pix2pockets: Shot Suggestions in 8-Ball Pool from a Single Image in the Wild

✉Jonas Myhre Schiøtt[1] Viktor Sebastian Petersen[1]
Dim P. Papadopoulos[1,2]
[1] Technical University of Denmark [2] Pioneer Center for AI
s204218@dtu.dk, s204225@dtu.dk, dimp@dtu.dk
https://pix2pockets.compute.dtu.dk/

Abstract. Computer vision models have seen increased usage in sports, and reinforcement learning (RL) is famous for beating humans in strategic games such as Chess and Go. In this paper, we are interested in building upon these advances and examining the game of classic 8-ball pool. We introduce pix2pockets, a foundation for an RL-assisted pool coach. Given a single image of a pool table, we first aim to detect the table and the balls and then propose the optimal shot suggestion. For the first task, we build a dataset with 195 diverse images where we manually annotate all balls and table dots, leading to 5748 object segmentation masks. For the second task, we build a standardized RL environment that allows easy development and benchmarking of any RL algorithm. Our object detection model yields an AP50 of 91.2 while our ball location pipeline obtains an error of only 0.4 cm. Furthermore, we compare standard RL algorithms to set a baseline for the shot suggestion task and we show that all of them fail to pocket all balls without making a foul move. We also present a simple baseline that achieves a per-shot success rate of 94.7% and clears a full game in a single turn 30% of the time.

1 Introduction

Artificial intelligence and reinforcement learning (RL) have proven to excel in complex games, such as Chess [30], Go [29], Starcraft [37], and Minecraft [21]. Apart from board and video games, computer vision models have recently started playing an important role in sports with several applications in generating sports analytics [13] and analyzing game strategies and tactics [36,24].

While RL in sports is not as widespread [34,36], table-based sports such as pool are a natural field for RL. The 8-ball pool variant is a popular game played worldwide by millions of people. According to Guinness records, the most downloaded mobile game is the 8-Ball pool game with 800 Million downloads [1]. Therefore, obtaining an easy-to-use framework for shot suggestions would be a helpful tool for both amateurs and professionals. Although the research on RL-based pool agents is limited, many approaches for the task exist [32,16,15].

[1] https://tinyurl.com/ystrrdu9

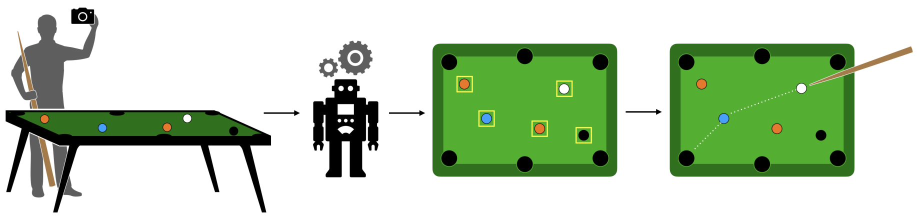


Fig. 1: **pix2pockets.** We introduce a new task for shot suggestions in pool games using a single input image. First, we detect the table and estimate the position of the balls. Then, they are fed into a pool environment, and a Reinforcement Learning agent predicts the best available shot (i.e., cue angle and shot power).

Several papers have proposed to detect the balls and edges on the pool table using a restricted setup [33,5,17,9]. All these approaches try to solve two separate tasks; first locate the balls on the table and then suggest a good shot. However, most of the approaches are limited to a single Bird's Eye View (BEV) of the table because their methods rely on a setup camera and traditional image processing techniques. This limits the training of such a model to a constrained situation, which is hard to obtain without any special equipment. Therefore, they fail to utilize modern deep learning models, which have shown great success in sports [28,39,4]. Consequently, to the best of our knowledge, an available dataset containing pool table images from various angles has not yet been established, which is needed for a model to work with images in the wild.

In this paper, we propose **pix2pockets**, a foundation for an RL-assisted pool coach. Given a single image in the wild, we analyze the situation on the table and suggest a good shot for pocketing the next ball. For the first task, we build an image dataset containing different angles and views of the table, allowing a trained model to analyze any user image. Our dataset contains 5748 manually annotated instances, with object bounding boxes for all balls and white dots [2] of the table. For the second task, we establish a standardized RL environment compatible with the widely used Gymnasium framework [3], which allows the easy training of any RL agent and the use of custom reward functions.

Our experimental results show that our detection model obtains a AP50 of 91.2%. Using these detections, we build a method to find the accurate locations of the balls and map them into our RL environment. We obtain a mean location estimation error of 0.4 cm corresponding to only 7% of the a pool ball diameter. Furthermore, we experiment with standard RL algorithms [27,18,8,19,11] on our developed RL environment. Even though they succeed in easy situations where only two pool balls are present, they fail to suggest a successful shot when all balls are present. We also present a simple algorithmic baseline that achieves a per-shot success rate of 94.7% and clears a full table in a single turn 30% of the

[2] https://www.libertygames.co.uk/pool-diamond-system/

time. We hope the release of the dataset and the environment will boost more work in the community for the game of pool.

