UNIVERSITY OF PADOVA

INP9087774 - COMPUTER VISION

Final project

Authors:
Simen Nesland and Jan Kristian Alstergren

July, 2024

Contents

List of Figures

List of Tables

1 Introduction

In this report we will explain the process and results of developing a computer vision system for detecting, segmenting and tracking of the balls in video clips of pool-games. The system is developed in c++ with OpenCV, and has the modules for table detection, ball detection/segmentation and a tracking module. As well as a for displaying the detected state of the game of pool. We will go through each module in this report.

2 Table Detector

2.1 Importance and simplifications

The goal of the project is to detect, segment and track the balls on a pool table. This means that everything outside the pool-table can be considered as noise. Therefore it is of great importance to detect where the table is, so that we don't look for balls outside the table. As we are allowed to assume that the table does not move during a clip, we can detect the table at the beginning of each clip, and then keep the region of interest for the whole clip. The simplification that we can detect the lines where the cloth meets the wood makes us able to easier detect the lines than if we were to detect the parts where the balls can be.

2.2 Approach

The initial thought when we read the problem was to use the Hough Line Transform to detect the four lines, and then concider the area between the lines as the table.

To try and enhance the lines of the table and reduce the noise in the image, a preprocessing was necessary. The preprocessing was done in several steps. First we removed the cloth coler by taking a color sample in the middle of the image and removing similar colors within a threshold. This is a risky manouver because one might miss the table or get balls in the sampled area. We were thinking about adding a "calibration part" where we ask the user to click on the table, but did not do it as the middle worked fine for the dataset, and then we did not have to click every time we ran the program. Also if the video is of a pool-table, we though a fair assumption is that a part of the table is in the middle. So the only real risk is that a ball affects the average color sample. The rest of the preprocessing was more safe with blurring and opencv's morphologyEx for noise removal.

When using the Hough transform we got more than the four lines we desired, but was able to tune both preprocessing and Hough so we got four "bundles" of lines at the borders of the table. To only get four lines, we simply chose one line from each bundle. This was done by starting at the first line, applying it to a unique-lines-vector. Then we loop through the rest of the lines, only adding them to the unique-vector if the angle or position was outside a threshold. So if it has the same angle, but large distance in position, it is probably a line in the bundle in the opposite side of the table. If it had large difference in angle, it would probably be a perpendicular line to the one we were looking at. After some tuning, this logic made us able to only choose four lines, one from each bundle, an example of this can be seen in 1.

The two last things worth mentioning about our table detector is that it makes an image that is black outside of the detected table. This is so that we can use the Hough transform in the detection of circles without being afraid of detecting circles outside the table. Also, because the ball detector detected false balls close to the edges of the table, we moved the intersections between the lines some pixels closer to the average point between the intersections. This helped against false positives near the edges, but for balls close to the edges with a camera with angle, it might lead to the ball not being detected.

2.3 Results table detection

The table detector ended up performing quite well, as we can see from 2, 3, 4 and 5. But as mentioned earlier the moving of the lines towards the middle could start hiding some balls around the edges. A case of this can be seen in the top right of 5, but this is a trade-off we have chosen to go for to detect less false positive around the edges.

2.4 Robustness of the approach

There is actually nothing stopping the system from detecting a line outside the table, and use it as part of the table. This means that if there is a clear line on the floor or in the background of the image, it could be detected as part of the table. Even thou the approach worked fine for the dataset provided, it is not hard to imagine an image where this approach could be insufficient. Then other ways of choosing what lines to use would have to be used.

