Project Report: Shot Recommendation in Pool
Seoul National University - Computer Vision Project
Fall 2022

Louise Cuypers
Seoul National University
2022-81727

Guillaume Jarry
Seoul National University
2022-81990

Jonas Bernhard
Seoul National University
2022-82470

1. Introduction

In this project we implemented a shot recommendation system for the game of pool. Given an image of the state of a game of pool, our solution tries to, first, accurately parse the configuration of the current game state and, second, based on a certain strategy, give shot recommendations.

2. Background and related work

3. Methods

The shot recommendation problem comprises of two stages: first, the object recognition trying to infer the state of the game from the given image, and, based on this, the shot recommendation that tries to identify promising shots given some strategy.

3.1. Table detection

To be able to reconstruct the state of the game, the table has to be detected first. For a given input image, we are interested in the four corners of the pool table. The process to detect the position of the corners goes as follows:

First, the background of the table is removed using a deep learning model called rembg. The model is based on the U^2-Net architecture described in [1]. The background is remove d in order to avoid detecting random lines because of background clutter. More information on the tool and its implementation can be found on Github (https://github.com/danielgatis/rembg). An example of an image with a removed background can be seen in Figure 1.

Next, the image is blurred using a Gaussian blur in OpenCV to remove high frequency noise. This will make the background of the table itself more smooth. The used kernel has a size of 3 by 3 and the used standard deviation σ is set to 0.6. Then, the edges are detected using the Canny edge detection in OpenCV. The detection involves computing the intensity gradient of the image with a Sobel kernel. For the Sobel kernel, a size of 3 by is used. The detection then uses hysteresis thresholding to decided whether

Figure 1. Example of a background removal using rembg

an edge is a real edge or not. For this, the lower threshold is set to 40 and the higher threshold is set to 170. After the edge detection, the Hough Line Transform in OpenCV is used to detect straight lines. The resolution of ρ is set to 1 and the resolution of θ to 1 degree. The minimum number of votes to detect a line is set to 170. The value should be large enough, since we only want to detect longer lines, namely the edges of the table. The value should also be small enough, since we rather have multiple lines per edge instead of not detecting an edge at all. The result of the line detection gives too many lines, so they have to be clustered and filtered. Before the clustering, the range of the line parameters are changed so ρ is non-negative and θ is between 0 and 360 degrees. Afterwards, k-means clustering is applied three times in total to detect the four edges of the table. First, the lines are clustered into two groups by θ. This way, each group contains lines of two opposite edges of the table. Then, both groups are clustered into two groups by ρ.

This results in four clusters each representing an edge of the table. To get only the inner edges of the table, each cluster is sorted by ρ. For the clusters closer to the origin of the coordinate system, the line with the largest ρ is chosen, for the ones further away, the line with the smallest ρ is cho-

sen. This method assumes that there are no outliers and that all the detected lines come from one of the edges of the table. After the inner edges are determined, the only thing left is computing the intersection of every non-opposite pair of edges to get the four corners. This is done by transforming each line from polar to rectangular form and then applying Cramer's rule. An example of the table detection can be seen in Figure 2.

Figure 2. Example of a table detection. The red outline are the detected edges of the table.

3.2. Table rectification

After the four corners are detected using the method described in Section 3.1, we have to compute the homography to transform images coordinates to table coordinates. Using the fact that the table is rectangular and the real ratio between the width and height of a ordinary pool table is known (we use a height of 112 cm and a width of 224 cm), we can match a corresponding table point for each corner. This gives us four pairs of correspondence points. These are then used to compute the homography matrix using OpenCV. With the homography matrix, any point on the image can be transformed to table coordinates. The rectified game state of the running example can be see in Figure 3.

Figure 3. Example of a rectified table.

3.3. Ball detection

The next step for reconstructing the state of the game is to detect the position of the balls. Using the homography from the previous section, the process goes as follows:

Since we already have the input image without the background from the table detection, we will use this image again to avoid background clutter. The image is converted to a grayscale image and then blurred using a Gaussian blur in OpenCV to remove high frequency noise. The used kernel has a size of 3 by 3 and the used standard deviation σ is set to 0.7. Then, the Hough Circle Transform in OpenCV is used to detect circles. The used method to detect edges is set to Hough Gradient Method. The resolution of the accumulator is set to 1, so it has the same resolution as the input image. The minimum distance between the centres of two detected circles is set to 20. This way, we avoid a circle to be detected as multiple circles. The higher threshold of the Canny edge detector used in the Hough Gradient Method is set to 15 and the accumulator threshold to 18. Both values are rather low to avoid false negatives. The maximum radius of the circles is set to 30 to avoid random circles that are to large. After the circle detection, the result is filtered to only contain circles located on the table. The white ball is also detected during this filtering.