2 Related Work

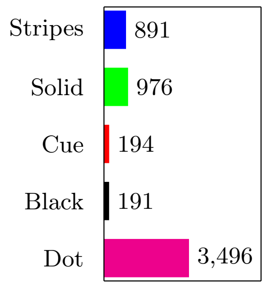
Computer vision for sports. Object detection systems are widely used in many sports such as football [28], basketball [39] and handball [4], for generating sports analytics [13], analyzing strategies [24], understanding broadcasts [10], and forecasting future actions [7]. For the game of pool, there are several approaches to detect balls and edges on a table [5,17,33,9,26,38,12]. However, they are limited to a single BEV of the table as they rely on a setup camera. This setup is hard to use in practice with images from various angles and lighting conditions. Moreover, these approaches use classic image processing based on thresholding [38], Hough transformation [26,17], and morphological operators [17]. Instead, we are interested in detecting pool tables and balls on images in the wild with diverse angles and lighting conditions.

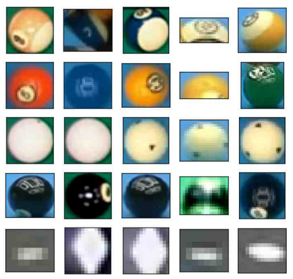
Learning to play board games. In 1996, the chess match between Garry Kasparov and the IBM computer DeepBlue was a milestone. Even this brute-force approach showed that artificial intelligence can catch up to human intelligence and defeat intellectual champions. Nowadays, most approaches rely on powerful RL models. One of the first examples is the 1992 IBM TDgammon [35], which used TD-lambda for playing backgammon. Recently, AlphaGo [29] and AlphaZero [30] have become increasingly advanced, surpassing human performance.

Learning to play video games. Other RL approaches focus on video games ranging from simple Atari games [20] like Pong or Space Invaders to complicated modern games like Counter-Strike [22], Minecraft [1], and Starcraft II [37]. An example of RL in video games is studied in [20], in which only the raw pixels from seven classic Atari games are used as input to an RL agent. This study shows that with most games, state-of-the-art methods could be implemented to achieve performance better or comparable to an expert human player [20]. Other papers [22,31] build RL agents that perform well in FPS games. One uses behavioral cloning from a large dataset consisting of Counter-Strike videos [22], while the other uses a modified Q-learning algorithm named RETALIATE [31].

Learning to play pool. Pool also requires strategy and outcome predictions. One attempt to create an agent for a pool environment is made in a series of YouTube videos [3]. However, this mainly showcases results, and the training implementation is limited. Another example is a playable pool implementation in pygame [4]. Other simulations in online games have the player compete against an AI, which often uses a set of predefined strategies, vector calculations, or search trees [16,32,15]. Instead, we implement a standardized RL environment which handles a variety of predefined agents using the Gymnasium library [3].

[3] https://github.com/packetsss/youtube-projects/tree/main/pool-game
[4] https://github.com/russs123/pool_tutorial





(a) **Dataset example images** (b) **Dataset statistics** (c) **Object examples**

Fig. 2: **Our Dataset.** (a) It contains 195 annotated images of tables captured from various angles with diverse lighting conditions. (b) We annotate 5748 objects with accurate segmentation masks. The maximum number of objects in the image varies from class to class. (c) Bounding box annotated examples. Note how sometimes the balls are not completely visible from the given view.

3 Dataset

We present our dataset collected to train object detection models in a pool game. The images are obtained from various online videos from different 8-ball championships. Most of these frames are taken from the *Predator World 8-Ball Championship* [5], as these are high-quality videos of many different angles filmed using BEV cameras and camera jibs. This allows us to gather diverse images with different views of the table. Example images are shown in Fig. 2a. We manually annotate the bounding boxes of all balls (cue, black, striped, and solid) on every image. To detect the table, we annotate the bounding boxes of white dots around the table. We found that this is a better choice than annotating the whole surface of the table, as the dots are located identically on all tables according to the *Diamond System*[2] and therefore uniquely identify the boundaries of the table. All annotations are obtained in Roboflow Annotate. Our dataset consists of two sets. The main dataset consists of 195 images (5748 bounding box annotations) used to train the detection model. The additional dataset contains 52 images (1624 annotations) of pool situations, where multiple images show the same table from different angles. There are 25 pool situations in total, which we use to study the projection error from view to view (Sec. 5.1). Examples of the annotated bounding boxes (balls and dots) are shown in Fig. 2c, while detailed dataset statistics are shown in Fig. 2b. Note that not all images have the cue ball, 8-ball or all dots, as they can be hidden behind the players in the images.