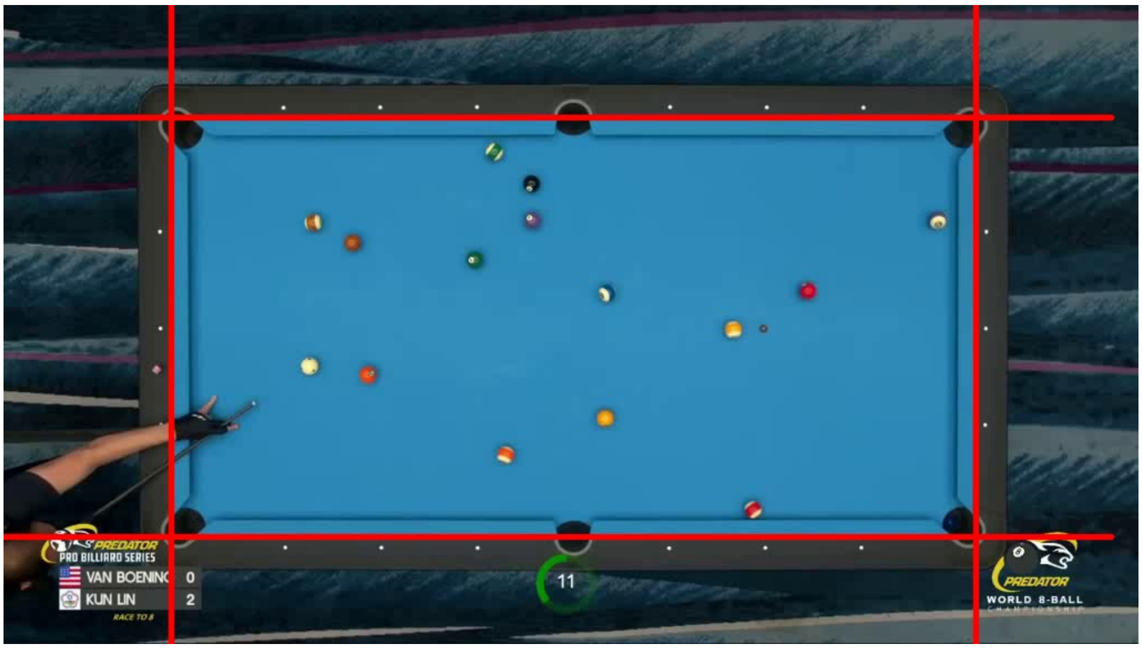


Figure 1: Detected lines of game 1 clip 1

3 Ball detection and segmentation

3.1 Detection Approach

To detect the balls, the plan was to use the Hough Circle Transform. Once again, a preprocessing the image was needed to get rid of noise and false positives. The chain of pre-processing we landed on after trial and error was: convert to grayscale, gaussian blur, equalize hist and then a Canny detector to detect the edges.

After all this we used the Hough transform. The main problem here was to detect all the balls while not getting too many false positives, we tried a lot of tuning and concluded that a quite strong blurring was good because it made the balls bigger and more evident in the Canny images which again made the Hough transform detection better. An example of the Canny image feeded to Hough can be seen in 16. It is from the first frame of game 1 clip 1 from the dataset. Even though the large smoothing removes noise from the table, one can see in 16 that there is still detected a lot of edges on the cloth. This image with edges is passed to the Hough circle transform, and in

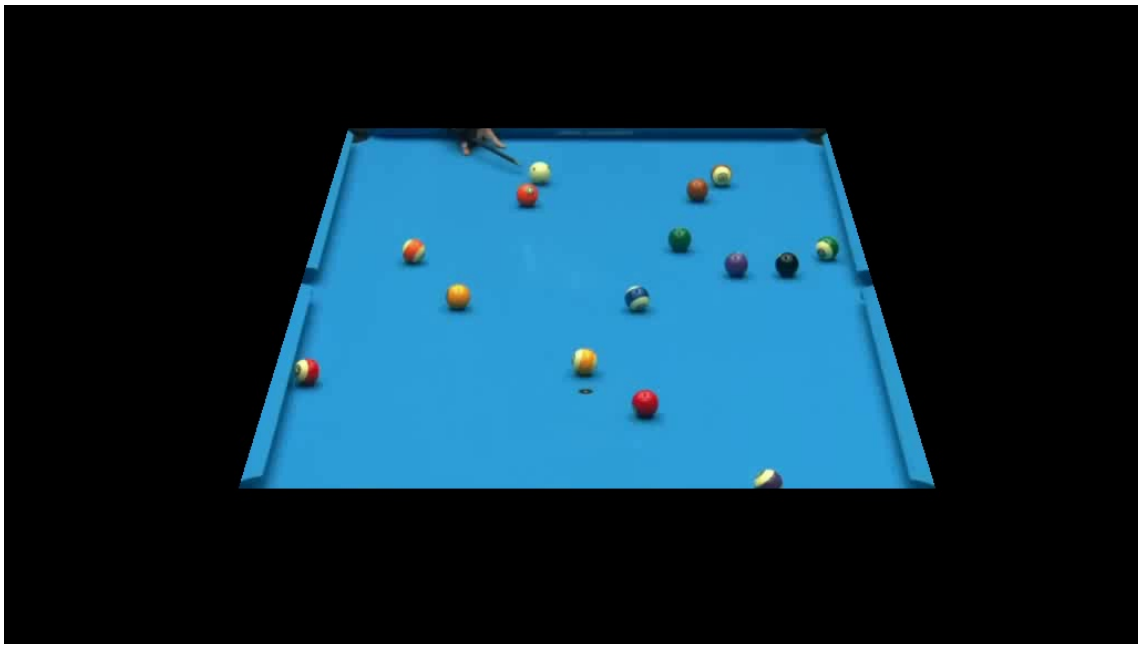


Figure 2: Detected table of game 1 clip 2

some cases this lead to detection of false positives. To get rid of some of the false positives while trying to miss as few real balls as possible, we decided to detect more balls (many false positives) in the Hough transform and then do a refinement of the detected balls.