Using the homography matrix, every circle centre is transformed to table coordinates and circles that are out of range are discarded. For the other circles, the average image intensity of the surface of the circle is computed and the one with the highest average intensity is selected as the white ball. The following steps assume that the circle detection was able to detect the white ball. An example of the ball detection can be seen in Figure 4.

Figure 4. Example of the ball detection. Each red dot corresponds to the center of one detected ball. Each ball is outlined with a circle of its detected radius. The white ball is marked with a blue outline and the remaining balls are marked with a green outline.

3.4. Reconstruction of the game state

After the table and the balls are detected, we can reconstruct the game state. Using the fact that we know the real radius of the balls, we can project them on the table plane and create a reflection of the game state. An example of this can be seen in Figure 5.

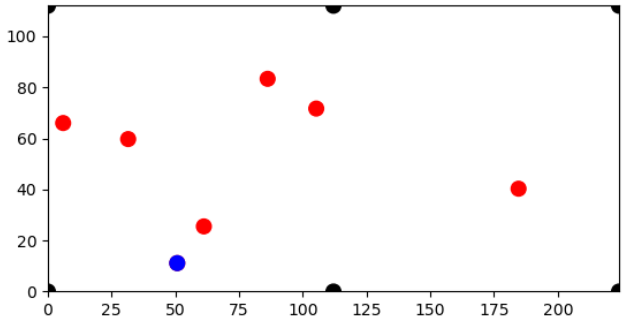


Figure 5. Example of the game state reconstruction.

4. Mechanics

To calculate the trajectory of the balls after impact, we need to look into the physics of the collision between two pool balls.

We assume that the collision between the two balls is inelastic (i.e. no kinetic energy is lost during the collision). While, this is not exactly correct (for example, the sound of the collision is in itself a transfer of kinetic energy into a sound wave), this provides a good enough approximation, especially when it comes to calculating the trajectory. Since the energy of the system of two balls is preserved, the quantity of the movement is also preserved. The relevant vector inequality is stated in Equation 1. A is the white ball, B is the ball struck, and the indices 1 and 2 signify the time just before and just right after collision, respectively. Because for ordinary pool balls, the mass of the balls is known and equal between the ball, we can simplify the masses in this expression. This expression is visualised in Figure 6.

$$m_A\overrightarrow{V_{1A}} = m_A\overrightarrow{V_{2A}} + m_B\overrightarrow{V_{2B}} \quad (1)$$

As the shots all aim to make the colored ball go into any of the holes, we already know the vector V_2B, as it simply is the normalized difference between the position of the hole and the position of the red ball. Then, because the two balls are sphere and their diameter is also known, we can calculate the center of the white ball at impact as well. It will be in the opposite direction of V_2B, a diameter of ball apart from the center of the red ball. And once we have the center at impact, we can compute the trajectory of the white ball necessary to strike the red ball in an appropriate way,

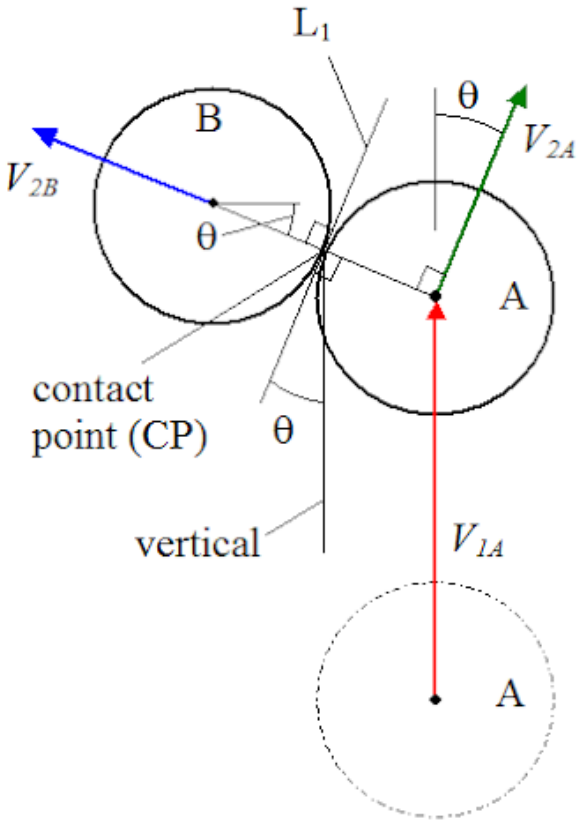


Figure 6. Visualisation of the mechanics behind inelastic shocks.

that is V_1A. V_2A is actually not necessary for this study. Indeed, a further study could implement the computation of the speed of the white ball ball after the impact, and where it will land next (counting the energy lost during the travel). Also, we currently only take direct shots into account and ignore possible shots that bounce of other balls or the table walls. An example of possible shots can be seen in Figure 7.

