International Journal of Information Technology & Decision Making
Vol. 13 (2014)
© World Scientific Publishing Company
DOI: 10.1142/S0219622014500278

ANALYZING AND COMPARING SHOT PLANNING STRATEGIES AND THEIR EFFECTS ON THE PERFORMANCE OF AN AUGMENT REALITY BASED BILLIARD TRAINING SYSTEM

CHIHHSIONG SHIH
Department of Computer Science
Tunghai University, No. 181 Section 3
Taichung Harbor Road, Taichung, Taiwan
Republic of China
shihc@go.thu.edu.tw

Published 17 December 2013

The shot planning of a cue after it collides with an object ball determines a player's success in a billiard game. This paper proposes three novel gaming strategies to investigate the effect of cue shots planning on gaming performance. The first algorithm considers the nearest pocket for every selected target object ball, seeking optimal post collision positions. The second algorithm considers all pocket and target object ball combinations during both the pre- and post-collision optimal shot selection processes. The third algorithm considers a multi-objective optimization process for optimal shot planning control. The simulations are conducted based on a collision model considering the restitution effects. An augmented reality training facility is devised to guide users in both aiming and cue repositioning control in a real-world billiard game. Experimental results not only prove the reliability of our training device in selecting a proper shot sequence using the all-pocket optimal shot planning algorithm, but it also proves the consistency with the restitution theory.

Keywords: All pockets game strategies; multi-objective optimization; augment reality learning systems; greedy shots planning; video streaming.

1. Introduction

To achieve an acceptable proficiency level in billiards requires considerable practice. As this can be frustrating and unfruitful for beginners, and at times also for more advanced players, most research on computer billiards has focused on creating a highly competitive billiards playing program that competes against humans. This paper proposes an augmented-reality visual guide system that includes calculating the ideal speed for repositioning the cue ball based on a shot planning strategy. The cue placement after its collision with an object ball determines its success on successive shots. Deriving a better shot planning algorithm and implementing it using our tutoring system is a challenging task for players at various skill levels. Three novel gaming strategies are proposed to investigate the effect of cue placement on gaming performance. The strategies search for the post-collision path, targeting the optimal combination of object balls and pockets based on the multi-objective criteria of tolerance and attack angles. The first strategy considers the nearest pocket for every selected target object ball to search for optimal post-collision positions. The second strategy considers combinations among all pockets and target object balls during both the pre- and post-collision shot selection processes. The third algorithm considers a multi-objective optimization process for optimal cue repositioning control. The objective functions considered include the tolerance angle and the inverse of the attack angle. The goal is to find an optimal combination of both quantities. The simulations are conducted based on a collision model, considering the restitution effects. The restitution effect usually occurs in the collision occasion of two balls in the real world. It greatly affects the cue deflection path and should be considered in the simulation scenario. Different threshold values are tested to emulate the proficiency levels of different real-world players. Two different object ball distributions are also selected to test the different algorithms under different threshold values. The nearest algorithm performs only slightly inferior than the all-pocket algorithm in the sparse ball configuration, while the all-pocket algorithm greatly outperforms the nearest algorithm in the cluster configuration. The all pocket algorithm is thus considered suitable for different kinds of ball configurations in real-world gaming scenarios.

The guidance system is modified from a vision tracking system and a PC running a visual display[1] of the table states including balls positions, aiming instructions and cue stick orientation. The guidance system of this work displays the captured real-time pool table image together with the attack instructions, including aiming direction, ideal post-collision path, optimal cue position for the following shot, and cue stick velocity. Users can adjust both the aiming direction and hitting velocity of the cue stick, according to the guidance information analyzed by the all-pocket search strategy in this work. The all-pocket game strategy is selected to apply the maximum tolerance angle search sequentially on the first and subsequent shots towards all accessible pockets along the post-collision paths, considering the restitution caused deflections. These functions greatly enhance system usability, which is the new feature of this work.

Experimental results from the all-pocket shot planning strategy using our training facility, tested by users with different skill levels, outperformed the results without guidance for the same user set. Players with different proficiency levels exhibit different performance enhancements. The high-skill players exhibit lower degrees of enhancements, while low-skill players exhibit greater enhancements in different ball configurations. The performance statistics from high-skill players using the guidance system exhibit lower deviation errors than the simulation results for different ball configurations. This not only proves the reliability of our augment reality training device in guiding users towards optimal performance, but it also proves the consistency between the theory of shot planning strategies with the experimental results on table states.

Various computer billiard gaming systems have been devised to address players in both real and virtual game environments. These systems can be classified into two research categories. The first category involves a system that is an opposing counterpart to a player and creates a game playing scenario. Many billiard video games and billiard robots belong to this category. The second category is enhancing users' skills using various training gears. The following gives brief reviews for both categories. In the first category, researchers have focused on creating intelligent robotics[20,21] for entertainment, including robot golf[2], yo-yo,[22] volleyball,[23] chess,[4,24] and ping pong[25]. Various simulations and analyses have been exercised on different popular billiard games,[3,5–7,26] including 8-ball, 9-ball and snookers (see Refs. 8–11). Smith[19] applied artificial intelligence to develop a playing strategy for the 8-ball game. Leckie and Greenspan[10,11] further analyzed the billiard motions' physics and the nature of collisions and produced a physics engine for actual game development. These works contributed to the correct simulation and prediction of game playing results. However, no work actually contributed to the real-world billiards gaming environment. The smart strategy and precise analysis present no methods to help user enhance their enjoyment of the game. Lander[12] developed many billiard models considering ball-ball and ball-rail collision as well as slip-slide friction physics.

These are all important phenomenon happening in real-world billiard games, critical to developing a simulation program. Many video billiard games use these mathematical expressions in their game engines. They also lack further integration with strategy planning and instruction media to help users in real-world games.

The second billiard system category aimed to help players in real billiard game training. Jebara *et al.*[13] demonstrated that a wearable computer and augmented reality helps players enhance billiards games. A vision algorithm was implemented, which interactively operated with the user to assist planning and aiming. This system was similar to ours, as it analyzed table states in a greedy manner and its instructions can be applied to enforce a precise stroke. Delay in the head-mounted display, however, can cause dizziness. The cost is higher because it uses LCD goggles as a front-end media. The game strategies are not discussed and implemented in the front end to help users.

Larsen *et al.*[5] described the Automated Pool Trainer (APT), a pool training system developed at Aalborg University. It is a multi-modality system, utilizing spoken interaction with a graphical output and computer-controlled laser pointer as gaming guides. The trainee selected a suitable exercise from several predefined courses. The system then issued instructions on how to place the balls on the table for optimal shot recording and player performance evaluation. A human expert's experience helped plan the training courses. Using human expertise not only causes inconsistency in the training program but also raises the system cost. No multi-objective gaming strategies are analyzed and implemented in the guidance system.

Chou *et al.*[11] propose a framework to build a billiards tutoring system based on nine-ball video broadcasting analysis. They detect balls and trace their positions at every time instant. The real-world spatial relationships between the table and balls provide the aiming and position play suggestions. Ball position information is also utilized to distinguish each play into a corresponding event using a rule-based method.

Previous papers by this author[1,14] described a novel vision-based billiards ball tracking system designed to provide players with an interactive guiding system to orient the cue stick properly on the pool table. The major goal was to increase aiming accuracy during the hitting process to promote fun play without a complex electronic or mechanical setup on or around the playing table. The system achieves this goal by tracking the actual ball center positions and cue stick orientation on the table using a real-time vision system.

The target pocket and object ball are selected based on a preliminary reposition analysis algorithm for the best sequential shots.[1] The sequential shot algorithm considers an idealized reflection path after the collision of cue with an object ball rolling into the nearest pocket. The imaginary guide line is calculated based on the basic physical law of collision and changes its orientation around the cue ball according to the selected target pocket and object ball position. The user then moves the cue stick on the pool table, which the vision system traces. The cue stick centerline is represented by another imaginary line on the display system. The user then adjusts this center line by moving the cue stick to match the calculated ideal collision line from the cue ball. Once both lines align on the visual display, the user can strike the cue ball and watch the object ball roll into the selected pocket. This requires extra computational resources to track and display the cue stick center line. The aiming error easily stacks from the tracking and display cue stick calculations. The idealized reflection path, however, rarely occurs in an actual billiard game. The need to cope the physical rebound model within the various shots planning algorithms thus arises.

Table 1 compares the different techniques used by other relevant works, and the uniqueness of this work is bolded. This paper applies a restitution model to evaluate the post-collision deflection. Gaming strategies are designed and tested for optimal performance on different pool states and player proficiency levels. The unique strategies studied by this work include an all pockets and the multi-objective shots planning algorithms. Optimal placement control of a cue after collision is then possible using these simulations when hitting a cue ball. The way in which a user learns about the hit stroke is also unique. The visual display of the analyzed instructions from our optimal game strategy is also unique. The ghost ball and optimal post-collision cue positions are displayed on a PC monitor, thus enhancing the users' fun and skill using the augment reality techniques. By merging these instructions with the real-time image of the billiard table, the cue stick does not have to be traced. This not only reduces the computation load of the gaming console, but also provides a more intuitive operation mode for users using the system.

This paper introduces the shot difficulty criterion as the basis of the shot planning strategies in Sec. 2. Section 3 presents a restitution effect on the colliding cue and object balls. The analysis is compounded within the search process on the

Table 1. Comparisons of relevant techniques.

Index	Work	Chua *et al.*[7,15,16]	Jebara *et al.*[13]	Cheng *et al.*[8]	Larsen *et al.*[5]	Nakama *et al.*[17]	Leckie and Greespan[9,10]	Greenspan *et al.*[18]	Smith[19]	Chou *et al.*[11]	Shih[1]	Shih (This work)
Vision tracking	Ball center	Y	Y	Y	Y	Y	N	Y	N	Y	Y	**Y**
	Cue stick tracking	N	N	N	N	N	N	Y	N	N	Y	**N**
Strategy Analysis	Fuzzy Logic	N	N	Y	N	N	Spin Anal.	Y	N	N	N	**N**
	Multi-objective + All pockets optimization	**N**	**N**	**N**	**N**	**N**	**N**	**N**	**N**	**N**	**N**	**Y**
	Restitution Model	**N**	**N**	**N**	**N**	**N**	**N**	**N**	**N**	**N**	**N**	**Y**
Service target	Robot	Y	N	Y	N	Y	Y	Y	Y	N	N	**N**
	Human	N	Y	N	Y	N	N	N	N	Y	Y	**Y**
Instruction media	PC Monitor	N	N	N	N	N	N	N	N	N	**Y**	**Y**
	LCD goggle	N	Y	N	N	N	N	N	N	N	N	**N**
	Laser pointer	N	N	N	Y	N	N	N	N	N	N	**N**
	image streaming	**N**	**N**	**N**	**N**	**N**	**N**	**N**	**N**	**N**	**N**	**Y**
Hardware Cost & Complexity		High	High	High	High	High	Low	High	Low	Low	Low	Low

post-collision path for the shot planning algorithms discussed in Sec. 4. Section 5.1 details the guidance system, including the data flow and visual interface, which repeatedly updates the screen from the pool table image stream. Given an optimal strategy, an estimated travel distance is evaluated for the best possible successive shots. Given this distance on the table, an initial velocity to drive the cue stick is inversely calculated using a least square transformation matrix in cubic form, as introduced in Sec. 5.2. This velocity is shown on the visual display as a guide for users to drive the cue stick consistently. Sections 5.3 and 5.4 present simulation results and those from the guidance system applying the shot planning algorithms, respectively. Section 6 summarizes the comparison results and the performance analysis of the shot planning algorithms.