The refinement consisted of deleting detected balls that had a similar color as the cloth. Also, as there were detected some balls along edges with parts of their bounding box outside the table, so we also removed circles that had more than a threshold of 50 pixels with value (0, 0, 0).

3.2 Segmentation Approach

After detecting the balls, the nest step was to segment them into black, white, solid and stripes. The approach we chose here was to first detect the white and black ball. Remove these from the ball-vector and then separate the striped/solid balls.

The segmentation of black and white balls from the rest was done by inspecting the color of the black and white balls in all the first-frames in the dataset. For a vector of the detected balls we then simply chose the ball with the most pixels within the range of inspected values.

Once we had segmented out the black and white balls, there were (in a perfect world) only solid and striped balls left. The idea was then to detect striped from solid by counting occurences of white pixels inside the bounding box of each ball. If there were more than a threshold white pixels, it is a striped ball, if not it is solid.

3.3 Results ball detection and segmentation

The results of the ball detection and segmentation are shown in figures 5 to 16, The mean IoU scores and the mAP scores are shown in Table 1. The results are not as impressive as we would have hoped. There are many false positives; hands and edges are being picked up on. Also there are many false negatives. Balls tight to the edges, near other balls or with less distinct colors were hard to segment.

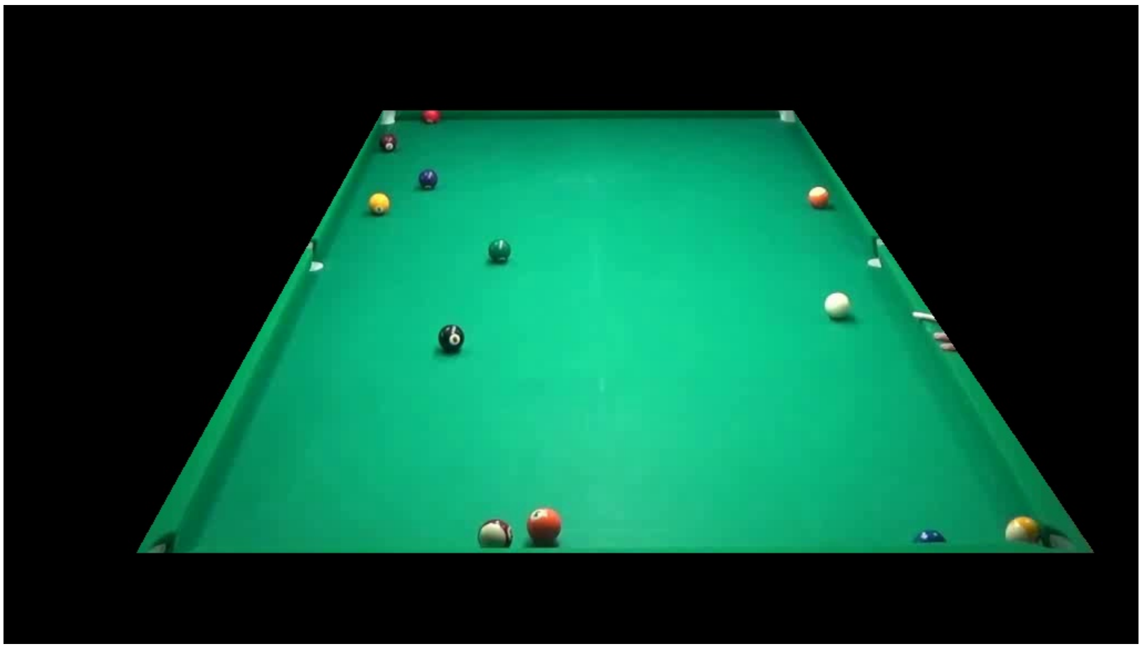


Figure 3: Detected table of game 2 clip 1

Clip Name	mAP First Frame	mAP Last Frame	IoU First Frame	IoU Last Frame
game1_clip1	0.0584416	0.0545455	0.221452	0.231008
game1_clip2	0.0584416	0.00757576	0.117778	0.117153
game1_clip3	0.020202	0.0	0.118632	0.11724
game1_clip4	0.00606061	0.010101	0.122551	0.11831
game2_clip1	0.424242	0.467532	0.141467	0.146247
game2_clip2	0.293706	0.347107	0.142422	0.145165
game3_clip1	0.477273	0.272727	0.133332	0.134636
game3_clip2	0.818182	0.545455	0.133533	0.134961
game4_clip1	0.181818	0.181818	0.0986815	0.0987914
game4_clip2	0.132231	0.324675	0.100452	0.0990029

Table 1: mAP and IoU results for each clip

3.4 Robustness

For the detection we saw that in 16 there were a lot of edges detected on the cloth, but in 17, which is the Canny image in the detection of game 3 clip 1, there detected less noisy edges. This shows that the system is affected by differences in lighting and/or some other factor that affects the algorithm. We were not able to find a way that with perfect accuracy detects all the balls without getting false positives. Maybe the problem was the approach, or it could be the tuning.