5. Shot recommendations

After enumerating the possible shots and the corresponding incoming and outgoing vectors we have to select a subset of shots that are promising. This set will serve as the shot recommendations for the current game state.

In order to rank the shots, we assign a score to each shot according to some strategy. Different strategies can be used for ranking the shots and which strategy is appropriate varies on the context. For example, if the player who wants to execute the shot is a beginner, different shots might be suitable than when recommending shots to an experienced player.

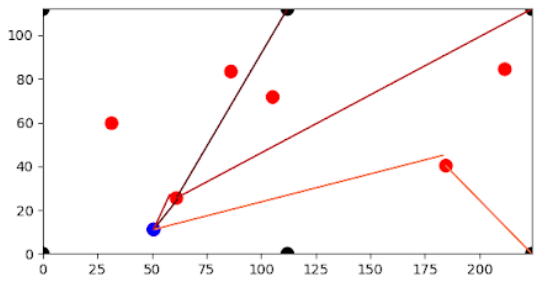


Figure 7. Examples of possible shots. Each shot has an incoming and an outgoing vector. The former represents the direction of the white ball before the hit, the latter is the direction of the colored ball after the hit.

Generally, shots that require a change in direction of the colored ball are harder to execute than a shot where the white ball, colored ball and hole lie on a straight line. One simple strategy that can take this into account is using the cosine similarity of the incoming and outgoing vector for ranking the shots. If the shot does not require any change of direction, both vectors will have the same direction and thus have a cosine similarity of 1. The larger the change of direction is, the smaller the cosine similarity is. Shots with negative similarity are even impossible to execute as they require a change in the direction of the incoming white ball.

Further, shots that cover larger distances are harder to execute because inaccuracies in the execution become more pronounced. Taking this into account, the score could punish shots where the distance from the white ball to the colored ball or the colored ball to the hole are larger.

One possible way to take multiple factors into account is using a weighted average. Appropriate weights have to be defined and could, for example, come from domain knowledge or learned from success rate of previous similar shot attempts.

So far, the discussed scores only consider a single round, i.e. one shot at a time, and ignore the resulting state after the shot. However, experienced players are usually required to plan ahead in order to set up an easy next shot or even multiple next shots. A more advanced strategy could calculate the scores over multiple rounds by considering the resulting game state. However, to allow a system to take into account multiple rounds, the system would need to extend the mechanics described in Section 4. For example, not only the direction of the balls but also the shot strength, subsequent movements and further collisions would need to be considered.

In our experiments, we use the cosine similarity as an example strategy. An example of shots scored using the cosine similarity can be see in Figure 8.

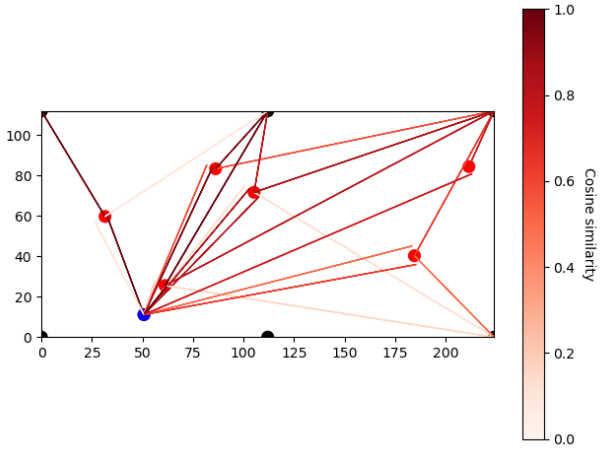


Figure 8. Example of shot recommendations. The color of the arrow represents the cosine similarity of the incoming and outgoing vectors of the shot (the darker the higher).

6. Experiments

We used multiple input images for evaluating our system. Figure 10 shows four images we used for our experiments. We used a set of diverse images covering multiple perspectives and and example of occlusion. Figure 11 shows the detected game state for the respective images. Most of the positions are detected accurately. However, two of the balls on the far end of the table are not detected correctly.

We observe that our table detection is robust to occlusion. The detected table for Figure 10c can be seen in Figure 9.

While the table detection is fairly accurate, we see that for views from an angle the ball detection can be prone to errors. Figure 12 shows an example of a false positive (the cue of the player is detected as a ball) and a false negative (the ball on the far end of the table is not being detected) in the same image.

Our experiments show that the detection gives decent results over varying images, even for images with less direct points of view, i.e. deviations from the easy case of the top down view. However, for the top view, we can see that the ball positions can be more accurately determined.

7. Conclusion

In this project we applied traditional computer vision methods learned in the course in order to capture the game

Figure 9. Example of table detection with occlusion.