2. Shots Difficulty Measurement

This section describes the criterion used to optimize the shot planning strategies. This criterion has been discussed by Jebara *et al.*[13] and is briefly stated here. Given a cue ball, we find an accessible path to an object ball and another accessible path from this object ball to the target pocket. For a planning strategy to succeed, one must find the optimal angle at which to hit a cue ball, i.e., angle c, as a deviation from the line connecting the cue ball to the solid object ball. A shot is harder when less tolerance (angle d) is needed on angle c. A smaller value for angle d makes it harder for a low-skill player to render the shot into the target pocket. Conversely, a high-skill player can manage the error to be within the tolerance zone with ease and make the shot. This is the basic theory for our shot planning algorithm.

$$b = a\sin(R/L), \tag{1}$$

$$c = a\sin\left(\frac{2r\sin(a)}{\sqrt{4r^2 + l^2 - 4rl\cos(a)}}\right), \tag{2}$$

$$d = a\sin\left(\frac{2r\sin(a+b)}{\sqrt{4r^2 + l^2 - 4rl\cos(a+b)}}\right) - c. \tag{3}$$

3. Rebound Restitution Effect Analysis

For an optimal sequential shot algorithm to find the best combination of pockets and object balls to sink, the correct rebound path of the cue after collision with selected object balls must be derived. We use linear collision physics and consider the restitution factor[12] to predict the cue ball motion after its collision with a selected object ball. In an ideal collision occasion of two rigid body balls, the cue deflection is always perfectly 90° from the rolling direction of the object ball towards the target pocket. However, the collision between two billiard balls involves both elastic and plastic ball body deformation in a real-world gaming scenario. The deflection direction of the cue after its collision with an object ball is analyzed first considering a restitution factor

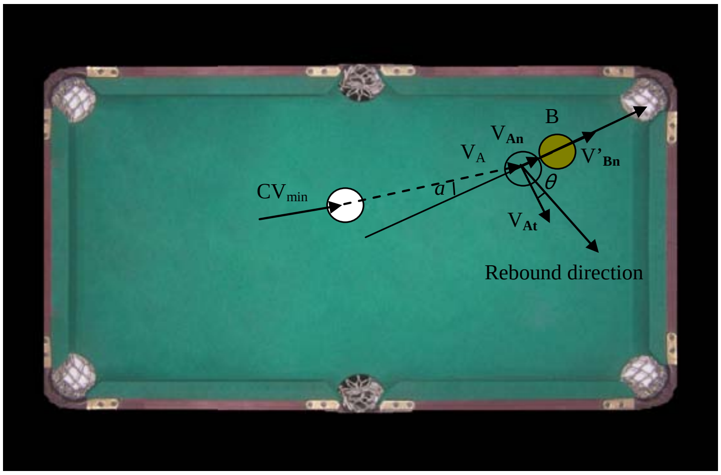


Fig. 1. Restitution relation notations.

that usually occurs in a real collision situation. The derivation below indicates that the deflection is not 90° from the rolling direction of the object ball towards the target pocket, as in Fig. 1. In Fig. 1, there are two balls: A is the cue ball, and B is the object ball. We intend to analyze the post-collision cue ball velocity. We first break the incident velocity of the cue ball, V_{A}, into its components along the collision line, $V_{\mathrm{A}n}$, and the tangent to the collision, $V_{\mathrm{A}t}$, where $V_{\mathrm{A}n} = V_{\mathrm{A}}\cos(a)$ and $V_{\mathrm{A}t} = V_{\mathrm{A}}\sin(a)$; '$a$' is the attack angle. The pre-collision velocity of the object ball, V_{B}, has zero components along both the collision line, $V_{\mathrm{B}n}$, and the tangent to the collision, $V_{\mathrm{B}t}$, i.e., $V_{\mathrm{B}n} = 0$, and $V_{\mathrm{B}t} = 0$.

The impulsive force acting during the collision is directed along the collision line. The t component of the velocity of each ball after its collision thus does not change, i.e., $V'_{\mathrm{A}t} = V_{\mathrm{A}} \times \sin(a)$ and $V'_{\mathrm{B}t} = 0$. Because the impulse forces are equal in magnitude and opposite in direction, momentum is conserved before and after the collision[12]:

$$m_{\mathrm{A}}(V_{\mathrm{A}n}) + m_{\mathrm{B}}(V_{\mathrm{B}n}) = m_{\mathrm{A}}(V'_{\mathrm{A}n}) + m_{\mathrm{B}}(V'_{\mathrm{B}n}), \tag{4}$$

$$m_{\mathrm{A}}(V_{\mathrm{A}n}) + m_{\mathrm{B}}(0) = m_{\mathrm{A}}(V'_{\mathrm{A}n}) + m_{\mathrm{B}}(V'_{\mathrm{B}n}). \tag{5}$$

Since $m_{\mathrm{A}} = m_{\mathrm{B}}$,

$$(V'_{\mathrm{A}n}) + (V'_{\mathrm{B}n}) = V_{\mathrm{A}}\cos(a). \tag{6}$$

This equation cannot be solved without more information. We introduced the restitution factor,[12] ε, which is the scalar value between 0 and 1 relating the bodies' velocities before and after a collision using the following equations. $V_{\mathrm{B}n}$ is zero before collision.

$$V'_{\mathrm{B}n} - V_{\mathrm{A}n} = \varepsilon(V_{\mathrm{A}n} - V_{\mathrm{B}n}), \tag{7}$$

$$V'_{\mathrm{B}n} - V'_{\mathrm{A}n} = \varepsilon(V_{\mathrm{A}}\cos(a) - 0). \tag{8}$$

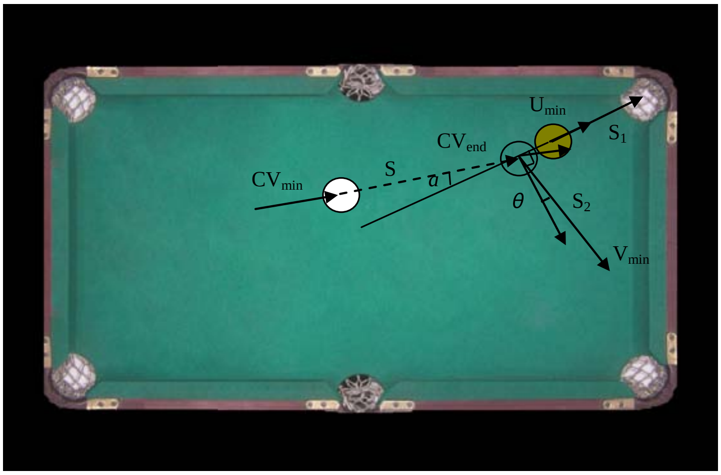


Fig. 2. Post collision physics schematic and notations showing minimum driving speed calculation.

Solving Eqs. (6) and (8), we get the velocities of the two billiard balls after the collision.

$$V'_{\mathrm{A}n} = (1-\varepsilon)\, V_{\mathrm{A}} \cos(a)/2, \tag{9}$$

$$V'_{\mathrm{A}t} = V_{\mathrm{A}} \sin(a), \tag{10}$$

$$V'_{\mathrm{B}n} = (1+\varepsilon)\, V_{\mathrm{A}} \cos(a)/2. \tag{11}$$

To derive the deflection angle, θ, in Figs. 1 and 2, we use the following equation:

$$\tan(\theta) = V'_{\mathrm{A}n}/\, V'_{\mathrm{A}t} = (1-\varepsilon)\, V_{\mathrm{A}} \cos(a)/2/\, V_{\mathrm{A}} \sin(a) = (1-\varepsilon)/2/\tan(a), \tag{12}$$

$$\theta = \mathrm{atan}[(1-\varepsilon)/2/\tan(a)]. \tag{13}$$

The deflection angle is thus found to be a function of the restitution coefficient and attack angle. Generally, given a constant restitution coefficient, the deflection angle from the normal object ball motion direction is inversely proportional to the magnitude of the attack angle. A smaller attack angle creates a larger deflection, and vice versa. This phenomenon can be evidenced from the simulation results of the proposed reposition algorithms in Figs. 7–9.

We examine deriving an optimal initial speed to drive the cue to an optimal location for the best subsequent shots, given the rebound cue deflection path. To achieve this goal, a minimum initial cue stick speed to drive the cue to hit a selected object ball and sink it into a pocket must first be considered. Given such an initial velocity, the object ball travels just enough to sink into the target pocket, and the cue is deflected and stops at a fixed location. This initial speed depends on the travel distance from the object position to the pocket and from the cue to the object. Given this minimum speed, it is possible to estimate the minimum post-collision cue position. To find the optimal cue stick speed to drive the object ball into a target pocket and send the cue to an optimal position for the next best shot, additional speed (or force) is needed. Our approach of finding this optimal speed is described below. We describe the procedure to find the minimum post-collision position of a cue ball with an initial object ball speed that is enough to roll it towards a proper target pocket and stop at the pocket center with zero speed. The basics of collision physics use a restitution factor and are shown in Fig. 2, where the relevant notations are stated. We use an inverse derivation process to find the minimum speed required by an object ball to travel to the target pocket and stop at the pocket center. Given this minimum object ball speed, the minimum initial cue speed after colliding with the object ball can be derived.

The minimum post-collision travel distance of the cue ball can then be solved, assuming a constant deceleration of the ball from the constant friction coefficient between the ball and table cloth. Equation (14) describes the regular Torricelli's equation, which evaluates the final velocity given a constant deceleration and known initial velocity without a known time interval. If we want the object ball to travel with a minimum initial velocity and stop at the pocket center, its final velocity, ν, must be zero. We thus derive Eq. (15), where $U_{\min}$ is the minimum required initial velocity to drive the object ball for a travel distance of S_1 at a constant deceleration, μ, due to friction. $U_{\min}$ is equivalent to the $V'_{\mathrm{B}n}$ in Fig. 1. Given the calculated initial object ball velocity, the incident velocity of the cue before colliding with the object ball can be calculated using Eq. (15) and the restitution solution of Eq. (11), solving them for the post-collision object ball velocity. The incident velocity of the cue before collision can thus be expressed using the restitution factor, ε, the travel distance, S_1, and the cut angle 'a', as in Eq. (16). We must next decide the minimum post-collision cue travel distance. After deriving the estimated initial cue velocity after collision, the minimum post-collision cue travel distance is expressed as in Eq. (18) by setting the end velocity to zero in Eq. (17).

Because the post-collision cue velocity can also be expressed as the sum of the squares of two vertical components, $V'_{\mathrm{A}n}$, and $V'_{\mathrm{A}t}$, the minimum post-collision cue travel distance, S_2, can be expressed as in Eq. (20), by combining Eqs. (18) and (19). We express $V'_{\mathrm{A}n}$, and $V'_{\mathrm{A}t}$ using the cut angle, 'a', and object sink distance, 'S_1'. Equation (21) is derived by dividing Eq. (9) by (11). By further substituting Eq. (15) into Eq. (21), we can express $V'_{\mathrm{A}n}$ as a function of the restitution factor, ε, and S_1, as in Eq. (22). By substituting the right-hand side of Eq. (16) into the V_{A} in Eq. (23), the $V'_{\mathrm{A}t}$ quantity can be expressed using the restitution factor, ε, and S_1. Equation (24) combines the $V'_{\mathrm{A}n}$ and $V'_{\mathrm{A}t}$ quantities from Eqs. (22) and (23) to form the final S_2 expression. The minimum post-collision travel distance thus appears to vary as the cue driving force increases. After further simplifying Eq. (24), this quantity is a function of the travel distance of an object into a pocket, the restitution factor and the cutting angle, as in Eq. (25). To calculate the minimum speed for the cue to force the collided object to sink into the target pocket, we use the same inverse procedure as in Eqs. (14) to (25). Given the cue travel distance from the starting position to the collision point, S, Eq. (26) gives the relation between the starting and ending velocities. After rearranging terms and substituting CV_{end} into Eq. (16), the required minimum cue speed is expressed as a function of S_2, ε, and S and cut angle 'a', as in Eq. (27).

$$v^2 = U_{\min}^2 - 2 \times \mu \times S_1, \tag{14}$$

$$V'_{\mathrm{B}n} = U_{\min} = \sqrt{2 \times \mu \times S_1} = \frac{(1+\varepsilon)\, V_{\mathrm{A}} \cos(a)}{2}, \tag{15}$$

$$CV_{\mathrm{end}} = V_{\mathrm{A}} = \frac{2\sqrt{2\mu S_1}}{(1+\varepsilon)\cos(a)}, \tag{16}$$

$$0 = V_{\min}^2 - 2 \times \mu \times S_2, \tag{17}$$

$$S_2 = \frac{V_{\min}^2}{2 \times \mu}, \tag{18}$$

$$V_{\min}^2 = V_{\mathrm{A}n}'^2 + V_{\mathrm{A}t}'^2, \tag{19}$$

$$S_2 = \frac{V_{\mathrm{A}n}'^2 + V_{\mathrm{A}t}'^2}{2 \times \mu}, \tag{20}$$

$$V'_{\mathrm{A}n} = \frac{(1-\varepsilon)}{(1+\varepsilon)}\, V'_{\mathrm{B}n}, \tag{21}$$

$$V'_{\mathrm{A}n} = \frac{(1-\varepsilon)}{(1+\varepsilon)}\sqrt{2\mu S_1}, \tag{22}$$

$$V'_{\mathrm{A}t} = V_{\mathrm{A}} \sin(a) = CV_{\mathrm{end}} \sin(a) = \frac{2\sqrt{2\mu S_1}}{(1+\varepsilon)\cos(a)} \sin(a), \tag{23}$$

$$S_2 = \frac{V_{\mathrm{A}n}'^2 + V_{\mathrm{A}t}'^2}{2 \times \mu} = \frac{1}{2 \times \mu}\left[\left(\frac{(1-\varepsilon)}{(1+\varepsilon)}\sqrt{2\mu S_1}\right)^2 + \left(\frac{2\sqrt{2\mu S_1}}{(1+\varepsilon)}\tan(a)\right)^2\right], \tag{24}$$

$$S_2 = S_1\left[\left(\frac{(1-\varepsilon)}{(1+\varepsilon)}\right)^2 + \left(\frac{2}{(1+\varepsilon)}\tan(a)\right)^2\right], \tag{25}$$

$$CV_{\min}^2 = CV_{\mathrm{end}}^2 + 2 \times \mu \times S, \tag{26}$$

$$CV_{\min}^2 = \left(\frac{2\sqrt{2\mu S_1}}{(1+\varepsilon)\cos(a)}\right)^2 + 2 \times \mu \times S. \tag{27}$$