[5] https://www.youtube.com/@ProBilliardTV

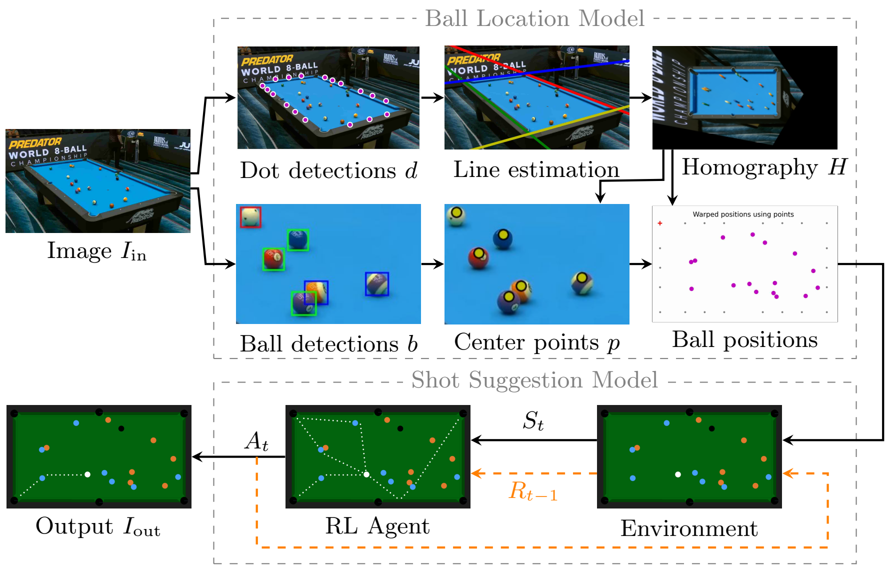


Fig. 3: **Full pipeline.** The input image I_{in} is run through the Ball Location Model to estimate the ball positions on the table, which is then handed to the Shot Suggestion Model. First, we obtain the dot detections d and the ball detections b on I_{in}. We use d to find the table lines and thus estimate a mapping H from I_{in} to a template T. Then, we use H to estimate the center point p for the balls b, resulting in the positions $\widetilde{p}$ for the environment. The Shot Suggestion Model sends the state $S \in \mathcal{S}$ to the agent, suggesting the $A \in \mathcal{A}$. During training, the environment evaluates the action, and the agent receives a reward $R \in \mathcal{R}$.

4 Method

We propose pix2pockets, a method for generating shot suggestions given a single input image I_{in}. The overall pipeline is depicted in Fig. 3. Our model can be split into two components: the Ball Location Model, and the Shot Suggestion Model.

4.1 Ball Location Model

The Ball Location Model provides precise ball locations given I_{in}. We use a predefined template, T, that shows a BEV of a pool table. The template T matches the dimensions in an RL environment, and the goal is to map the ball centers in I_{in} to T using an estimated homography matrix H. First, we use an object detection model to obtain the ball and dot detections on the table. Then, we perform a line estimation approach on the dots to ensure the table is rotated correctly and estimate H. We also use H to estimate the ball centers. Finally, we map the ball centers to T, and use them at the Shot Suggestion Model.

Dot detections. To locate the table in the image, we obtain a set of dot detections d. Pool tables follow the diamond system [2], which ensures that the dots will be in a special format, allowing us to learn the dimensions of the real table.
Line estimation. We use a linear line interpolation from d, to identify the four sides of the table, thereby dividing d into four subsets. We iteratively use Ransac [6] to find the line that interpolates the most points. We remove the inliers from each iteration and end up with the four lines (four table sides). To ensure that the line estimation is correct, we find all the intersections between them and make sure that only four intersection points are within the table. Furthermore, we find the intersections of these lines and add them to the set of points d. To estimate H, the order of the elements in d needs to match the order in T. This is achieved by sorting the four sets of points (one for each line) by their mean (x, y) values and then sorting the individual points within each set by x. After this, we check whether the first line contains 3 (short side) or 6 (long side) points since this indicates if we need to permute the ordering to ensure that the narrow table side is not mapped to the long side and vice versa.
Homography. Given the template T and found dots d, we estimate a mapping from I_{in} to T with H. In theory, we only need 4 points to estimate H, but we use all 22 available points to make the estimation more robust.
Ball detections. In parallel, we obtain a set of ball detections b from I_{in}. They are represented as bounding boxes (x, y, w, h, c), where (x, y) is the upper-left corner of the box, (w, h) is the width and height, and c is the predicted label.
Approximate ball center points. To map the balls b to T, we determine a single point p_i for each b_i. This point should be in the center of the bounding box for BEV images, whereas for angled images, lie closer to the top of the ball. To approximate p, we approximate the camera angle using H and map it linearly between the center point and the topmost point of the bounding box.
Ball positions. To correctly position the balls inside the environment we use H to map each p_i to a point $\widetilde{p}_i \in T$ used as ball positions for the balls in Sec. 4.2.

4.2 Shot Suggestion Model

The Shot Suggestion Model provides shot suggestions based on the given ball positions. We utilize RL to achieve this. For the RL agent to give shot suggestions, we create an environment in which it can gain experience.
Environment. The environment is built using the widely used Python module `Gymnasium` [3]. Here, we can define the layout and rules for which the RL agent can be trained. Our environment is created to be identical to a real pool table. We ensure this by using the dimensions of all elements from the top-view images from our dataset, such that the environment follows the actual pool table dimensions. The same applies to the size of the pockets relative to the balls and their locations. Due to the complexity of this task, we choose to simplify the physics as much as possible. We assume that the cue ball is always hit in its center and that every collision is perfectly elastic. We use the Python module `Pymunk` to simulate the physics. Since the environment is fully observable, the state $S \in \mathcal{S}$ contains the whole observation. That is $S_t = (x_1, y_1, c_1, ...x_{16}, y_{16}, c_{16})_t$, i.e., the