For the segmentation part, the white and black detection worked very well when white/black ball was detected in good way. In some cases the white ball was detected poorly, with the bounding box only containing half the ball, and this could lead to striped balls being detected as white.

The striped/solid segmentation approach was not working very well. We had problems tuning the value for how many white pixels should be needed to classify as a striped ball. Too low threshold lead to white-ish balls such as yellow ones and balls that was reflecting light was detected as striped. Too high threshold lead to too few striped balls. There were also a great variation of how many white pixels was detected in the different cases. A good threshold form one image, could be bad for the next. Maybe an approach using some kind of relation such as percentage of total pixels being white would be better to solve this problem.

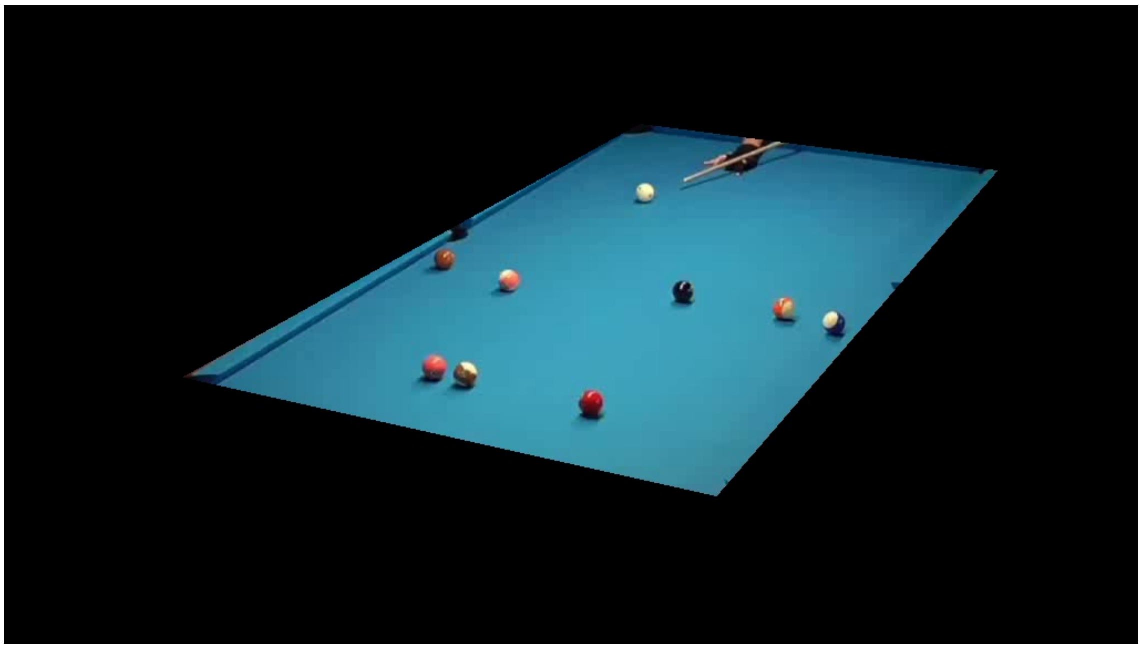


Figure 4: Detected table of game 3 clip 1

4 Tracker

4.1 Tracker approach

The tracker needed bounding boxes as input for each element it was supposed to track. A problem here was that once the player hit the ball, the ball moved too far between two frames, so the tracker was not able to follow the ball. If you put a bounding box on a slower moving object, the tracker worked fine for that object. Unfortunately we were mostly tracking fast moving objects, so we had to try to find a solution.

The first approach was to search for and try the different trackers that opencv provides. We landed on the TrackerBoosting, which worked better than the others in our tests. But still it was not able to track the balls that moved too fast, and we understood that the problem was that the faster balls are basically moving out of their box between two frames. We then tried to expand the bounding boxes so that the ball did not move out of the box between two frames, but too large boxes could make one box contain one ball, and a part of the neighbouring ball. This again lead to balls, especially the white one, ”stealing” other balls boxes which is unfortunate. Then it came to tuning to get the best trade-off

4.2 2D visualization approach

We used the getPerspectiveTransform from openCV to transform the coordinates to a 2D system. We updated a matrix with points where balls had moved and used houghLines to draw the trajectories

4.3 Results

The results were not impressive as a consequece of the segmentation. OBS: the final images are included in the output folder.