Given this selected object and pocket combination, the motion direction and minimum post-collision cue travel distance are determined using the analysis results in Eq. (27). We then search along the path from the minimum stop points of post-collision motion for an optimal position, using different search algorithms introduced in Sec. 4. After finding the optimal position, S_{opt} is determined. The relationship between pre-collision, CV_{end}, and post-collision, V_{opt}, cue velocity can be expressed as in Eq. (28) and illustrated in Fig. 3. The rebound deflection angle, θ, is decided using Eq. (13) and used to calculate the necessary initial velocity to drive the cue and stop at an optimal position. By combining Eqs. (28) and (29), the ending cue velocity, CV_{end}, can be expressed in Eq. (30) using S_{opt}, θ and 'a'. Equation (31)

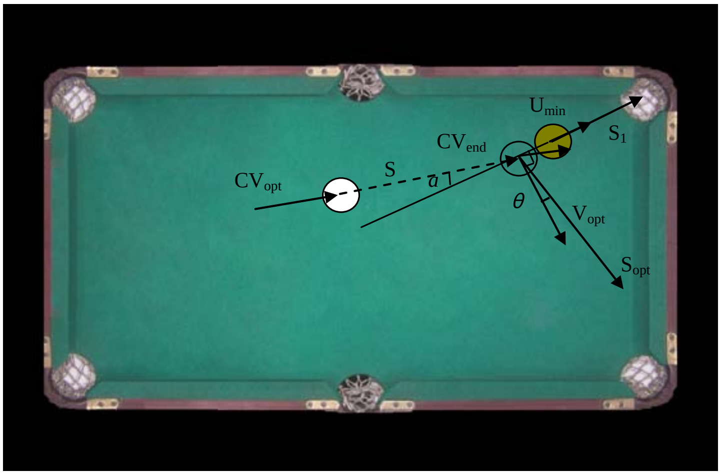


Fig. 3. Post collision physics schematic and notations showing optimal driving speed calculation notations.

describes the relationship between the initial driving and ending speeds for the pre-collision cue motion. After rearranging terms in Eq. (31) and substituting CV_{end} with the right-hand side terms of Eq. (30), Eq. (32) gives the expression for the optimal initial cue stick speed, CV_{opt}, using S_{opt}, S, cut angle 'a', and rebound deflection angle, 'θ'. This quantity is used below to mark the estimated cue stick speed for both the algorithm simulations results and the GUI front end of the guidance experiments setups.

$$V_{\text{opt}} = CV_{\text{end}} \frac{\sin(a)}{\cos(\theta)}, \tag{28}$$

$$V_{\text{opt}} = \sqrt{2\mu S_{\text{opt}}}, \tag{29}$$

$$CV_{\text{end}} = \sqrt{2\mu S_{\text{opt}}} \frac{\cos(\theta)}{\sin(a)}, \tag{30}$$

$$CV_{\text{end}}^2 = CV_{\text{opt}}^2 - 2 \times \mu \times S, \tag{31}$$

$$CV_{\text{opt}}^2 = \frac{2 \times \mu \times S_{\text{opt}}}{(\sin(a))^2} \cos(\theta)^2 + 2 \times \mu \times S. \tag{32}$$

This is the estimated velocity that the GUI draws for users to follow, placing the cue at this position after colliding with the selected object ball.

4. Shots Planning Algorithms

Game playing strategies decide the success of many games. Chess relies heavily on planning strategies several steps ahead of each move to win a game. Billiard games emphasize both physical skills and strategy. The strategic part involves both deciding the object balls and pocket combination to target and the position control of the cue after colliding with selected object balls. The cue ball placement has decisive effects on the follow-up shots. One extreme example is placing the cue close to object balls that are near the jaws of nearby pockets, making it easy to sink the object ball. The other extreme case occurs when the cue is far from an object ball or the object balls are far from the pockets. We propose three search algorithms to serve as game-playing strategies to locate the optimal cue position for the best follow-up shots. Following the analysis results from Sec. 3, the cue picks the best object ball and pocket combination for its first shot based on a maximum error tolerance criterion with different strategies. The first strategy is based on a nearest pocket selection criterion. Figure 4 details the optimal nearest pocket search algorithm. The search has two rounds: pre- and post-collision for a given cue position. An *a priori* search on all object balls is exercised to find the best candidate with a maximum tolerance angle to sink into the nearest pocket. A post-collision search is then conducted to find the optimal position for the cue to sink the next object ball into the nearest pocket with a maximum tolerance angle. Both searches consider collision-free cue paths towards and away from the object ball for optimization process candidates. We deliberately set a threshold value after evaluating the maximum tolerance angle in the sequential search process of the algorithm. If the maximum tolerance angle calculated during the sequential shot evaluation process falls below a certain threshold value, the search stops. As players may exhibit different proficiency levels, the threshold value can be set at different values to emulate the sequential shot results. A lower threshold value indicates that sequential shots can easily accrue without breaking the search process for optimal next shots, while a higher threshold value means that sequential shots may suffer an early search process termination. Generally, a lower threshold value incurs more sequential shots and represents a highly proficient player, and vice versa. The nearest pocket algorithm is analyzed to have $O(MN)$ efficiency, where M is the number of object balls and N is the number of post-collision search points. The computation complexity order is in the second-degree polynomial order, caused by the double-nested loop of a post-collision search for the optimal position for follow-up shots.

To test the algorithm, we select two object ball distribution patterns. The first ball configuration used to test the algorithms is a sparse distribution with every ball around the jaws of each pocket, while the second configuration is a more clustered distribution with each ball farther from each pocket. We define the ball distribution difficulty as the sum of the distance of the object balls to their nearest pockets. Figure 5 shows one sample search based on the algorithm in Fig. 4, under a sparse pattern with a low threshold value. Figure 6 shows sample search results of the algorithm in Fig. 4, with a cluster pattern under a low threshold value. The cue starts at a selected position, which is the same for Figs. 5 and 6. The continuous sequence of successful shots is illustrated for the same initial cue position. As marked in Fig. 5, the first shot picked by the nearest pocket algorithm is the number 3 object ball, which is closest to the cue and at the jaw of a nearby pocket, the number 2 pocket.

```
Optimal nearest pocket shots planning algorithm
Input:
$O=\{O_1, O_2,\ldots, O_n\}$;//initial table state of 'n' number of object balls' positions
$C_s$ = start cue position;
Th= the threshold value of tolerance angle to regulate if the attack shall continue or not.
Output:
$S=\{C_1, C_2,\ldots, C_m\}$;//optimal stop positions of cue after the 'j'th strike, $1<j<m$, $m<=n$.
SI={$SI_1$, $SI_2$, …, $SI_m$};//index of sinking object ball ID. after the 'j'th strike, $m<=n$. $1<SI_j<n$.
    BID(1~n): the object ball ID. Number with the maximum tolerance angle among all object balls.
    $ANG_{max}(0\sim90°)$: temporary storage for the maximum tolerance angle calculation.
    $REP_{max}(0\sim90°)$: temporary storage for the tolerance angle calculation in the outer while loop traversing each accessible object ball.
Algorithm:
1.  Set $i$=1, $ANG_{max}$ =0; // initialize temporary storage for the maximum tolerance angle calculation
2.  for each nonblocked objects, $O_i \in O$ (meaning no other obstacles on the linear path to cue and to the nearest pocket and on the post collision path of cue)
  2.0 find nearest pocket, $PK_c$, for $O_i$
  2.1 given $C_s$, $O_i$ and $PK_c$ calculate 'ang' (tolerance angle) using Eq. (3).
  2.2 if ($ANG_{max}$ < 'ang') then
        $ANG_{max}$ ='ang';
        BID = $i$;
      end if
      $i=i+1$;
        end for in step 2.
3.  set $REP_{max} = ANG_{max}$,// initialize temporary storage for the tolerance angle calculation in the outer while loop traversing each accessible object ball.
4.  $j$=1;//index to store the 'j'th optimal cue position.
5.  $SI_1$= BID;//store the ball ID of the target object ball for the first strike
6.  while there still are accessible object balls on the table and $REP_{max}$ > Th
7.  given the target ball ID, $SI_j$, nearest pocket $PK_c$ for $O_{SIj}$ and $C_s$ (current cue position) find a linear post collision path using Eq. (13) of Sec. 3.
8.  collect all legal stop positions, PS={$PS_1$, $PS_2$, …, $PS_k$} along the path from step 7, after the minimum post collision position of cue as calculated by Eq. (25) of Sec. 3.
9.  for each stop point $PS_i$ in PS of cue positions collected from step 8, set $C_s = PS_i$, repeat step 1 and 2, extract $ANG_{max}$ and BID.
10. if ($REP_{max} < ANG_{max}$)
        $REP_{max} = ANG_{max}$
        $SI_{j+1}$= BID
        store the current cue position, $PS_i$, as $C_j$
         endif
11. update the cue position from $PS_i$ to $PS_{i+1}$
12. end for in step 8,
13. mark the $O_{SIj+1}$ as next attack target
14. inversely calculate the driving initial speed for cue, given $C_j(S\text{opt})$ and $C_{j-1}(S)$ using Eq. (32), this is the estimated initial speed to be displayed
15. user drives the cue using the estimated speed and sinks a previous best candidate object ball
16. $O=O-O_{BID}$
17. place the cue at $C_s= C_j$, $j=j+1$, repeat step 6
```

Fig. 4. Optimal shots planning algorithm aiming for the nearest pocket.

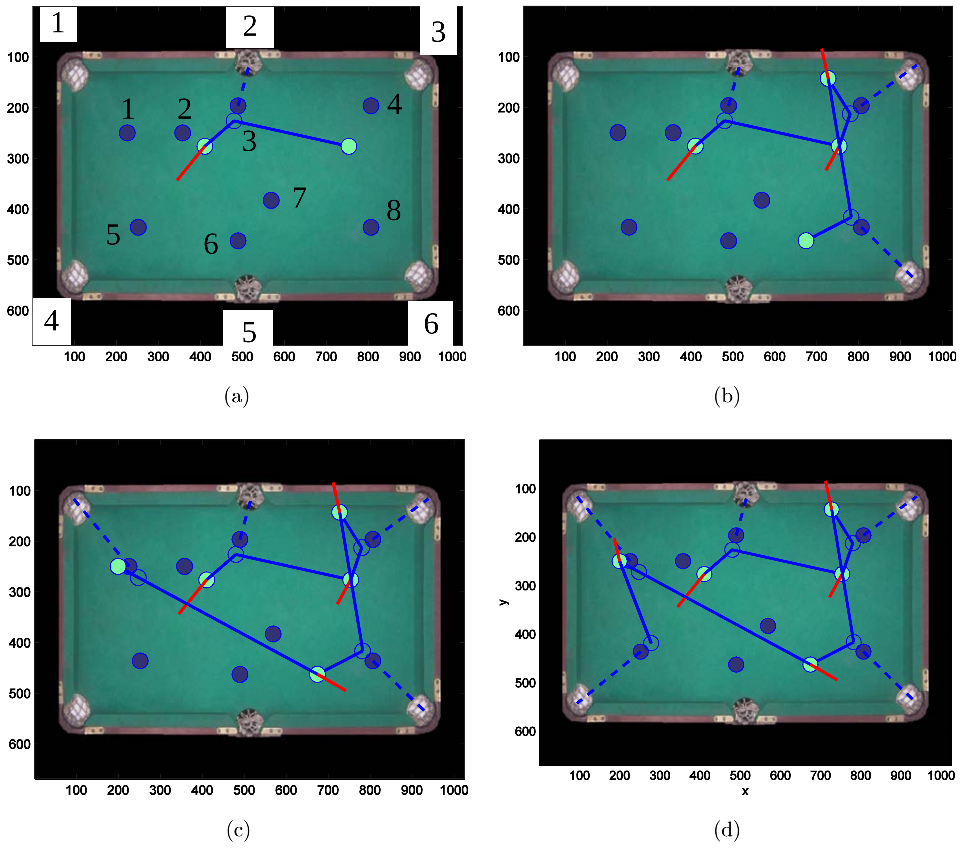


Fig. 5. Simulated sequential shots from low threshold values, using the nearest pocket algorithm under a sparse configuration.