Table 1: **Ball and table detection.** The performance of the object detection model before and after post-processing (before | after). The post-processing step increases the precision by about 10%.

b | b'	Stripes	Solids	Cue	Black	Dots	Average
Precision	84.0 | 97.8	90.0 | 95.3	86.4 | 95.0	85.7 | 100	83.5 | 90.8	85.9 | 95.8
Recall	89.9 | 89.9	89.0 | 89.0	95.0 | 95.0	94.7 | 94.7	91.6 | 90.8	92.0 | 91.9
F1	86.8 | 93.7	89.5 | 92.0	90.5 | 95.0	90.0 | 97.3	87.4 | 90.8	88.8 | 93.8
AP50	90.0 | 91.4	89.9 | 90.4	92.2 | 92.3	92.2 | 92.4	89.5 | 89.3	90.8 | 91.2
AP50:95	79.1 | 80.4	80.7 | 80.9	83.6 | 84.0	84.9 | 85.4	51.0 | 51.4	75.9 | 76.4

position and class of all the balls at timestep t. The state space $\mathcal{S}$ contains all possible pixel locations inside the table boundaries for all balls, with the conditions that (a) the center of the ball cannot be closer to a cushion than its radius, and (b) any two balls cannot be closer to each other than their radii combined. To visualize and run the environment, we use the Python module `Pygame`.

RL agent. Given S_t at timestep t, the agent suggests a single shot as an action A_t. The action space $\mathcal{A} = \{\alpha, \rho\}$ where α is the direction and ρ is the power of the shot. During training, A_t is evaluated by $R : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{R}$ where $\mathcal{R}$ is the set of possible rewards from the reward function R described in Sec. 5.2.

Output. The model output is an image I_{out} showing the state S from the environment with a depiction of the suggested action A.

5 Experiments

In this section, we present our experimental results. First we present experiments for the Ball Location Model, and then for the Shot Suggestion Model.

5.1 Ball Location Model

We use a pre-trained YOLOv5 model [25], which we finetune to our dataset. We split our dataset into 155 training, 20 validation, and 20 test images. The images are resized to 640×640, and we use a batch size of 20, a constant learning rate of 0.01, and 2000 epochs, as fewer epochs lead to a model that can't distinguish the cue from the 8-ball. To evaluate the detection model, we use precision, recall, F1-score, and the average precision with an IoU-threshold of 50 (AP50).

Detection Results. The object detection results are shown in Tab. 1. We observe qualitatively that many of these errors are caused by detections that are not possible (e.g., detecting more than 18 dots or more than one cue). In other cases, balls are detected as multiple classes, resulting in a very high overlap of several objects. Since we know this can never happen, we constrain the model by post-processing the detections. We establish a post-processing procedure $b \rightarrow$

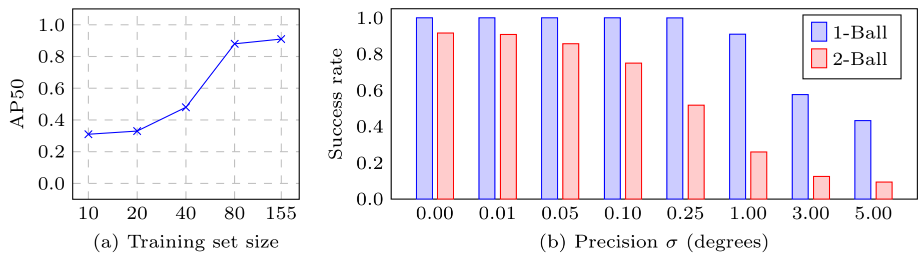


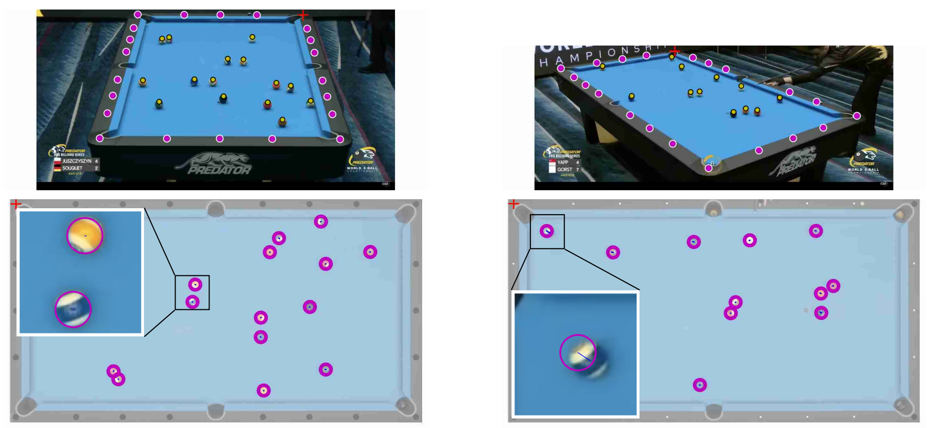
Fig. 4: **(a) Training size.** The AP50 of models with different training set sizes, showing diminishing gains after 80 images. **(b) Shot Accuracy.** To determine the shot precision, we test the performance for different σ values. In the 1-Ball environment, a precision of 0.25 degrees is enough to pocket the ball. Ball-ball interaction requires larger precision (0.01 degrees) when using additional balls.

b', that enhances the detection results. The post-processing first runs a class agnostic non-maximum-suppression (NMS) to eliminate detections with high overlap. Then, the highest confidence detections are kept while respecting the total number of class instances in an image (i.e., 18 balls). We see that the use of post-processing generally improves performance for all metrics. Due to the removal of detections, the post-processing sometimes removes correct but low-confidence detections, resulting in a lower recall, but with a precision increase of 10 percentage points, we obtain a more robust model.