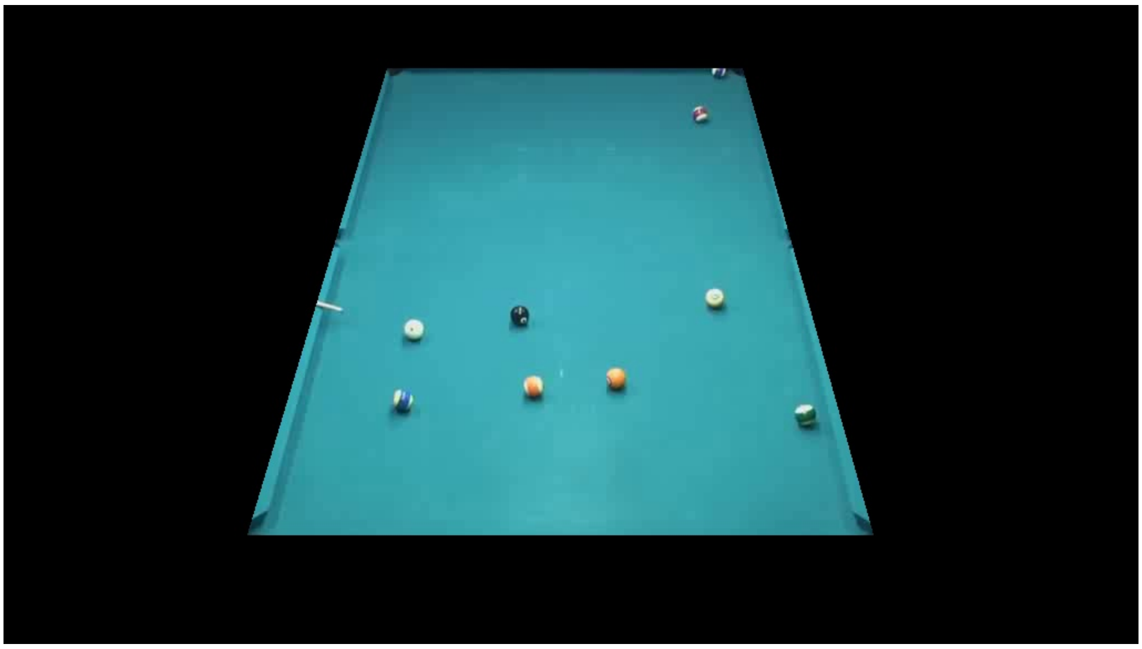
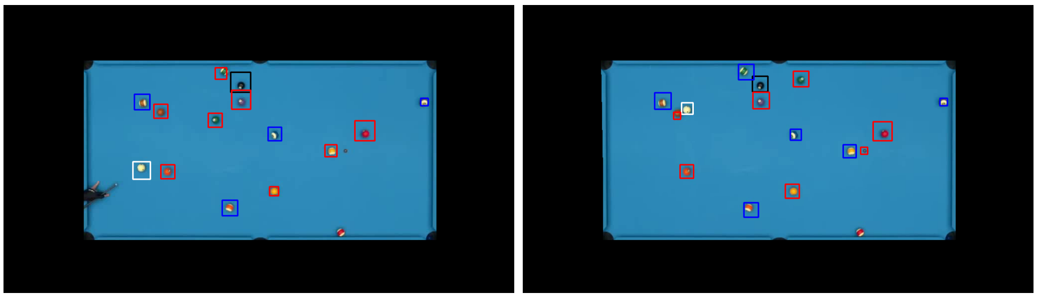


Figure 5: Detected table of game 4 clip 1

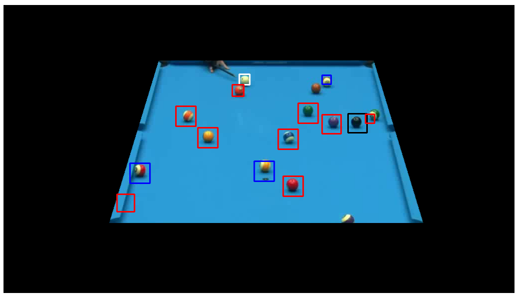


(a) First Frame (b) Last Frame

Figure 6: Output for game1_clip1

5 Contributions and working hours

The project and report is done by Simen Nesland and Jan Kristian Alstergren. Simen has had the main responsibility for the ball segmentation, tracking and the 2D-top view, while Jan has done the most on the table-detection and ball detection. The performance metrics were done together at the end of the project. When it comes to ideas, both have come up with most ideas at their own part. Working hours for both students are around 30 hours.