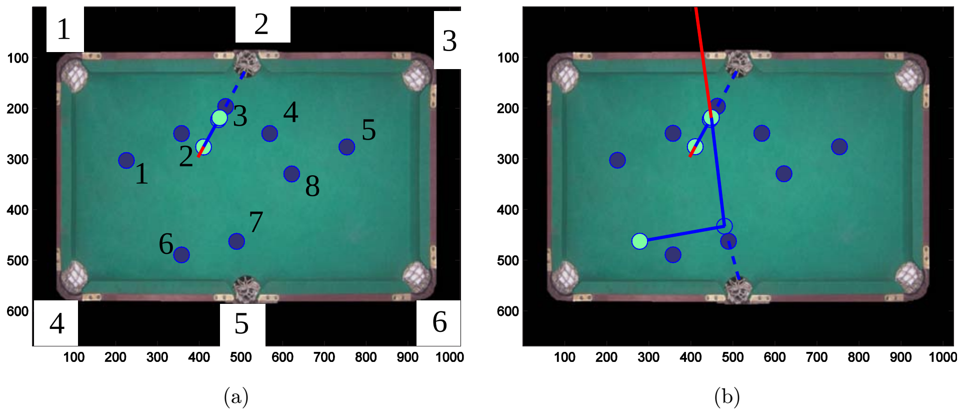
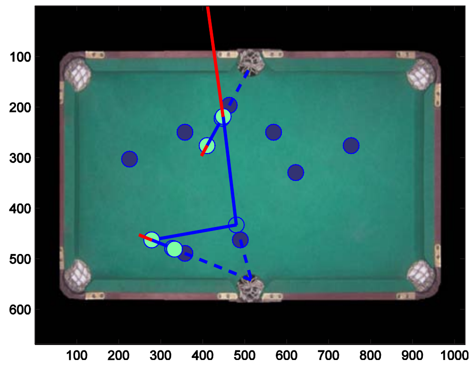
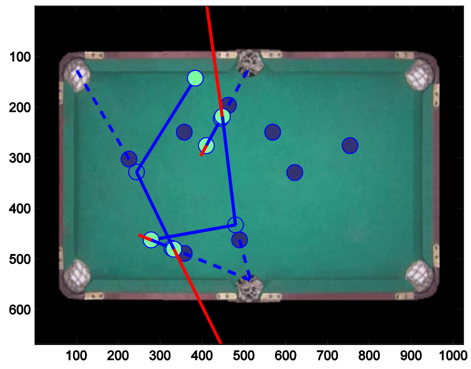


Fig. 6. Hit sequence simulation results from low threshold values, using the near pocket algorithm with a clustered configuration.



(c)



(d)

Fig. 6. (*Continued*)

Not only does this give an intuitive proof of the correctness of our algorithm in identifying the best first shot with the maximum error tolerance criterion, but the post-collision cue stop position also provides a clear shot for the number 4 object ball to sink into pocket number 3. The sequential shot order to clear the table is aiming for the number 3, 4, 8, 1, and 5 object balls; the corresponding pocketing sequence is 2, 3, 6, 1, and 4. The cue rebound position after sinking each object ball is also illustrated in each sub-figure of Fig. 5. For the fourth optimal shot, the program picked ball number 1 instead of balls number 5, 6, and 7. This also illustrates that the algorithm can filter balls with inaccessible angles, bounded by their nearest pockets. Given a fixed start cue position, the maximum number of sinkable object balls is limited by the threshold value in the simulation situation and the aiming accuracy that real-world players can render using our guidance system. More simulation results on this configuration with different threshold values are shown below. A more difficult ball configuration is arranged to test this algorithm in selecting the nearest pockets, as in Fig. 6. There are only four sequential shots observed, given the same start cue positions and threshold values as in Fig. 5.

The sink order in Fig. 6 is number 3, 7, 6, and 1, respectively. After colliding with ball number 1, the cue stops close to the rail near pocket 2, and it becomes difficult to sink all other object balls on the table, as these balls are fixed to sink into the nearest pockets, making the tolerance angles close to zero for remaining object balls, firing from the final cue stop position. This phenomenon is interesting and drives the need to explore another algorithm, as in Fig. 7, to consider all pockets during the search process.

Figure 7 details the optimal all-pocket search algorithm. The search again has two rounds: pre- and post-collision for a given cue position. An *a priori* search on all object balls towards all accessible pockets is exercised to find the best candidate with a maximum tolerance angle. A post-collision search is then conducted to find the optimal position for the cue to sink the next object ball into a suitable pocket with

```
Optimal all pockets shots planning algorithm
Input:
$O=\{O_1, O_2, \ldots, O_n\}$;//initial table state of ‘$n$’ number of object balls’ positions
$C_s$ = start cue position;
Th= the threshold value of tolerance angle to regulate if the attack shall continue or not.
Output:
$S=\{C_1, C_2, \ldots, C_m\}$;//optimal stop positions of cue after the ‘$j$’th strike, $1<j<m$, $m<= n$.
PK={$PK_1$, $PK_2$, …, $PK_m$};//target pocket number of sinking object balls after the ‘$j$’th strike, $m<= n$, $1<=PK_i<=6$.
GT={$GT_1$, $GT_2$, …, $GT_m$};//ghost ball positions of cue after the ‘$j$’th strike, $1<j<m$, $m<= n$.
SI={$SI_1$, $SI_2$, …, $SI_m$};//index of sinking object ball ID. after the ‘$j$’th strike, $m<= n$. $1<SI_j<n$.
    BID($1\sim n$): the object ball ID. Number with the maximum tolerance angle among all object balls.
    $ANG_{max}$($0\sim 90°$): temporary storage for the maximum tolerance angle calculation
    curAkID($1\sim n$): current attack object ball number.
Algorithm:
1. Given a start cue position, $C_s$, find a best first shot on ‘BID’ object ball with max. tolerance angle around all accessible pockets.
2. $j=1$, $O=O-O_{BID}$
3. while there still are accessible object balls on the table and $ANG_{max}$ >Th
4. set curAkID = BID
5. Set $ANG_{max}$ =0; // initialize temporary storage for the maximum tolerance angle calculation
6. for each accessible pocket, $PT_k$, $k=1\sim 6$(clear path from object, $O_{curAkID}$, to pocket and clear from post collision of cue with other object balls)
        6.1 find a post collision deflection path of cue starting from $C_s$ to drive curAkID object toward the target pocket, $PT_k$, and rebound at $GT_j$ per rule of Eq. (13) of Sec. 3.
        6.2 collect all legal stop positions, PS={$PS_1$, $PS_2$, …, $PS_w$} along the path from step 6.2, after the minimum post collision position of cue as calculated by Eq. (25) of Sec. 3.
        6.3 for each stop point $PS_{jp}$, $jp=1\sim w$, in PS of cue positions collected,
                6.3.1 for each nonblocked objects, $O_i \in O$ (meaning no other obstacles on the linear path to cue and on the post collision path of cue) , $i=1\sim n$.
                    6.3.1.1 for each accessible pocket, $PT_{pk}$, $pk=1\sim 6$ (clear path from object, $O_i$, to pocket and clear from post collision of cue with other object balls)
                    6.3.1.2 given a cue position, $PS_{jp}$, for each accessible object ball, $O_i$, and pocket, $PT_{pk}$, combination, calculate tolerance angle =ang using Eq. (3)
                    6.3.1.3 if ($ANG_{max}$ < ang) then
                                $ANG_{max}$ =ang;
                                BID = $i$;
                                $SI_{j+1} = i$;
                                $PK_{j+1}=pk$;
                                $C_{j+1}= PS_{jp}$;
                            endif
                    end for(of $PT_{pk}$ traversal of step 6.3.1.1)
                end for(of $O_i$ traversal of step 6.3.1)
        6.4. update cue position for a fixed amount of increment in the direction of deflection, until new cue position reach table edge
            end for (of $PS_j$ traversal of step 6.3)
end for (of $PT_k$ traversal of step 6)
```

Fig. 7. Optimal shots planning algorithm considering all pockets.

```
7. Given start cue position, $C_s$, $C_{j+1}$ (optimal post collision stop position), $\mathrm{SI}_{j+1}$ (optimal
target object ball for the following shot) and $\mathrm{PK}_{j+1}$, inversely calculate the driving initial
speed given $S$opt and $S$ using Eq. (32), this is the estimated initial speed to be displayed
8. user drives the cue using the estimated speed and sinks a previous best candidate object
ball
9. $O = O - O_{\mathrm{BID}}$
10. $C_s = C_{j+1}$, repeat steps 2~10
```

Fig. 7. (*Continued*)

the maximum tolerance angle. The search along the deflected path, calculated using Eq. (13), aims to find the optimal combination of object and pockets with a maximum tolerance angle for each cue sample point on the deflection path. We deliberately set a threshold value after evaluating the maximum tolerance angle in the sequential algorithm search process. If the maximum tolerance angle calculated during the sequential shots evaluations process falls below a certain threshold value, the search stops; otherwise the *a priori* selected optimal object ball is marked as invisible. The post-collision selected optimal candidate object and pocket combination are marked as new *a priori* selected optimal objects, and the whole search process repeats. The all-pocket algorithm is analyzed to have $O(MNK)$ efficiency, where M is the number of object balls, N is the number of post-collision search points, and K is the number of pockets. The computation complexity order is in the third-degree polynomial order, because the triple search operation of the nested loop on the post-collision path seeks the optimal position for follow-up shots.

Figure 8 shows the simulated sequential shot results of the algorithm in Fig. 7, given the same initial cue position as in Figs. 5 and 6 under a sparse pattern with a low threshold value. The sequential shot order to clear the table aims for the number 3, 4, 8, 1, 6, 7, 2, 5 object balls, as in Fig. 6. However, the corresponding pocketing sequence is 2, 3, 6, 1, 5, 6, 1, and 5; this differs from Fig. 6, which presents 2, 3, 6, 1, 5, 5, 2, and 4. This indicates that the algorithm can correctly find the optimal object and pocket combinations along the rebound path of *a priori* object ball choice. As marked in Fig. 8, the sixth shot picked by the all-pocket algorithm is the number 7 object ball sinking into pocket 6. Pocket 6 is certainly not the closest pocket to the number 7 ball. The following shots continue to clean the table by sinking the number 2 ball into pocket 1 and the number 5 ball into pocket 5, which are not nearby pockets. Not only does this give an intuitive proof of the correctness of our algorithm in identifying the best object ball and pocket combinations in the post-collision search process, but the cue stop positions also provide a comparable selection of sequential shots as the algorithm in Fig. 4. The same ball configurations shown in Fig. 6 are arranged in Fig. 9 to test the algorithm in Fig. 7 in searching all pockets. There are only eight successful sequential shots, as observed given the same start cue positions and threshold values as in Fig. 6. The sink order of object balls is 3, 7, 6, 1, 8, 4, 2, and 5; the corresponding order of sinking pockets is 2, 5, 4, 1, 6, 5, 5,