Hough Circles. We compare our model with a traditional image-processing method (Hough Circles). We first apply an adaptive threshold on the image and then use the Hough Circles approach with $dp = 1$, $minDist = 10$, $param1 = 300$, $param2 = 0.7$, $maxRadius = 30$. The $minRadius$ is set to 0 for dot detections and 15 for balls. After detecting the circles, we perform an NMS using the radius as a confidence score. The Hough Circles achieve an AP50:95 of 0.26 without dots and 0.21 with dots, while our model, when disregarding classes, achieves 0.82 without dots and 0.67 with dots. In addition to being a better detector, our model can also classify them, which the Hough Circles is not able to.

Training size. To test the training performance using the dataset, we perform ablation on the size of the training set by training several YOLOv5 models with varying training set sizes, shown in Fig. 4a. We observe that AP50 is very low (<50%) when training with less than 50 images. Using 80 images yields much better results close to our full model, which yields an AP50 of 91%.

Table size. The further away one is from the table, the harder it is to distinguish the ball classes. We evaluate this by calculating the AP50 of each test image as a function of the table area in the image. The AP50 is above 95% when the table covers at least 40% of the image. When the table occupies a smaller area such that objects occupy less than 8×8 pixels, recognizing their categories becomes challenging, and AP50 drops significantly to about 50% in these few cases.



Best case. Mean table shift: 0.22 cm. **Worst case.** Mean table shift: 0.76 cm.

Fig. 5: **Projection error.** To estimate the projection error, the front-view and 45-view projections are compared to the top-view ground truth. The projection result is shown on the top-view image for accuracy assessment. The blue lines indicate the distance from the estimated center point p to the ground truth. The mean shift is compared to the table length in T, and scaled to a regular 9ft table.

Projection Error. To evaluate H, we establish a controlled experiment, where we use two different frames for the same pool situation: one BEV and another view using the additional dataset of 25 sets of situations from multiple angles. Here, we treat the BEV image as ground truth, as the template transformation is trivial. Then, we compare the results of the ball locations from the two estimated homographies, and the shift error is calculated as the distance from the ball centers using the BEV homography to the centers using the other homography. We obtain a mean error of 0.4 cm (good and bad examples in Fig. 5). We observe that the error is relatively small compared to the size of a real ball (5.7 cm).

5.2 Shot Suggestion Model

In this section, we discuss the specific choices taken in building the environment and then we present the results and evaluations of the trained RL models.
Environment settings. We set the screen size to 410 and 735 pixels. These lengths consist of the actual table itself, but also the width of the edges. The ball's radius is set to 7 pixels, and the pocket's radius to 15 pixels. Deducting the sides of the table, we have a table state space with $|\mathcal{S}| = 337 \times 662$ valid ball positions. When training, we generate states at random so that there is no overlap with neither another ball nor a cushion. As precision lower than 0.01 degrees has proven to result in lower performance, we use a multidiscrete action space with 36.000 equidistant angles and 30 values for the shot power matching the power of a shot in a real tournament, leading to an action space of size $|\mathcal{A}| = |\alpha| \times |\rho| = 36000 \times 30$. The target balls will be denoted "blue" balls.

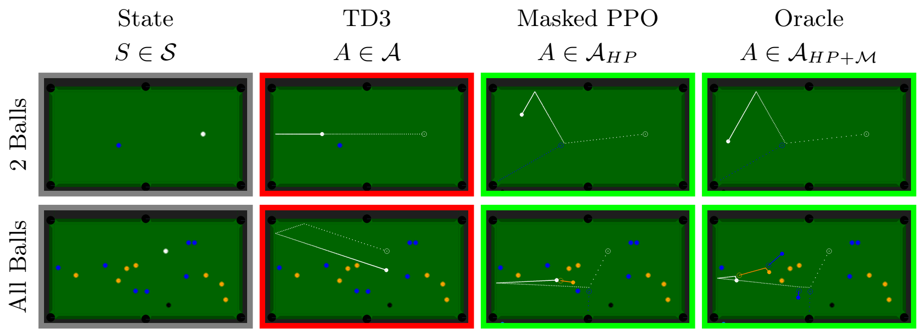


Fig. 6: **Example of shot suggestions.** The first column shows two initial states. The second, third, and fourth columns show the action predictions of TD3, Masked PPO, and the Oracle, respectively. Each action image is highlighted, indicating whether the shot was successful (green) or not (red).