(a) First Frame

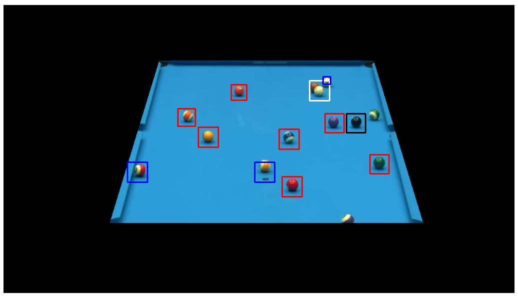
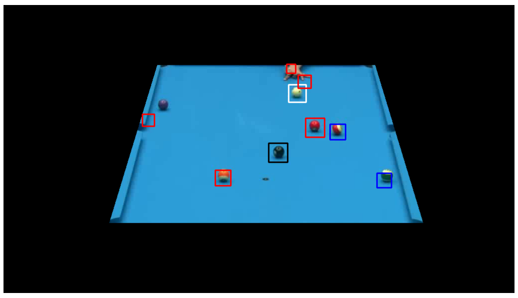

(b) Last Frame

Figure 7: Output for game1_clip2


(a) First Frame

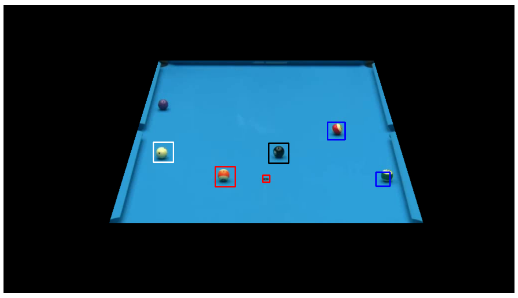
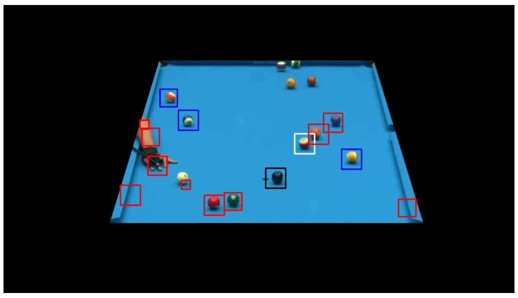

(b) Last Frame

Figure 8: Output for game1_clip3


(a) First Frame

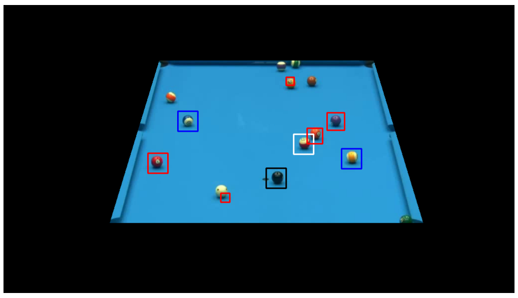
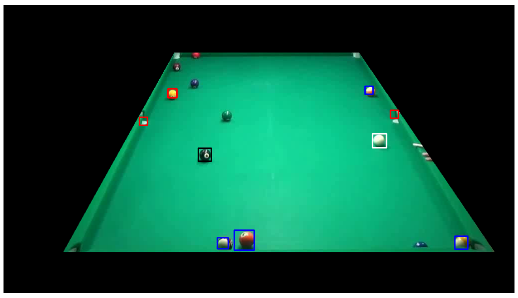

(b) Last Frame

Figure 9: Output for game1_clip4


(a) First Frame

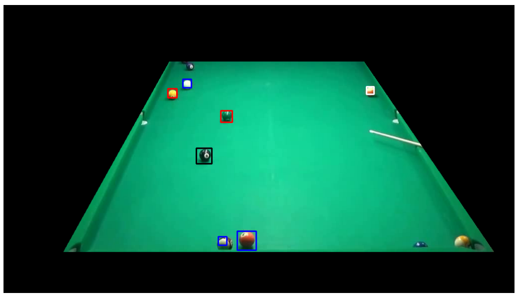
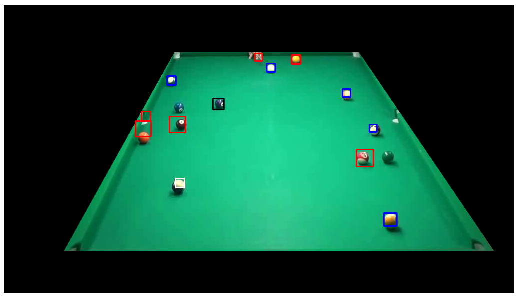

(b) Last Frame

Figure 10: Output for game2_clip1


(a) First Frame

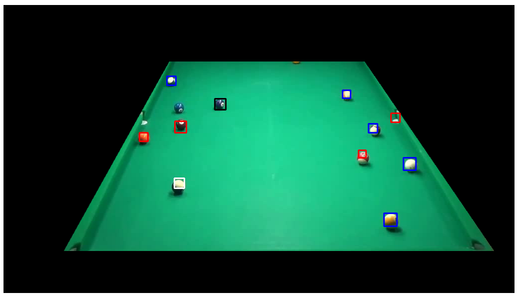
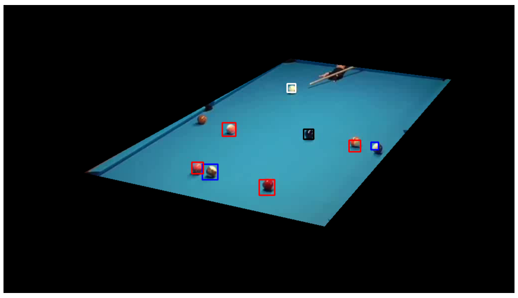