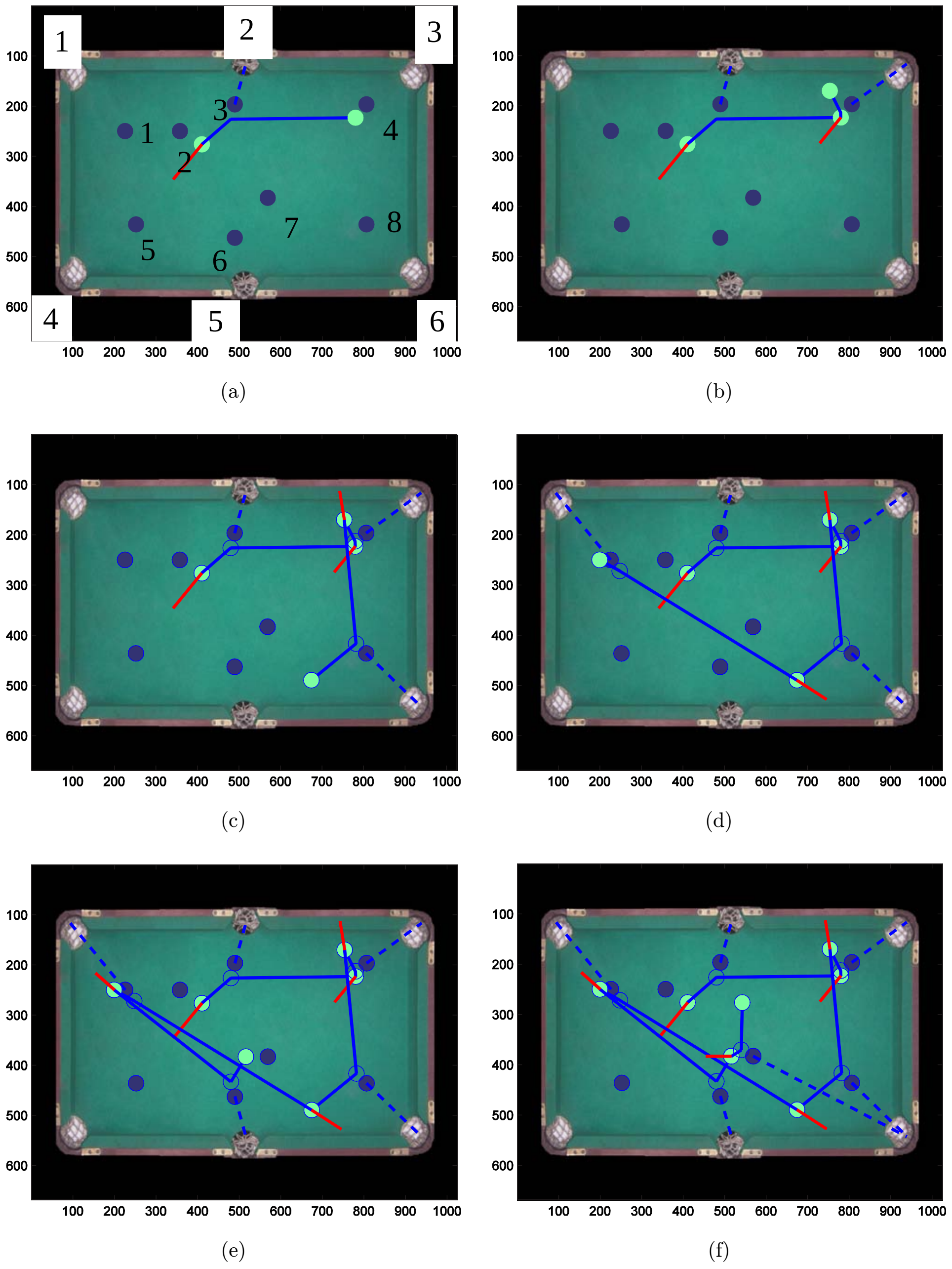
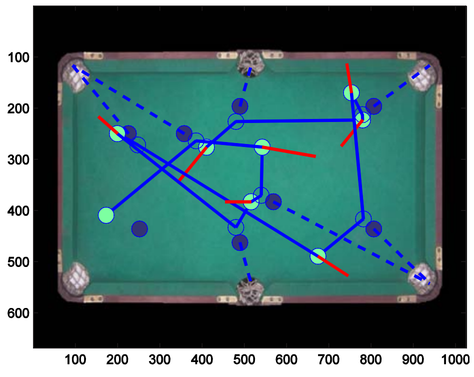


Fig. 8. Sequence shot sparse distribution using the all-pocket algorithm with low threshold values.


(g)

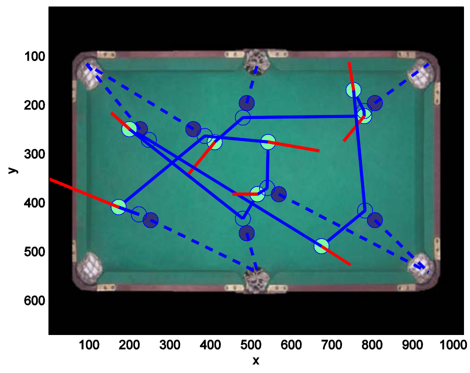
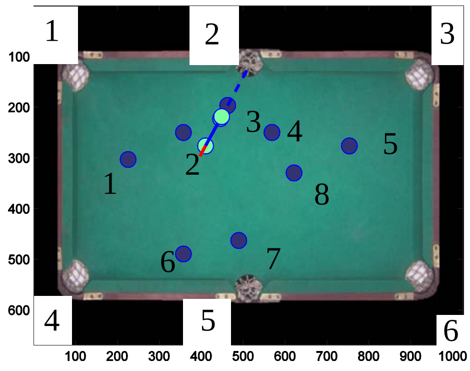
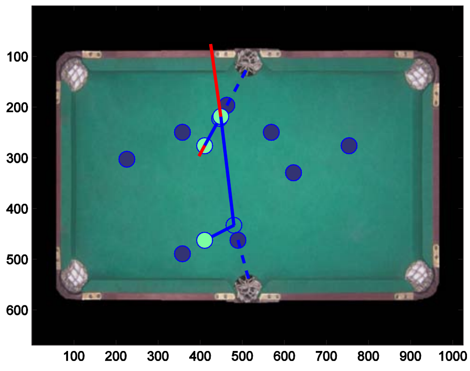
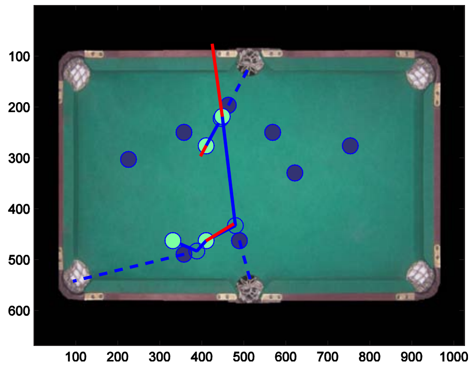

(h)

Fig. 8. (*Continued*)



(a)


(b)


(c)

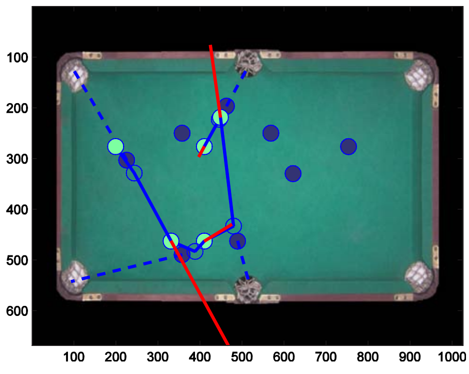

(d)

Fig. 9. Hit sequence of the all-pocket algorithm on a cluster configuration with low threshold values.

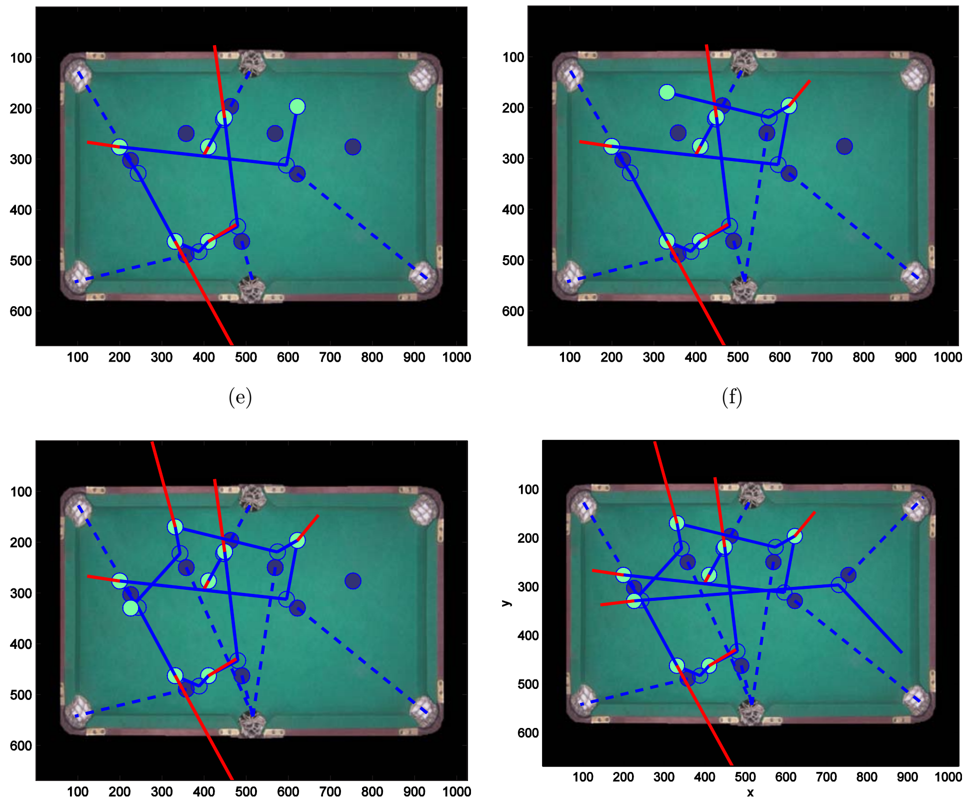


Fig. 9. (*Continued*)

and 3. Comparing the results of Figs. 6 and 9, the effectiveness of the all-pocket search algorithm over nearest pockets algorithm in Fig. 4 is apparent with the same threshold value. The shorter sequence of successful shots on object balls in Fig. 6 is the same as the first four shots in Fig. 9. Their order is 3, 7, 6, 1, while the sinking pocket order is 2, 5, 5, and 1 for Fig. 6. By selecting different pockets to sink object ball number 6, the subsequent cue placement patterns are different to allow Fig. 9 to extend the shots and be able to clear the table.

To explore the effect of different optimization criteria on the outcome of the continuous shots, a multi-objective criterion is used in the search for the optimal post-collision cue positions in Fig. 10. This algorithm uses the same all-pocket search as in Fig. 7. The multi-objective criterion combines the tolerance angle and the inverse of the attack angle as the optimization goal. The varying parameters are the uniformly distributed sample points along the post-collision path. For each candidate post-collision cue position, the tolerance angle plus the inverse of the attack angle are calculated in step 7.3.1.2. We aim both to optimize the tolerance angle and minimize

```
Multi-objective optimal all pockets shots planning algorithm
Input:
$O=\{O_1, O_2, \ldots, O_n\}$;//initial table state of '$n$' number of object balls' positions
$C_s$ = start cue position;
Th= the threshold value of tolerance angle to regulate if the attack shall continue or not.
Output:
$S=\{C_1, C_2, \ldots, C_m\}$;//optimal stop positions of cue after the '$j$'th strike, $1<j<m$, $m<=n$.
PK=$\{PK_1, PK_2, \ldots, PK_m\}$;//target pocket number of sunk object balls after the '$j$'th strike, $m<=n$, $1<=PK_i<=6$.
GT=$\{GT_1, GT_2, \ldots, GT_m\}$;//ghost ball positions of cue after the '$j$'th strike, $1<j<m, m<=n$.
SI=$\{SI_1, SI_2, \ldots, SI_m\}$;//index of sunk object ball ID. after the '$j$'th strike, $m<=n$. $1<SI_j<n$.
    BID($1\sim n$): the object ball ID. Number with the multi-objective optimization index among all object balls.
    $ANG_{max}$($0\sim 90°$): temporary storage for the maximum tolerance angle calculation
    maxTol($0\sim 90°$): temporary storage for the maximum multi-objective optimization index calculation
    curAkID($1\sim n$): current attack object ball number.
Algorithm:
1. maxTol=0
2. for each nonblocked objects, $O_i \in O$ (meaning no other obstacles on the linear path to cue and on the post collision path of cue) , $i=1\sim n$.
    2.1. for each accessible pocket, $PT_{pk}$, $pk=1\sim 6$, (clear path from object, $O_i$, to pocket and clear from post collision of cue with other object balls)
        2.2. given a start cue position, $C_s$, for each accessible object ball, $O_i$, and pocket, $PT_{pk}$, combination, calculate tolerance angle =ang using Eq. (3) and the '$a$' angle (attack angle) in Fig. 1
        2.3. Calculate the combined optimization index tmpIndex = ang + 1/'$a$' angle
      2.4. if (tmpIndex > maxTol) then
                    maxTol = tmpIndex;
                    $ANG_{max}$ = ang;
                        BID =$i$;
                        $SI_1=i$;
                        $PK_1=pk$;
        endif
      end for (of PT traversal of step 2.1)
end for (of $O_i$ traversal of step 2)
3.  $j=1$, $O=O-O_{BID}$
4.  while there still are accessible object balls on the table and $ANG_{max}$ >Th
5.  set curAkID = BID
6.  Set maxTol=0; // initialize temporary storage for the calculation of multi-objective optimization index
7. for each accessible pocket, $PT_k, k=1\sim 6$ (clear path from object, $O_{curAkID}$, to pocket and clear from post collision of cue with other object balls)
        7.1 find a post collision deflection path of cue starting from $C_s$ to drive curAkID object toward the target pocket, $PT_k$, and rebound at $GT_j$ per rule of Eq. (13) of Sec. 3.
        7.2 collect all legal stop positions, PS=$\{PS_1, PS_2, \ldots, PS_w\}$ along the path from step 6.2, after the minimum post collision position of cue as calculated by Eq. (25) of Sec. 3.
        7.3 for each stop point $PS_{jp}, jp=1\sim w$, in PS of cue positions collected,
            7.3.1 for each nonblocked objects, $O_i \in O$ (meaning no other obstacles on the linear path to cue and on the post collision path of cue), $i=1\sim n$.
```

