried the method in our context, but the result is very bad. It's tough to find a good parameters combination. Either it finds many noises that are not ball objects, or it miss quite a few balls. Also, the accuracy decrease significantly when we apply it to the moved ball.

Idea2 – Template Matching

It uses the image matching technique. To implement it, we need to build a template image lib that contains a large amount of balls, considering the different turning positions, and put them into different categories (tagged by ball number). Then, do a match between video frame and template, to find the equivalent class.

This approach has a few serious drawbacks:

- ✗ need sufficient template ball images.
- ✗ High time complexity, as for each template we need to match the whole table area.
- ✗ When two balls' color are close enough, it's hard to distinguish them.

Idea3 - Blob Detection + Color Range Mask + Further Estimation (Final solution for the project)

Back to our case, the ball regions have a very strong feature: all these regions can be filtered out by the table cloth, just like the holes. So we can use blob detection[1] which is provided by OpenCV (cv2.SimpleBlobDetector_create() function). There are also some parameters/threshold that we can control, for example, filter out the size of blobs.

Here is the detection effect:

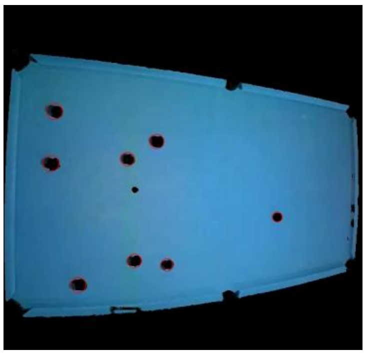


Figure 9. Blob detection

Each blob area(hole) is either one ball, or set of multiple balls.

Now, for each blob, we apply the mask using specific ball's color range. for example, if we want to find #5 ball, we use #5 ball's standard color range as mask. It will generate two possible results:

1. this blob will be entirely removed, that means this blob doesn't include the #5 ball.
2. This blob will not be entirely removed, there are still some pixels left. It means the blob potentially includes the #5 ball. We put this region into one list as an *candidate region*.

If there is no *candidate region*, return None, means we cannot find this ball in the frame.

The last step is to do some further estimation to find the most possible ball position among *the candidate regions* generated above.

Color Estimation and Comparison Strategy for *Candidate Ball Regions*

We create a estimation system based on Color Distance.

```
for each candidate ball region:
        for each pixel:
                calculate the distance between the pixel
color and template ball's mean color.
        Calculate the average color distance in the region.
        Compare all regions' average color distance, make
the smallest one the ball position.(it's approximate position, as
we will do final adjustment which will be stated  later on. )
```

The basic algorithm is shown below:

The key issue in distance calculation, is to choose a distance measurement: Cosine Distance or Euclidean Distance?Cosine Distance vs Euclidean Distance

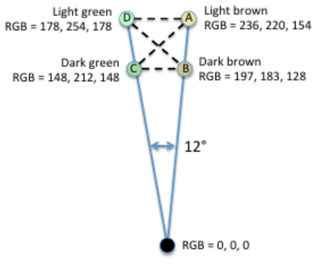
In RGB color space, the Euclidean Distance is:

$$D_E(v_1, v_2) = \sqrt{(r_1 - r_2)^2 + (g_1 - g_2)^2 + (b_1 - b_2)^2}$$

In RGB space the Euclidean Distance is more sensitive to differences in intensity than differences in color. This can cause problems if a light version of a color should match a darker version.The Cosine Distance is:

$$cos\theta = \frac{v_1^T v_2}{||v_1||||v_2||}$$

The distance is represented by the vector angle. The distance between two colors having the same chromaticism but different intensity will be the same.



(a) Visual representation of color distance

-	A	B	C	D
A	0	59.7	88.6	71.4
B	59.7	0	60.3	88.9
C	88.6	60.3	0	59.7
D	71.4	88.9	59.7	0

(b) Euclidean distances

Figure 10. Different color distance measurement

The Figure shows how these two distance measurement works. In our case, the chromaticism and intensity of the color should both be heavily considered in our ball identification system, and the brightness is stable and have small weight, so we choose Euclidean Distance.

Ball Position Fine-tuning

The above steps help the system to find the approximate ball region. To make the ball position more precise, we design a small Fine-tuning strategy.

We setup a small window area, whose center is the region center. Traverse every point in the window, make them as ball center respectively, estimate the average color distance within the ball radius. Compare all the possible ball center point, pick up the point with smallest average color distance as our final ball center.

4.4. Real-time Tracking

Once we can successfully identify pool balls, then real-time tracking could not be a big issue for this project. The two important aspects for this part are the tracking efficiency and the tracking robust.