(b) Last Frame

Figure 11: Output for game2_clip2


(a) First Frame

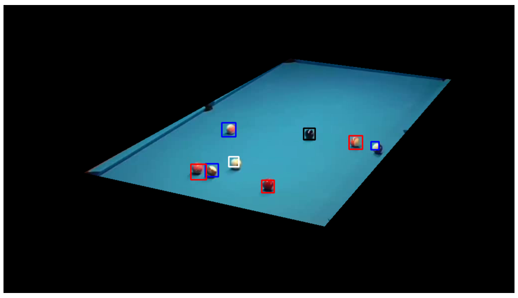
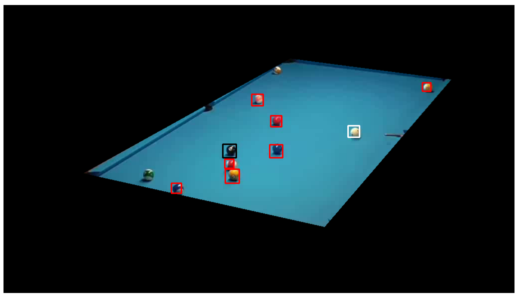

(b) Last Frame

Figure 12: Output for game3_clip1


(a) First Frame

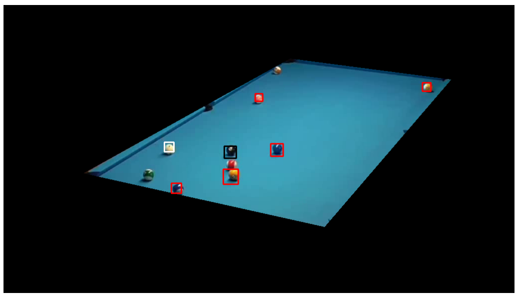
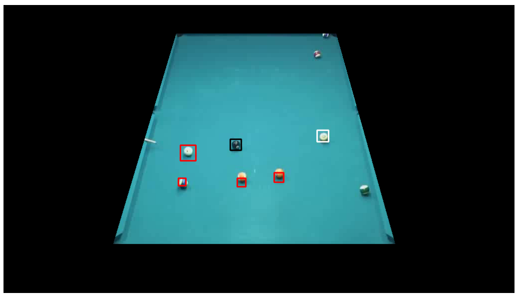