Fig. 10. Multi-objective optimal shots planning algorithm considering all pockets.

```
                    7.3.1.1 for each accessible pocket, $\mathrm{PT}_{pk}$, $pk$=1~6 (clear
                            path from object, $O_i$, to pocket and clear from
                            post collision of cue with other object balls)
                    7.3.1.2 given a cue position, $\mathrm{PS}_{jp}$, for each accessible
                            object ball, $O_i$, and pocket, $\mathrm{PT}_{pk}$, combination,
                            calculate tolerance angle =ang using Eq. (3) and
                            the '$a$' angle (attack angle) in Fig. 1
                    7.3.1.3 calculate the combined optimization index
                            'tmpIndex' = ang + 1/'$a$' angle
                    7.3.1.4 if (maxTol < tmpIndex) then
                        maxTol = tmpIndex;
                            $\mathrm{ANG}_{\max}$ = ang;
                            BID = $i$;
                            $\mathrm{SI}_{j+1} = i$;
                            $\mathrm{PK}_{j+1}=pk$;
                        $\mathrm{C}_{j+1}= \mathrm{PS}_{jp}$;
                         endif
                  end for(of $\mathrm{PT}_{pk}$ traversal of step 7.3.1.1)
              end for(of $O_i$ traversal of step 7.3.1)
      7.4. update cue position for a fixed amount of increment in the direction of
      deflection, until new cue position reach table edge
          end for (of $\mathrm{PS}_j$ traversal of step 7.3)
end for (of $\mathrm{PT}_k$ traversal of step 7)
8. Given start cue position, $C_s$, $C_{j+1}$ (optimal post collision stop position), $\mathrm{SI}_{j+1}$ (optimal
target object ball for the following shot) and $\mathrm{PK}_{j+1}$, inversely calculate the driving initial
speed given $S$opt and $S$ using Eq. (32), this is the estimated initial speed to be displayed
9. user drives the cue using the estimated speed and sinks a previous best candidate object
ball
10. $O= O-O_{\mathrm{BID}}$
11. $C_s = C_{j+1}$, repeat step 4
```

Fig. 10. (*Continued*)

the attack angle; the attack angle is thus inversely calculated and summed with the tolerance angle. Physically, this means users get to play with the maximum tolerance angle and minimum attack angle. Because the extremes of both quantities cannot co-exist, the optimization algorithm in Fig. 10 generates the optimal cue positions with trade-offs for both quantities. Although the optimization process uses the multi-objective criterion to seek the ideal post-collision cue position, the stop criterion for the sequential shots sought in step 4 of Fig. 10 is the tolerance angle, compared to a threshold value for performance evaluation. This is reasonable, as the tolerance angle for each sequential shot decides the physical difficulty of that specific shot. A human player tends to have a slight degree of deviation from an ideal strike line due to grab jitter on the cue stick in the actual strike motion. The extent of deviation depends on the proficiency level. Filtering a threshold value on the tolerance angles during the sequential shots can reasonably symbolize an actual game scenario for different proficiency levels. A small threshold value represents a proficient player.

The multi-objective algorithm is analyzed to have $O(MNK)$ efficiency, where M is the number of object balls, N is the number of post-collision search points, and K is the number of pockets. The computation complexity order is in the third-degree polynomial order, because the triple search operation of the nested loop on the post-collision path seeks an optimal position for follow-up shots. The computation complexity of this algorithm is the same as that of the all-pocket algorithm.

Figure 11 shows the simulation results of this algorithm on the same cluster balls configuration as in Fig. 9, with the same low threshold value. The sink order of object balls is 2, 3, 7, 6, 8, 5, 4, and 1 as observed in sub-figures (a)–(h). The corresponding order of sinking pockets number 1, 2, 5, 6, 5, 3, 4, and 4. The sink order of object balls is 3, 7, 6, 1, 8, 4, 2, and 5 for Fig. 9. The corresponding order of sinking pockets is 2, 5, 4, 1, 6, 5, 5, and 3. Comparing the results of Figs. 9 and 11, the effectiveness of the multi-objective search algorithm in aligning the angle between cue, object and pocket is clear. Most continuous shot attack angles are small, with attackable tolerance angles with small threshold values. This allows the search for optimal shots to continue until the table is cleared.

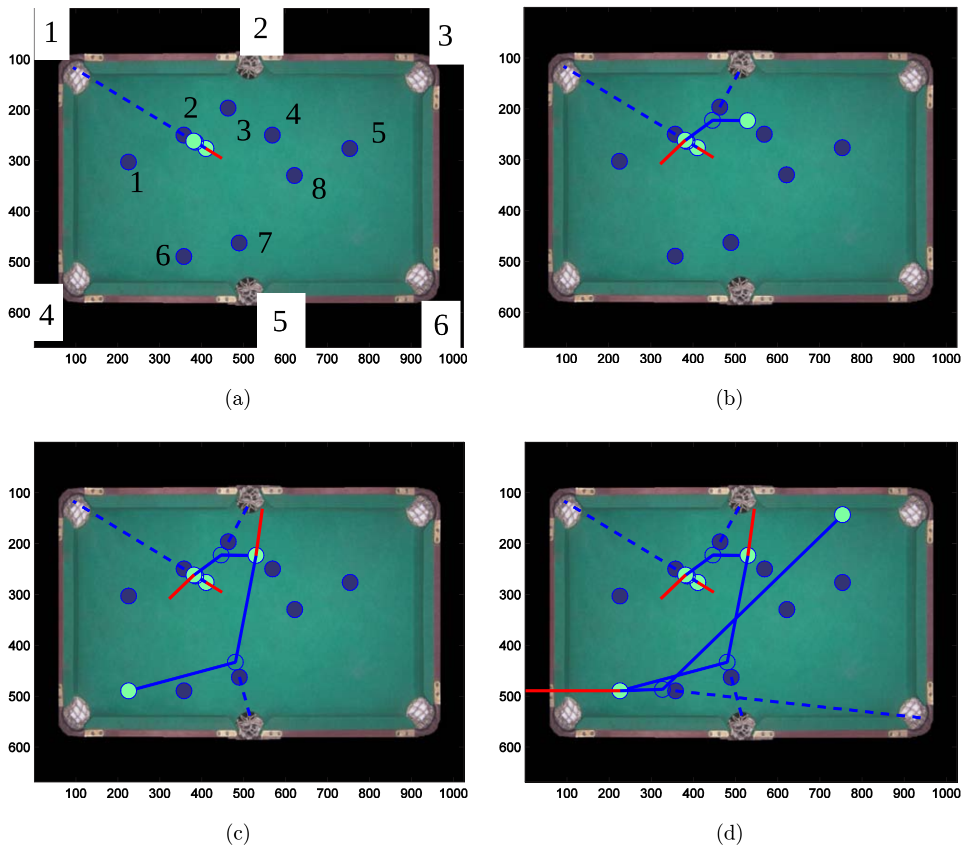
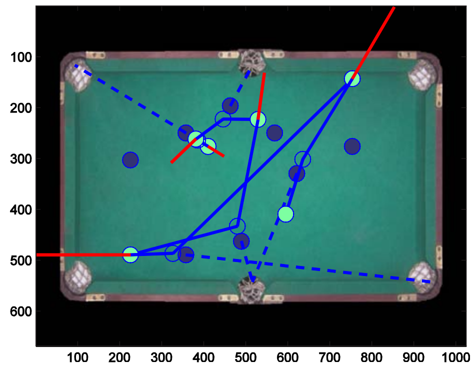
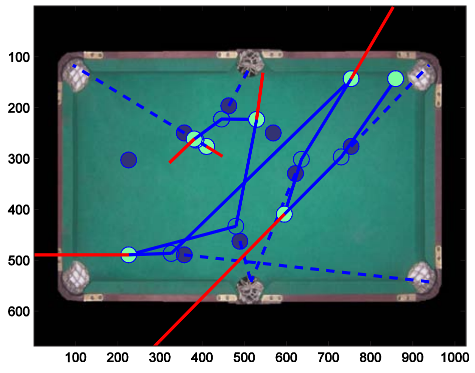


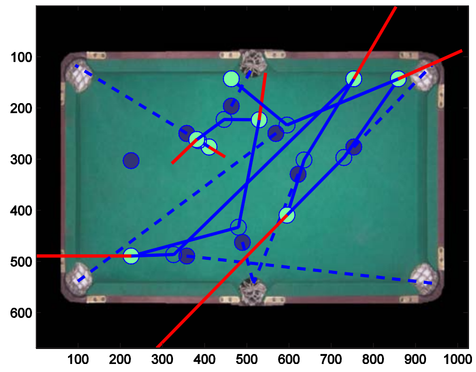
Fig. 11. Simulated sequential shots of the multi-objective optimization algorithm on a cluster configuration.



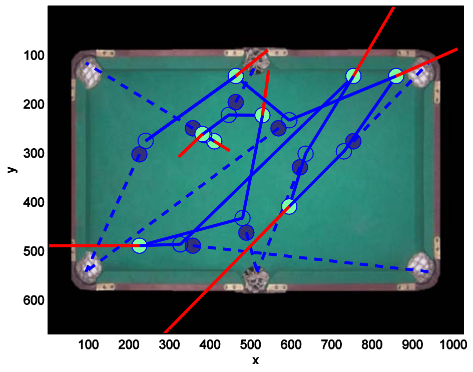
(e)



(f)



(g)



(h)

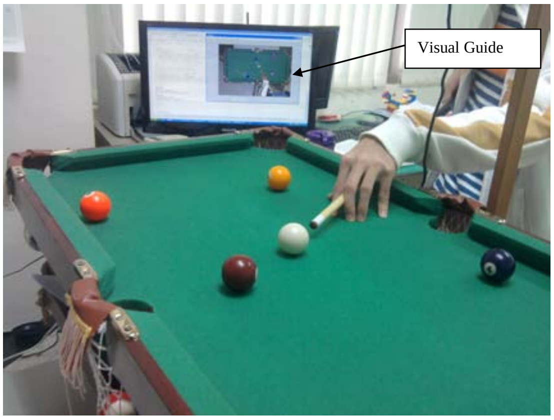
Fig. 11. (*Continued*)

5. Experimental Results

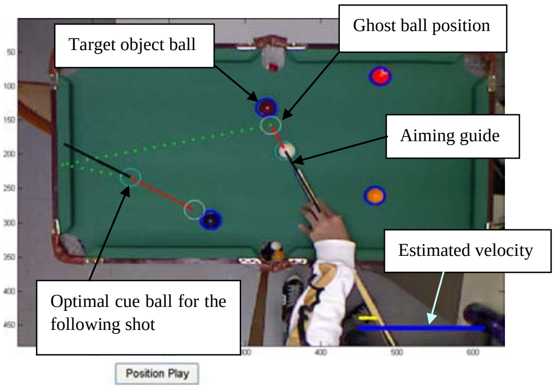
5.1. *Guidance system description and data flow integration*

The guidance system is modified from a previous work[1] with an addition of a visual display of the captured real-time pool table image, aiming direction, optimal post collision path, cue position for next shot, and cue stick velocity estimation, as in Fig. 12(a). The requirements for the system are the same as in previous studies,[1] in that the CCD camera field of view must cover the whole billiard board table with a minimal amount of surrounding environment pixel information enclosed. However, the cue stick does not need to be tagged as such in the previous design.[1] The graphical interface displays both the static pool table images and the user's dynamic aiming action captured in real time using a four-core AMD PC.

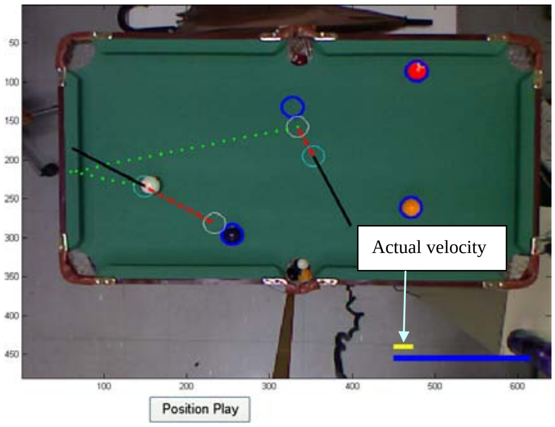
A back-end program processes both the visual display of the instructions for users to place the cue stick on the pool table and the execution of the optimal game



(a)



(b)



(c)

Fig. 12. Augmented reality-based shots reposition guidance system.

strategies, given ball coordinate information grabbed by the CCD camera. Given the analysis results from back-end simulations about the optimal aiming direction, aiming ghost ball position, optimal cue position for the following shot, post-collision path, and required velocity to drive the cue to its next best position shown in Fig. 12(b), the user then aligns the cue stick with the guidance line, aims for the ghost ball and exercises the shots for proper cue stick acceleration. The user then strikes with proper force to sink the target object balls into the target pocket selected by the optimal strategies and places the cue near the desired best position for the next shots, as in Fig. 12(c). The post-collision cue position may be inconsistent with the estimation shown on the visual display, depending on the user's skills. However, the analyzed attack path display does enhance the usability in their strokes. Users can compare the stroke results with the theory estimation and gain intuition for their control. This is another new feature in this work.