4.4.1. Tracking efficiency

The simple approach for real time tracking is to apply the pool ball identification algorithm for each whole frame, so that the system could locate the pool balls and track them in real time. However, this is not very efficient, which causes the video to play a little slower than original one. The solutions could be:

4.4.1.1. Resize each frame before processing

Resize each frame in the beginning for further processing every time, which is a common method to improve the efficiency.

4.4.1.2. Ignore several frames

Another simple solution is to discard a few frames each time in terms of the large quantities of frames for a video. If the pool ball identification algorithm (explained in section C) is very accurate, discarding a few frames and continue to track is feasible to improve the tracking efficiency without loss the identification quality. However, this method is not appropriate if the ball identification algorithm is not robust or the requirement of tracking continuity is high.

4.4.1.3. Define small window for ball identification

Not only consider the tracking efficiency, but also to guarantee ball identification accuracy as much as possible, another solution is to define a small window as searching space and only perform the ball identification algorithm (explained in section C) within that small window for several internal frames. After each several internal frames, performing the ball identification on the whole frame at a time. Only if the ball identification on small window fails, it would try the identification on the whole frame again.

For this project, finally we apply the combination of 1st method and 3rd method. (Resize each frame before processing&Define small window for ball identification)

4.4.2. Tracking robust

Although the accuracy of ball tracking part mostly depends on the success of ball identification part, we still need to consider the tracking robust in case of the failure on ball identification part as well as improving the tracking efficiency.

4.4.2.1. Directly discard potential contours based on distance.

Extended from section 3 on ball identification algorithm, since we may find several contours (keypoints) candidates which may be the real specific ball position, if the distance between current contour center and last identified real ball center is larger than the predefined threshold, than we confidently think this is an incorrect identification and discard this candidate directly and check other candidates, which could also improve the tracking efficiency.

4.4.2.2. Predict ball center

Predict the ball center if all candidates failExtended from section 3 on ball identification algorithm, if all candidate contours fail according to the Euclidean distance threshold, then we need to predict a center for that specific pool ball. Currently, since we cannot obtain the next frames in real-time tracking, the simple way we adopt is to predict the center as a point on the straight line extended from the previous two center points. However, this could not be a good approach especially when the pool ball collides with other balls and the moving direction changes suddenly.

Below is one snapshot of real-time pool ball tracking:

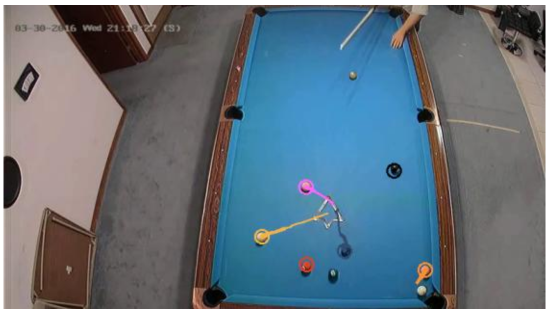


Figure 11. real-time tracking effect

4.5. 2D Projection and Simulation

In the real-time tracking step, the system would record all tracking balls' position history in the video. After tracking, the system can project all these records onto the 2D simulation board.

The key technique used in this process is *Homography*[8].

A ***Homography*** is a transformation (a 3×3 matrix) that maps the points in one image to the corresponding points in the

$$H = \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix}$$

other image. The matrix format is:

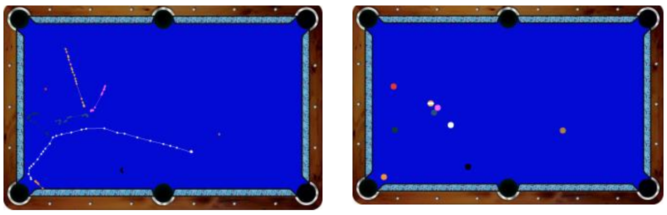
Let us consider the first set of corresponding points — (x1,y1) in the first image and (x2,y2) in the second image. Then, the Homography H maps them in the following way:

In our project, we need to calculate the homography from video table to 2d simulation table. Since we already get the four corners of table in the video, and simulation board's table

$$\begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} = H \begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix} = \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix} \begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix}$$

corners are easy to acquire, we can use them to calculate the homography by calling the OpenCV built-in function (cv2.findHomography). After getting the homography, the remaining job is very simple, just apply the homograph transform on each point in the video tracking record, to get the corresponding point on the simulation board. In the end, the system can use the points to build a simulation system, show the ball trajectory and even animation playback.