(b) Last Frame

Figure 13: Output for game3_clip2


(a) First Frame

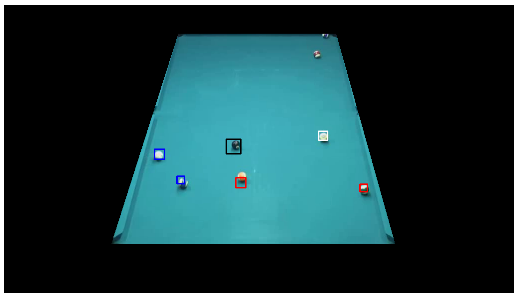

(b) Last Frame

Figure 14: Output for game4_clip1

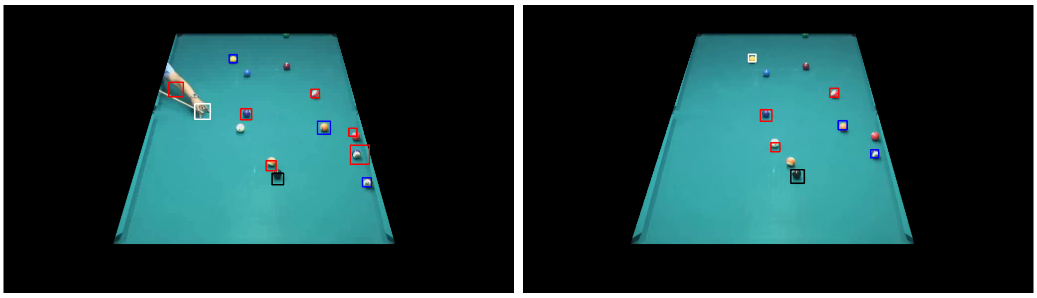

(a) First Frame (b) Last Frame

Figure 15: Output for game4_clip2

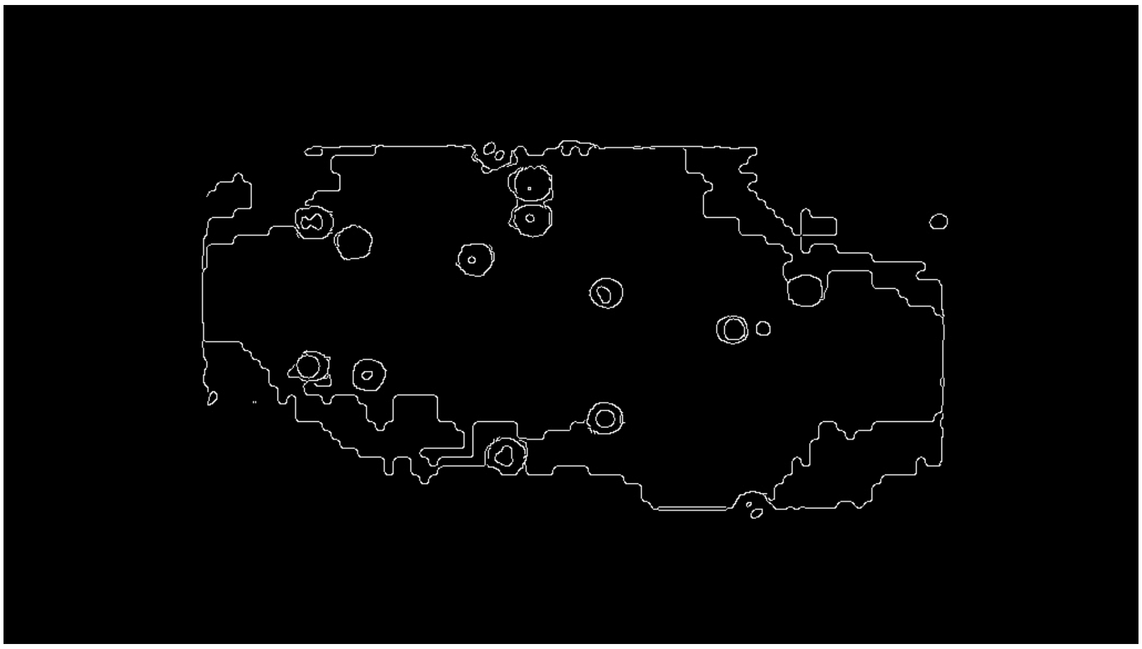


Figure 16: Detected edges from Canny edge detector in game 1 clip 1

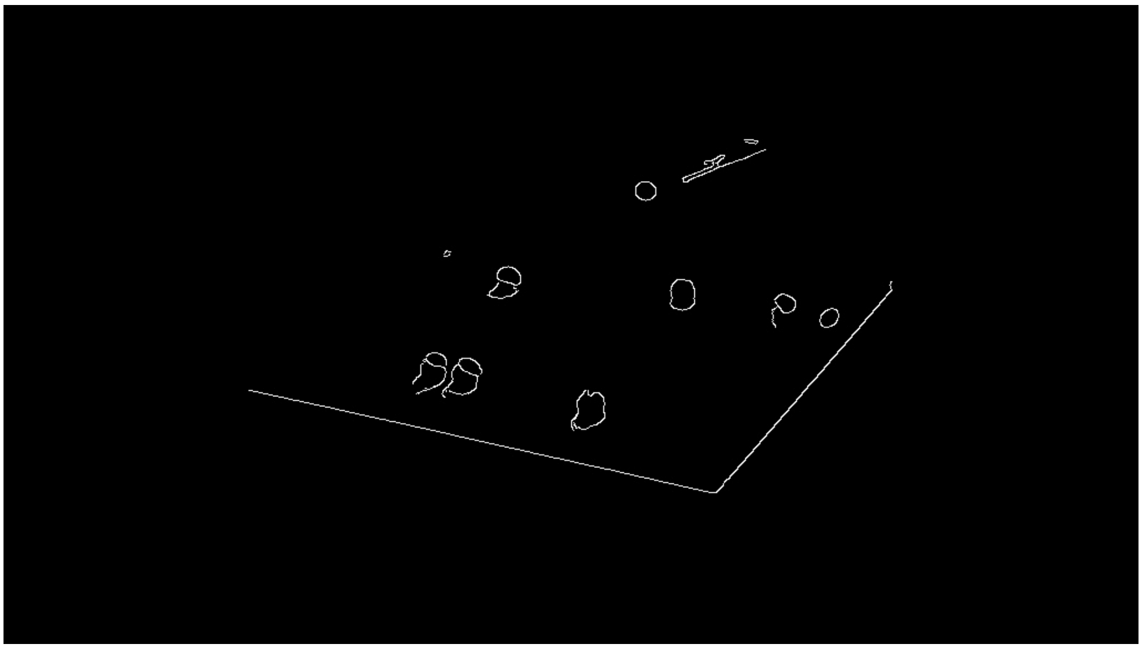


Figure 17: Detected edges from Canny edge detector in game 3 clip 1