Figure 13 describes the whole data flow and process steps for system integration. We first grab an image of the pool table with users playing or exercising. The steps exercised for the integration process include the following: (1) ball and cue stick tip tracking. (2) Transforming the image coordinates of all balls and cue stick tips to the world coordinates on table. (3) Optimizing the shot planning algorithm, which generates an aiming guide and estimated velocity guide for optimal sequential shots. (4) Inverse transforming world coordinates of aiming and velocity guide vectors into image coordinates. (5) Superimposing the scaled aiming aid and post-collision strategy aid on the pool table image grabbed in step 1; repeat steps 1 to 5 to update the guidance information as users keep sinking object balls according to the optimal shot control algorithm. Details of cue ball tracking and pool table calibration can be observed in a previous paper.[14] The standard deviation error between real-world and transformed coordinates for each grid point is on the order of 0.1 cm, as reported in a previous study.[14] The major difference of the current system from the previous design is that the visual guide uses real-time captured pool table images for the presentation background, rather than using a fixed pool table image and adding the cue stick line in the background. The guidance system in Fig. 12 works without graphical cue stick presentation, because the cue stick image is grabbed and displayed in the visual system in real time. Only guidance lines, ghost ball, next optimal cue position and cue stick velocity controls for cue position placements are added.

5.2. *Cue friction calibration process*

All balls on the table tend to slow to a stop position due to friction with the table cloth. To combine the optimal shot planning algorithms in Sec. 4 for a precise estimation of the force exerted on the cue ball, the balls' decelerations must be fully investigated. A previous work[1] developed a deceleration calibration procedure, described in Sec. 6.2. We use the same procedure to collect data from about 20 dry-run shots by users with different skill levels. Figure 14 records and shows two

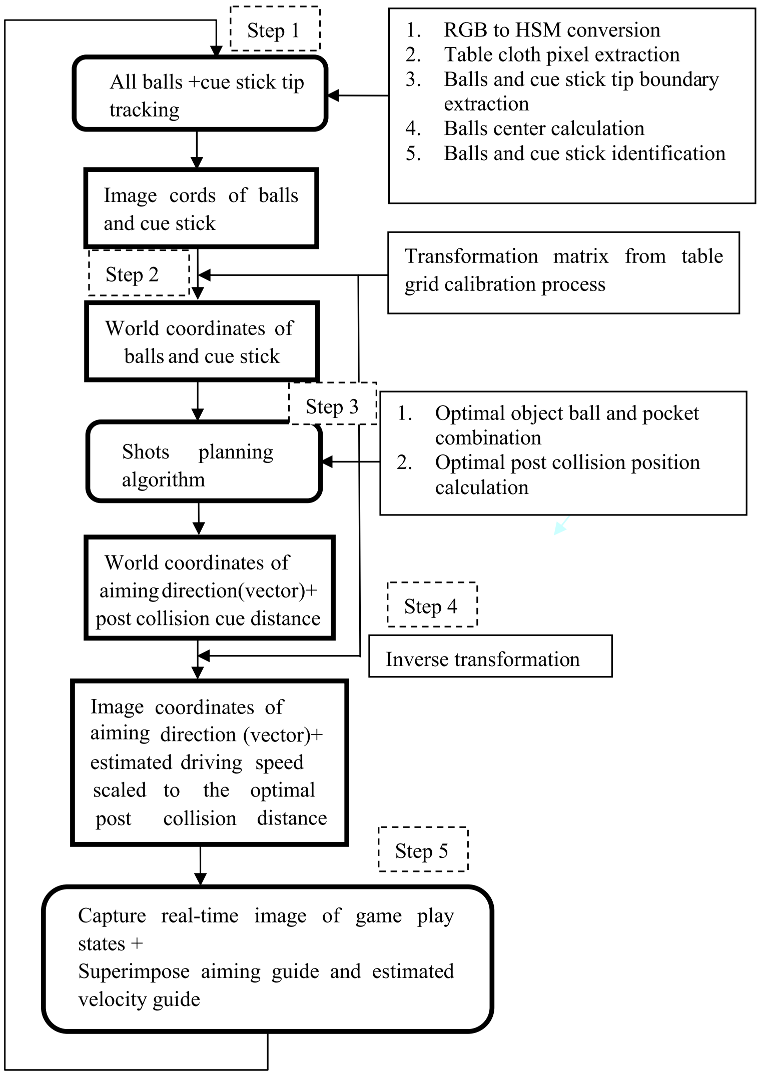
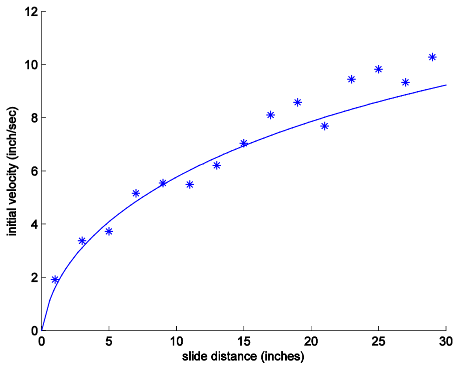
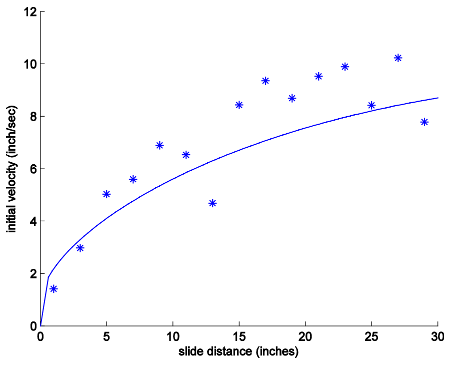

Step 1
All balls +cue stick tip tracking
1. RGB to HSM conversion
2. Table cloth pixel extraction
3. Balls and cue stick tip boundary extraction
4. Balls center calculation
5. Balls and cue stick identification
Image cords of balls and cue stick
Step 2
Transformation matrix from table grid calibration process
World coordinates of balls and cue stick
Step 3
1. Optimal object ball and pocket combination
2. Optimal post collision position calculation
Shots planning algorithm
World coordinates of aiming direction(vector)+ post collision cue distance
Step 4
Inverse transformation
Image coordinates of aiming direction (vector)+ estimated driving speed scaled to the optimal post collision distance
Step 5
Capture real-time image of game play states +
Superimpose aiming guide and estimated velocity guide

Fig. 13. System integration flow diagram of reposition guidance system.

corresponding characteristic data sets of the velocity to the travel distance. We record the tracked velocity and corresponding travel distance of the hit ball for each shot. Figure 14(a) displays the experiment data for users with high proficiency, while Fig. 14(b) shows that for low-proficiency users. As the relationship of hand motions exercising the cue stick is complicated, we use the cubical polynomial calibration equation to generate the correlation of distance to velocity instead of a parabolic equation.[1] This guarantees a more precise mapping from travel distance to an initial speed. For each pair of corresponding velocity and sliding distance, V_i and S_i, their interrelation is given as follows.



(a)



(b)

Fig. 14. Velocity measurements and calibration results for players with different proficiency.

$$\begin{bmatrix} S_1^3 & S_1^2 & S_1 & 1 \\ S_2^3 & S_2^2 & S_2 & 1 \\ & \bullet & & \\ & \bullet & & \\ S_n^3 & S_n^2 & S_n & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} V_1 \\ V_2 \\ \bullet \\ \bullet \\ V_n \end{bmatrix}. \tag{33}$$

Given an initial velocity, V_i, the travel distance, S_i, is recorded when the ball stops.

Equation (33) is further denoted as $B * T = A$, where B stands for

$$\begin{bmatrix} S_1^3 & S_1^2 & S_1 & 1 \\ S_2^3 & S_2^2 & S_2 & 1 \\ & \bullet & & \\ & \bullet & & \\ S_n^3 & S_n^2 & S_n & 1 \end{bmatrix},$$

T stands for $\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}$ and A stands for $\begin{bmatrix} V_1 \\ V_2 \\ \bullet \\ \bullet \\ V_n \end{bmatrix}$. T, a correlation matrix, is then calculated using a least-squares error transformation equation.[1] Using this transformation matrix, any value of distance (S) is transformed to the velocity value (V) using $[S^3 \quad S^2 \quad S \quad 1] * T = V$. This value is substituted into Eq. (32) to estimate the start speed and is scaled on the visual display of Fig. 13. The curves in Fig. 14 are the results of plotting the transformed velocity against the increasing sliding distance. These curves represent the motion characteristics of the players hitting the balls. Players with different skill levels have different characters that must be calibrated before each one plays on different pool tables.

5.3. *Simulation results of shots planning algorithms*

Given the variety of winning requirements for all real-world games, i.e., 8-ball or 9-ball, the games share one common feature: a player who can execute more sequential shots is more likely to win the game. We use this index for performance comparison between the simulation results of various cue placement algorithms. The shot planning algorithm designs in Sec. 4 allow users with different proficiency levels to simulate real game playing situations using different threshold values. The condition to continue playing the shots is regulated by a comparison between the tolerance angles of the optimal shots selected by different algorithms with a threshold value. Only shots with tolerance angle values over a certain threshold value can continue searching for the next round. This comparison relation emulates the capability of a player with certain proficiency level to render a successful shot within a certain tolerance angle. A low threshold value imitates a highly proficient player who can render low-tolerance angle shots (over the threshold values). A high threshold value imitates a low-skill player who easily misses low-tolerance angle shots. We then adjust the threshold values for each algorithm under different ball configurations. There are three algorithms and two ball configurations: the (1) nearest pocket algorithm, (2) all-pocket algorithm and (3) multi-objective algorithm; the two ball configurations are sparse and cluster distributions, as in Sec. 4. Figures 15 and 16 are the shot sequences of the nearest pocket algorithm with a sparse distribution with low (0.00001) and high thresholds (1), respectively. Figure 15 is drawn from Fig. 5(f). These figures illustrate the effect that the threshold value has on the shot sequences for the same initial start cue position. The number of successful sequential shots is drastically reduced from eight to two in Fig. 16, due to the lower tolerance angle and the high threshold value for the third shot aiming at

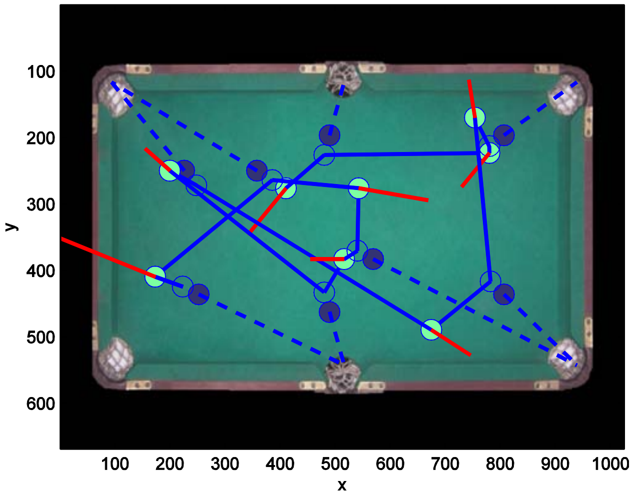


Fig. 15. All shots of the near pocket algorithm with a sparse distribution with a low threshold (0.00001).