Environments. We experiment with 3 environments: 1-ball, 2-ball, and All-ball. In 1-ball, only the cue ball is present, and the goal is to pocket it. In 2-ball, the cue and 8-ball are present, and the goal is to hit the cue ball and pocket the 8-ball. All-ball simulates a full game, and the goal is to first pocket all blue balls, and then the 8-ball, without mistakes. In all cases, the environment is reset if a shot is unsuccessful, whereas a new shot is awarded if a ball is pocketed.

Baseline models. We use standard RL algorithms provided in the SB3 library [23]. These include the Proximal Policy Optimization (PPO) [27], the Deep Deterministic Policy Gradient (DDPG) [18], the Twin Delayed DDPG (TD3) [8], the Advantage Actor Critic (A2C) [19], and the Soft Actor-Critic (SAC) [11]. In addition, we use Masked PPO [14], which masks out invalid actions from PPO.

Reward function. We reward the agent for winning the game by $+100$ and penalize it for losing it by -100. The agent must pocket all blue balls and then the 8-ball to win. To lose, the agent must pocket the cue ball or the 8-ball when any blue balls remain. To incentivize hitting the blue balls, the agent gets $+10$ for each blue ball hit and $+50$ for each blue ball pocketed. If a blue ball is hit, it gets a reward depending on the minimum angle between the hit balls' velocity and the pockets as described by $r(v) = 1000/(v + 10) - 50$, where v is the angle mentioned above. If the agent doesn't hit anything, it is penalized depending on the distance to the closest blue ball when all balls lie still again, as described by $r(d) = -50d/D$, where d is the distance, and D is the length of the diagonal of the table. If the agent pockets the cue ball, it is penalized by -80. Lastly, the reward is clipped to to the interval $[-210, 210]$ and normalized to $[-1, 1]$.

Hitpoints. We introduce a calculation of all possible directions for a successful shot that pockets a target ball. These directions are indicated by the point the cue ball needs to hit to deliver an impulse to the given ball in the correct direction. We refer to these points as hitpoints HP. Hitpoints can be used for action masking. By limiting the agent to shoot in directions that lead to correctly

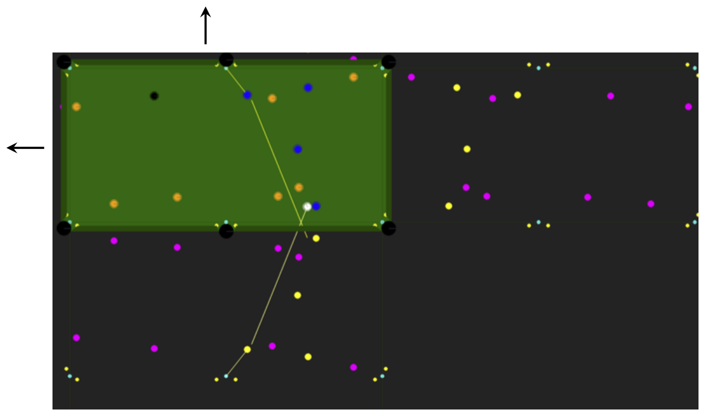


Fig. 7: **Mirror table.** To include kick and bank shots, we establish an approach to mirror the balls and pockets to a separate space on each table side. Aiming for a ball or pocket in the mirrored space $\mathcal{M}$ is equivalent to hitting a cushion and then a ball on the real table. The cyan points indicate the aiming points.

pocketed balls, we can significantly reduce the action space from the original $|\mathcal{A}| = 36000 \times 30$ to the subspace $|\mathcal{A}_{HP}| = |HP| \times 30$.

Oracle. We propose our Oracle model $\mathcal{O}$ to quantize the value of the hitpoints and determine the best option by two factors. First, a change in direction due to an angled collision makes a shot harder, and thus, we consider the cosine similarity of the proposed direction vector for the cue ball and the ball to hit. Second, we use the target window (the angle from the ball to the two sides of the pocket hole) as this determines the precision needed to fulfill the shot. The Oracle can suggest the best shot directly without using RL or quantize the shots for the reward function to advance training further.

Introduce Mirror table. While direct shots are typically easier, kick or bank shots (i.e., hitting the cushions during the shot) can offer better options. To include these options, we expand the hitpoint space $\mathcal{S}_{HP}$ by introducing mirrored table versions $\mathcal{M}$ on each side (Fig. 7). By mirroring the table, the cue ball direction can be pointed towards a ball in $\mathcal{M}$ for kick shots, resulting in hitting the cushion and then going to the desired ball in $\mathcal{S}$. Similarly, bank shots can be simulated by aiming at balls pocketing in $\mathcal{M}$. This expands the hitpoint set and improves the chances of finding a good shot. We also apply a penalty multiplier (0.33) for each cushion hit, as direct shots are easier to perform in reality.

Results. We train each baseline model at each environment for 500.000 timesteps using three-layer linear policy networks. We add action noise for algorithms with continuous action space (TD3, SAC, and DDPG) to help with exploration. All other hyperparameters are the default ones from SB3. In Tab. 2, we list the completion percentages for every model in each environment. A large gap between the 1-ball and the 2-ball environment indicates a significant difficulty spike between the two. No standard RL model manages to complete the all-ball envi-

Table 2: **Success-rate using various RL algorithms.** We train all baselines for one shot in the three environments, with little success in the latter two.