Here are some effect snapshots:



A. Ball movements trajectory B. One frame of animation

Figure 12. Projection and Animation

5. Test and Evaluation

5.1. Testing and Evaluation Overall

Our Evaluation includes these units:

- Table detection accuracy
- Ball detection and identification accuracy
- Ball tracking efficiency

Our testing data are mainly from 4 on-line game videos (Including 9 different camera positions/angles). To make datasets more sufficient and cover more cases, we split them into different clips, and use image processing technique to do some manual transform (rotation, brightness transform, resolution change, etc)

The real table/ball positions and real ball numbers are marked by us manually. We designed a system to help us to mark them by click with mouse.

5.2. Table Detection Accuracy

In this test unit, we evaluate system's automatic table detection.

Testing Data

different pool game image with table in it. The testing data covers:

- different table cloth color
- different brightness effect: dark/normal/light
- different illumination effect: normal/mixed

Testing Index

- True/False: if table can be found in the image
- Distance: average position difference of 4 detected corners refers to real corners.

Testing Result

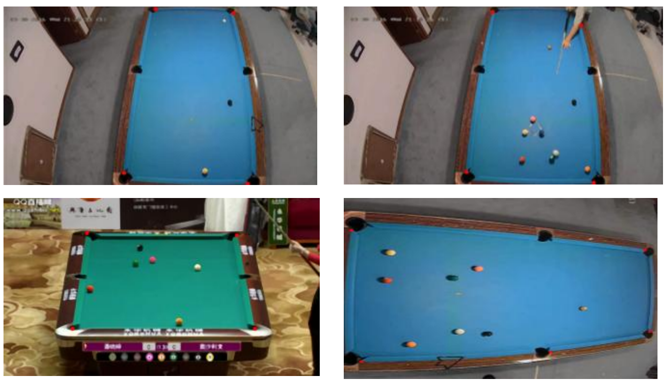
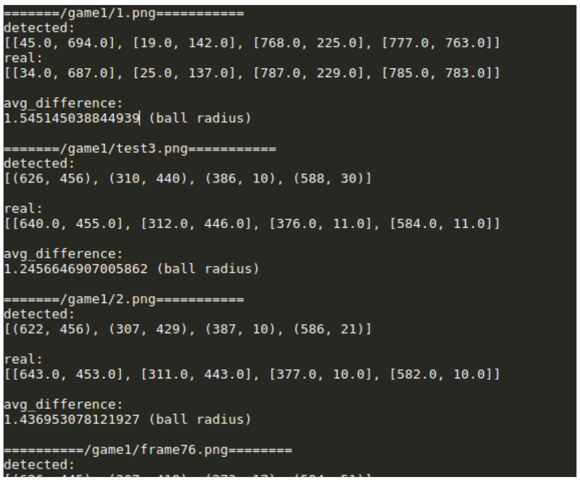


Figure 13.Table detection in different cases



```
=======/game1/1.png===========
detected:
[[45.0, 694.0], [19.0, 142.0], [768.0, 225.0], [777.0, 763.0]]
real:
[[34.0, 687.0], [25.0, 137.0], [787.0, 229.0], [785.0, 783.0]]

avg_difference:
1.545145038844939 (ball radius)

=======/game1/test3.png===========
detected:
[(626, 456), (310, 440), (386, 10), (588, 30)]

real:
[[640.0, 455.0], [312.0, 446.0], [376.0, 11.0], [584.0, 11.0]]

avg_difference:
1.2456646907005862 (ball radius)

=======/game1/2.png===========
detected:
[(622, 456), (307, 429), (387, 10), (586, 21)]

real:
[[643.0, 453.0], [311.0, 443.0], [377.0, 10.0], [582.0, 10.0]]

avg_difference:
1.436953078121927 (ball radius)

==========/game1/frame76.png========
detected:
```

Figure 14. Table detection testing result

We can see, for images that there are people or object occlusion, the result is not that good (average bias is nearly 2 ball radius), but for other case, the automatic detection works better (within 1.5 ball radius).

5.3. Ball Detection and Identification Accuracy

In this test unit, we evaluate system's ball detection and identification accuracy. Since in our project, ball detection and identification are implemented in one pass, we test them together.

Testing Data

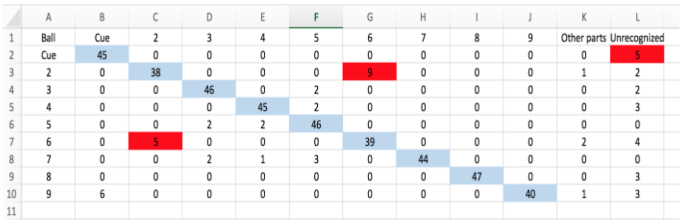
different pool game image. The testing data covers:

- Balls are placed apart. There are no balls attached together.
- Some ball are attached together. There are some cluster of balls.