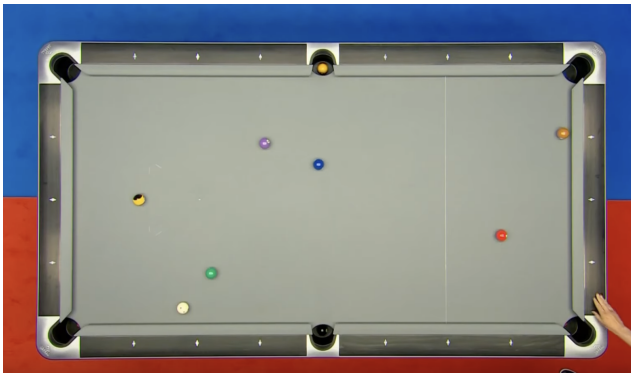
state of an active pool game. Further, we used laws from mechanics to enumerate possible shots and described simple strategies for scoring them.

As described in Section 4, future work could extend our methods by taking more elaborate mechanics into account and not only enumerating direct shots. This would allow for more intricate strategies and would make scores that consider multiple rounds possible. Also, currently the holes are modeled as perfect circles and a ball has to hit the center of the hole in order to be scored. However, in reality there is a wider path from the table surface leading to the hole which allows for more possible shots to be considered.

We showed that acceptable shot recommendation results can be achieved using just a single input image and traditional computer vision methods.

References

[1] Xuebin Qin, Zichen Zhang, Chenyang Huang, Masood Dehghan, Osmar R. Zaiane, and Martin Jagersand. U$\hat{2}$-Net: Going Deeper with Nested U-Structure for Salient Object Detection. 106:107404. 1



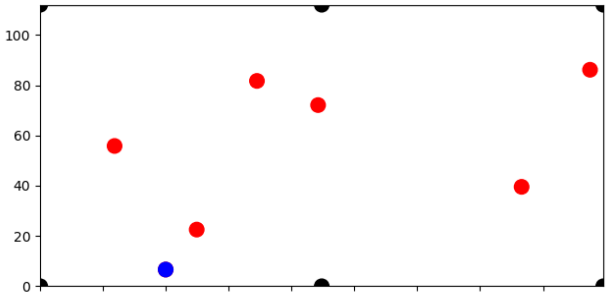
(a) Pool table viewed from the top

(b) View of the pool table from an angle

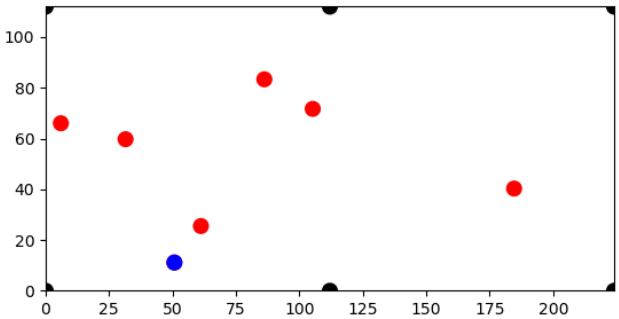
(c) View of the pool table from an angle with occlusion

(d) Side view of the pool table

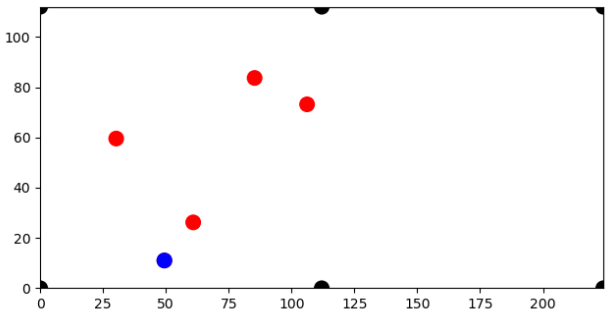
Figure 10. The input images used for evaluating our system. The images include multiple perspectives and occlusion.



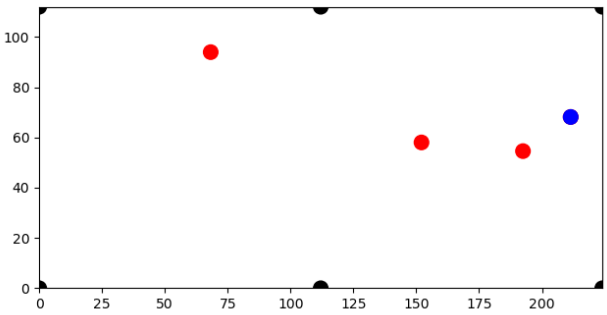
(a) Game state for pool table viewed from the top



(b) Game state for pool table from an angle



(c) Game state for pool table from an angle with occlusion.



(d) Game state of side view of the pool table

Figure 11. Detected game state for different input images.

Figure 12. Example of false negative and false positive in the ball detection. The cue of the player is detected as a ball, which is a false positive. The ball on the far end of the table is not detected, which constitutes a false negative.