Env	Random	PPO	TD3	A2C	DDPG	SAC
1-ball	23.6%	84.6%	97.4%	35.0%	99.8%	88.6%
2-ball	0.90%	1.10%	6.30%	0.00%	5.50%	9.90%
all-ball	4.30%	12.8%	7.60%	10.6%	6.40%	5.90%

Table 3: **Success-rate using hitpoints.** Using the same environments, we use hitpoints (direct | direct,mirror) as action masks. We also test stages of using the oracle with randomness or evaluated maxima. In addition, we test the completion of a full turn (all blue balls and then the black ball).

Env	PPO_{mask}	$\text{PPO}_{\text{mask}}(\rho_{\max})$	$\mathcal{O}(\alpha_{\text{rand}}, \rho_{\text{rand}})$	$\mathcal{O}(\alpha_{\text{rand}}, \rho_{\max})$	$\mathcal{O}(\alpha_{\text{best}}, \rho_{\max})$
1-ball	100% | 100%	100% | 100%	84.3% | 86.2%	100% | 100%	100% | 100%
2-ball	63.2% | 68.5%	66.4% | 67.5%	65.9% | 34.4%	88.5% | 59.9%	95.2% | 98.7%
all-ball	49.7% | 60.7%	53.4% | 57.1%	68.7% | 37.1%	84.6% | 57.3%	90.2% | 94.7%
Full Turn	0.00% | 0.00%	0.00% | 0.00%	0.00% | 0.00%	1.50% | 0.30%	13.8% | 30.0%

ronment, although they pocket at least one ball 4-13% of the time. Masked PPO outperforms the other baselines, but only Oracle completes the all-ball environment with a 30% success rate. As a reference, a professional 8-ball player can "break and run" 39.2% of the time [2]. In Fig. 6, we observe that Masked PPO and Oracle succeed in contrast to TD3. This may be due to the large action space lowered for Masked PPO and completely removed for Oracle. The simple reward system may also be a reason for the low performance of the RL agents.
Ball shift effect. To measure the impact of the projection error on a good shot, we set up a 2-ball environment and make the Oracle find a shot. Then, we shift the target ball some distance in a random direction and let the shot play out. The distances shifted are sampled uniformly either from the mean shifts found in Sec.5.1 (shift-1) or a fixed distance of 2.5 cm (shift-2). We find that the Oracle using direct hitpoints drops from 97.06% success to 77.88% when the ball has been shifted by shift-1. However, when including the mirrored hitpoints, the drop is from 98.46% to 84.52%. For shift 2, the new percentages are 31.38% and 33.78%, respectively. This shows that the small projection error of our Ball Location Model does not change the results drastically, but a larger one will.

6 Conclusion

We presented a foundation for an RL-assisted Pool Coach named pix2pockets. Given an image, the model can suggest a shot by predicting the best direction and power. We built a dataset to enable the training of a pool-ball detector such that it can detect balls in any user image captured in the wild. We created simplified and open-source RL environments that provide standard benchmarks for training any RL algorithm. We show that pocketing the cue ball directly is easy, but pocketing all balls without making a foul move is much harder to accomplish. Our work marks a new direction for systematically benchmarking algorithms to solve the game of pool and assist in training amateur and professional players.

Acknowledgements. D. Papadopoulos was supported by the DFF Sapere Aude Starting Grant "ACHILLES".