Testing Index

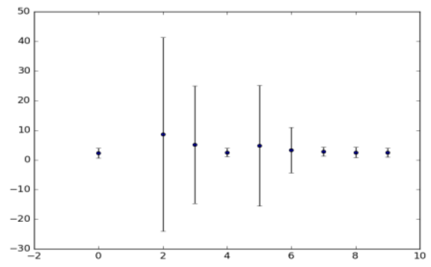
- Ball reorganization error analysis: use *Confusion Matrix* to show if balls are recognized correctly.
- Ball position accuracy analysis: use Error bar to show ball position accuracy.

Testing Result



	A	B	C	D	E	F	G	H	I	J	K	L
1	Ball	Cue	2	3	4	5	6	7	8	9	Other parts	Unrecognized
2	Cue	45	0	0	0	0	0	0	0	0	0	5
3	2	0	38	0	0	0	9	0	0	0	1	2
4	3	0	0	46	0	2	0	0	0	0	0	2
5	4	0	0	0	45	2	0	0	0	0	0	3
6	5	0	0	2	2	46	0	0	0	0	0	0
7	6	0	5	0	0	0	39	0	0	0	2	4
8	7	0	0	2	1	3	0	44	0	0	0	0
9	8	0	0	0	0	0	0	0	47	0	0	3
10	9	6	0	0	0	0	0	0	0	40	1	3
11												

A. Confusion matrix



B. Distance Error bar for balls

Figure 15. Ball identification result

From the above figures, cue ball is unrecognized in some cases especially when ball clubs is too closed or touch the ball, which might affect blob detection. Besides, ball 2 and ball6 are misclassified to each other in some cases because the color space of two balls is very similar even using Euclidean distance cannot distinguish them. Generally, ball identification works well (The average distance bias is within 1 ball radius).

5.4. Ball Tracking Efficiency

In this test unit, we evaluate system's ball tracking efficiency (time consumption).

Testing Data

- recorded process time of each frame. Classified by key frame and regular frame. (key frame and regular frame use different method)

Testing Index

- Time analysis

Testing Result

Average time consumed by the ball identification algorithm without resizing frames is about 0.0518 seconds(testing 100 frames),

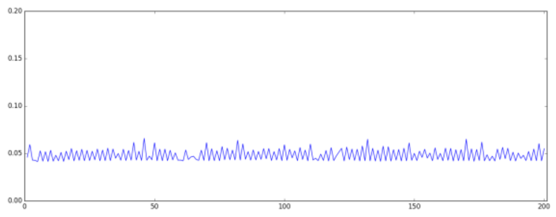


Figure 16. Time cost for each frame (without resize)

while applying resize, the average time decrease to 0.0382 seconds, which is acceptable.

6. Conclusion

We have successfully developed a system using python3 and opencv3 library to realize our goal. We use color features and contour detection techniques to recognize the pool table, and use blob detection, color features and color distance (Euclidean distance) to detect and identify the balls. For real-time tracking, we use some optimized methods mentioned above to improve the tracking efficiency and accuracy. Finally, we calculate the homography, project the ball movements from the video onto the 2D simulation plain board, and build a simple animation playback interface.

From the experiments, we can see the system get a decent performance. However, there are still some problems left. The main problem is about the ball identification accuracy in the real-time tracking. The identification would fail in some cases such as extreme brightness, the occlusions of other objects and ball in fast motion. To improve the system, we consider some potential methods:

- Apply Support Vector Machine to get optimal use of ball features to get higher identification accuracy.
- Apply other dynamic models in tracking such as Kalman Filtering[6] to correct the tracking result.
- Improve the projection and animation interface.

References

[1] Anne Kaspers, Biomedical Image Sciences Image Sciences Institute UMC Utrecht, Blob Detection, 2011.

[2] Mohammad Moghimi, Computer Science and Engineering Department University of California, San Diego, Using Color for Object Recognition, 2011.

[3] Peng Chang, The Robotics Institute Carnegie Mellon, John Krumm Microsoft Research Microsoft, Object Recognition with Color Cooccurrence Histograms, 1999.

[4] Samuel Bauza, Brian Choi, Chuanquiao Huang and Olga Souverneva, University of California San Diego, POOL-AID: Detection and Identification of Pool Balls for the purpose of recommending shots to beginner players, 2016.

[5] Pranjal Vachaspati Massachusetts Institute of Technology, A Computer Vision System for 9-Ball Pool.

[6] Erik Cuevas1,2, Daniel Zaldivar1,2 and Raul Rojas1, kalman filter for vision tracking, 2005.

[7] Anonymous, ICGVIP submission, ID–1466, Mean Shift Driven Particle Filter for Tracking Players in Sports, 2008.

[8] Jozsef Nemeth, Csaba Domokos, Zoltan Kato, Recovering Planar Homographies between 2D Shapes, 2009.