References

1. Alaniz, S.: Deep reinforcement learning with model learning and monte carlo tree search in minecraft (2018)
2. Alciatore, D.G.: Break statistics (2024)
3. Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., Zaremba, W.: Openai gym (2016)
4. Burić, M., Pobar, M., Ivašić-Kos, M.: Adapting yolo network for ball and player detection. In: Proceedings of the 8th International Conference on Pattern Recognition Applications and Methods. vol. 1, pp. 845–851 (2019)
5. Chua, S.C., Wong, E.K., Koo, V.C.: Pool balls identification and calibration for a pool robot. In: ROVISP 2003: Proc. Intl. Conf. Robotics, Vision, Information and Signal Processing. pp. 312–315 (2003)
6. Derpanis, K.G.: Overview of the ransac algorithm. Image Rochester NY (2010)
7. Felsen, P., Agrawal, P., Malik, J.: What will happen next? forecasting player moves in sports videos. In: Proceedings of the IEEE international conference on computer vision. pp. 3342–3351 (2017)
8. Fujimoto, S., van Hoof, H., Meger, D.: Addressing function approximation error in actor-critic methods. CoRR (2018)
9. Gao, J., He, Q., Gao, H., Zhan, Z., Wu, Z.: Design of an efficient multi-objective recognition approach for 8-ball billiards vision system. Kuwait Journal of Science **45**(1) (2018)
10. Giancola, S., Amine, M., Dghaily, T., Ghanem, B.: Soccernet: A scalable dataset for action spotting in soccer videos. In: Proceedings of the IEEE conference on computer vision and pattern recognition workshops. pp. 1711–1721 (2018)
11. Haarnoja, T., Zhou, A., Abbeel, P., Levine, S.: Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor (2018)
12. Hsu, C.C., Tsai, H.C., Chen, H.T., Tsai, W.J., Lee, S.Y.: Computer-assisted billiard self-training using intelligent glasses. pp. 119–126 (2017). https://doi.org/10.1109/ISPAN-FCST-ISCC.2017.36
13. Huang, C.L., Shih, H.C., Chao, C.Y.: Semantic analysis of soccer video using dynamic bayesian network. IEEE Transactions on Multimedia **8**(4), 749–760 (2006)
14. Huang, S., Ontañón, S.: A closer look at invalid action masking in policy gradient algorithms. The International FLAIRS Conference Proceedings **35** (May 2022), http://dx.doi.org/10.32473/flairs.v35i.130584
15. Landry, J.F., Dussault, J.P.: Ai optimization of a billiard player. Journal of Intelligent and Robotic Systems **50**, 399–417 (12 2007)
16. Leckie, W., Greenspan, M.: Monte-carlo methods in pool strategy game trees. In: van den Herik, H.J., Ciancarini, P., Donkers, H.H.L.M.J. (eds.) Computers and Games. pp. 244–255. Springer Berlin Heidelberg, Berlin, Heidelberg (2007)
17. Legg, P.A., Parry, M.L., Chung, D.H., Jiang, R.M., Morris, A., Griffiths, I.W., Marshall, D., Chen, M.: Intelligent filtering by semantic importance for single-view 3d reconstruction from snooker video. In: 2011 18th IEEE international conference on image processing. pp. 2385–2388. IEEE (2011)
18. Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D.: Continuous control with deep reinforcement learning (2019)
19. Mnih, V., Badia, A.P., Mirza, M., Graves, A., Lillicrap, T.P., Harley, T., Silver, D., Kavukcuoglu, K.: Asynchronous methods for deep reinforcement learning. CoRR (2016), http://arxiv.org/abs/1602.01783

20. Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M.: Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602 (2013)
21. Oh, J., Chockalingam, V., Lee, H., et al.: Control of memory, active perception, and action in minecraft. In: International conference on machine learning. pp. 2790–2799. PMLR (2016)
22. Pearce, T., Zhu, J.: Counter-strike deathmatch with large-scale behavioural cloning (2021)
23. Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., Dormann, N.: Stable-baselines3: Reliable reinforcement learning implementations. Journal of Machine Learning Research **22**(268), 1–8 (2021)
24. Ramanathan, V., Huang, J., Abu-El-Haija, S., Gorban, A., Murphy, K., Fei-Fei, L.: Detecting events and key actors in multi-person videos. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 3043–3053 (2016)
25. Redmon, J., Divvala, S., Girshick, R., Farhadi, A.: You only look once: Unified, real-time object detection (2016)
26. Samarasinghe, S.M.: Development of Poolbot: an Autonomous Pool Playing Robot. Ph.D. thesis, Asian Institute of Technology (2016)
27. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. CoRR (2017), http://arxiv.org/abs/1707.06347
28. Seweryn, K., Chęć, G., Łukasik, S., Wróblewska, A.: Improving object detection quality in football through super-resolution techniques (2024)
29. Silver, D., et al.: Mastering the game of go with deep neural networks and tree search. Nature **529**, 484–503 (2016)
30. Silver, D., et al.: Mastering chess and shogi by self-play with a general reinforcement learning algorithm. CoRR (2017)
31. Smith, M., Lee-Urban, S., Muñoz Avila, H.: Retaliate: learning winning policies in first-person shooter games. In: Proceedings of the 19th National Conference on Innovative Applications of Artificial Intelligence - Volume 2. AAAI Press (2007)
32. Smith, M.: Running the table: An ai for computer billiards. In: Proceedings of the national conference on artificial intelligence. vol. 21, p. 994. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999 (2006)
33. Sousa, L., Alves, R., Rodrigues, J.M.F.: Augmented reality system to assist inexperienced pool players. Computational Visual Media (Jun 2016)
34. Sun, X., Davis, J., Schulte, O., Liu, G.: Cracking the black box: Distilling deep sports analytics. In: Proceedings of the 26th acm sigkdd international conference on knowledge discovery & data mining. pp. 3154–3162 (2020)
35. Tesauro, G., Murray, A.F.: TD-Gammon: A Self-Teaching Backgammon Program (1995)
36. Tuyls, K., Omidshafiei, S., Muller, P., Wang, Z., Connor, J., Hennes, D., Graham, I., Spearman, W., Waskett, T., Steel, D., et al.: Game plan: What ai can do for football, and what football can do for ai. Journal of Artificial Intelligence Research **71**, 41–88 (2021)
37. Vinyals, O., Babuschkin, I., Czarnecki, W.e.a.: Grandmaster level in StarCraft II using multi-agent reinforcement learning. Nature **575**(7782), 350–354 (2019)
38. Weatherford, S.: Pool cue guide (2013)
39. Yoon, Y., Hwang, H., Choi, Y., Joo, M., Oh, H., Park, I., Lee, K.H., Hwang, J.H.: Analyzing basketball movements and pass relationships using realtime object tracking techniques based on deep learning. IEEE Access (2019)