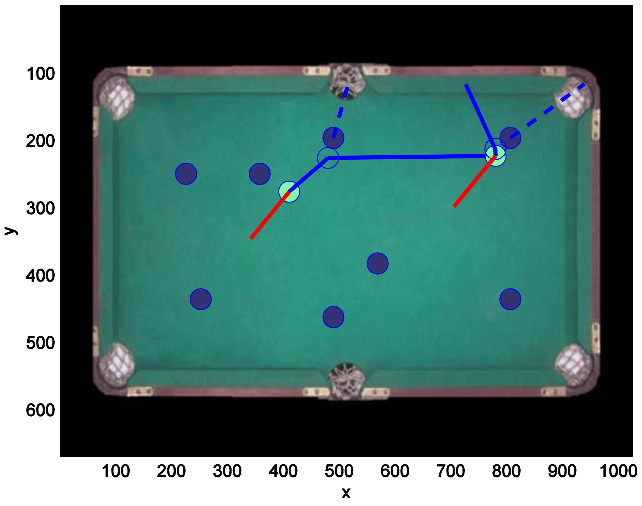


Fig. 16. All shots of the nearest pocket algorithm with a sparse distribution with a high threshold (1).

ball number 8. The algorithm stops the further search process, as it figures that the user cannot successfully sink ball number 8, as the user's skill can render shots higher than the threshold value with ease (evaluated with tolerance angle). Any lower tolerance angle causes users to miss shots. We made this worst-case assumption to compare the performance of all relevant algorithms. Figures 17 and 18 are the shot sequences of the all-pocket algorithm with a sparse distribution with low (0.00001) and high thresholds (1), respectively. The same sequential shot reduction phenomenon is observed in Fig. 18, as the threshold value increases. The maximum number of successful sequence shots that our simulation program can enable given a fixed

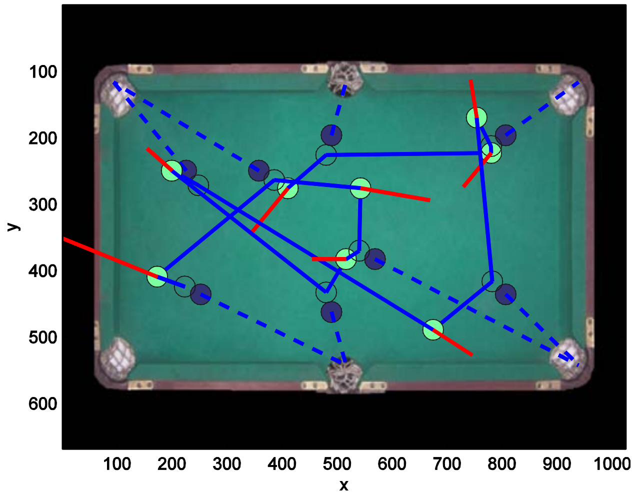


Fig. 17. All shots of the all-pocket algorithm with a sparse distribution with a low threshold (0.0001).

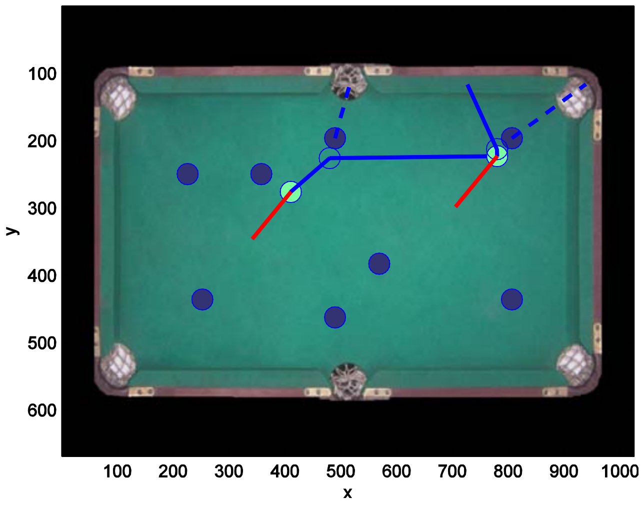


Fig. 18. All shots of the all-pocket algorithm with a sparse distribution with a high threshold (1).

start cue position varies and depends on the threshold values. Furthermore, different start cue positions have different numbers of maximum successful sequence shots for the same threshold value. A distribution profile of the maximum number of successful sequence shots can be evaluated for various cue positions on the pool table for a fixed threshold value. Three optimal shot planning algorithms are exercised for every uniformly distributed start cue position. Figure 19 displays the calculated profiles for the sequential sink rate of the near pocket algorithm with low (a) and high (b) thresholds in a sparse configuration. Figure 20 shows the calculated profiles for the sequential sink rate of the all-pocket algorithm with low (a) and high (b) thresholds in a sparse configuration. Figure 21 displays the calculated profiles for the sequential sink rate of the multi-objective algorithm with low (a) and high (b) thresholds in a sparse configuration. Table 2 summarizes sequential shot statistics as

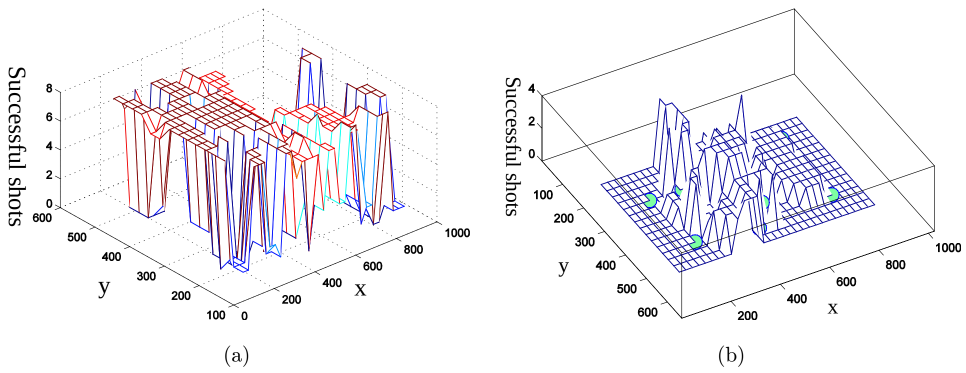


Fig. 19. Sequential sink rate distribution profile of the near pocket algorithm with low (a) and high (b) thresholds in a sparse configuration.

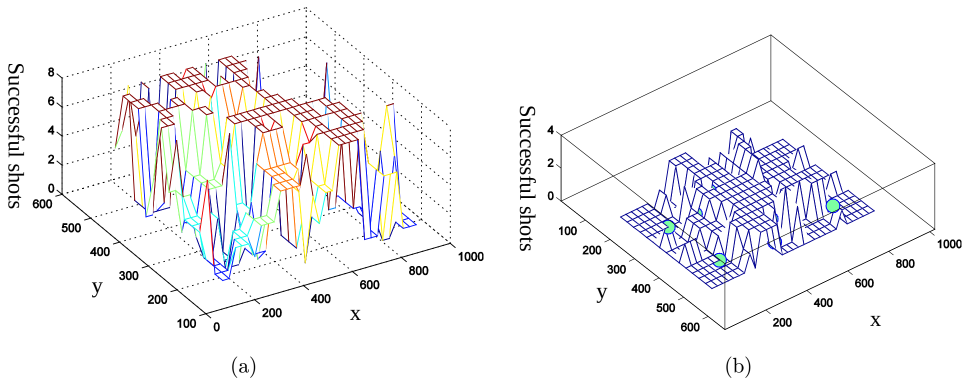


Fig. 20. Sequential sink rate distribution profile of the all-pocket algorithm with low (a) and high (b) thresholds in a sparse configuration.

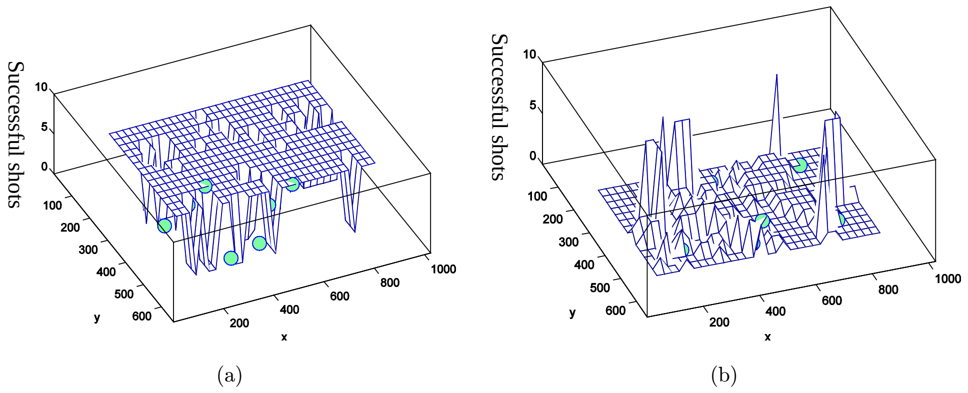


Fig. 21. Sequential sink rate distribution profile of the multi-objective all-pocket algorithm with low (a) and high (b) thresholds in a sparse configuration.

functions of different threshold values of the near pocket algorithm with a sparse distribution. Table 3 summarizes the sequential shot statistics as functions of different threshold values of the all-pocket algorithm with a sparse distribution. Table 4 summarizes the sequential shot statistics as functions of different threshold

Table 2. Sequential shot statistics as functions of different threshold values of the near pocket algorithm with a sparse distribution.

Threshold	Number of minimum equential shots count	Number of maximum sequential shots count	Average sequential shots count	High sequential shots count
0.0001	0	8	3.304	72
0.5	0	6	1.15	0
1	0	4	0.61	0

Table 3. Sequential shot statistics as functions of different threshold values of the all-pocket algorithm with a sparse distribution.

Threshold	Number of minimum sequential shots count	Number of maximum sequential shots count	Average sequential shots count	High sequential shots count
0.0001	0	8	4.04	168
0.5	0	6	1.80	0
1	0	3	0.65	0

Table 4. Sequential shot statistics as functions of different threshold values of the near pocket algorithm with a cluster distribution.

Threshold	Number of minimum sequential shots count	Number of maximum sequential shots count	Average sequential shots count	High sequential shots count
0.0001	0	7	2.63	12
0.5	0	7	1.94	2
1	0	6	0.75	0

values of the near-pocket algorithm with a cluster distribution. Table 5 provides the sequential shot statistics as functions of different threshold values of the all-pocket algorithm with a cluster distribution. Table 6 shows the sequential shot statistics as functions of different threshold values of the multi-objective algorithm with a cluster distribution. Table 7 summarizes the sequential shot statistics as functions of different threshold values of the multi-objective algorithm with a sparse distribution.

Table 5. Sequential shot statistics as functions of different threshold values of the all-pocket algorithm with a cluster distribution.

Threshold	Number of minimum sequential shots count	Number of maximum sequential shots count	Average sequential shots count	High sequential shots count
0.0001	0	8	5.8	297
0.5	0	8	3.4	112
1	0	5	0.9	0

Table 6. Sequential shot statistics as functions of different threshold values of the multi-objective algorithm with a cluster distribution.

Threshold	Number of minimum sequential shots count	Number of maximum sequential shots count	Average sequential shots count	High sequential shots count
0.0001	0	8	6.79	435
0.5	0	8	1.746	26
1	0	8	0.421	1

Table 7. Sequential shot statistics as functions of different threshold values of the multi-objective algorithm with a sparse distribution.

Threshold	Number of minimum sequential shots count	Number of maximum sequential shots count	Average sequential shots count	High sequential shots count
0.0001	0	8	6.72	429
0.5	0	8	1.94	60
1	0	8	0.65	15

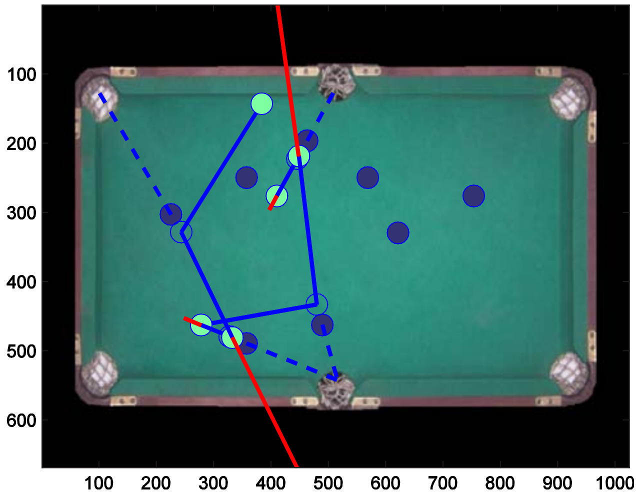


Fig. 22. All shots of the near pocket algorithm with a cluster distribution with a low threshold (0.0001).

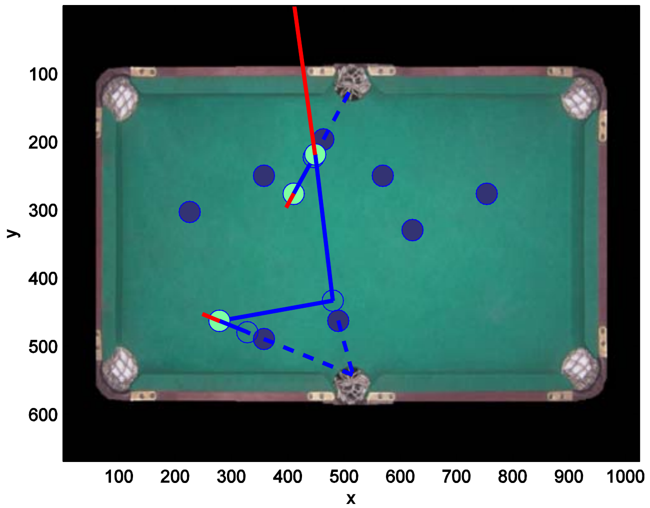


Fig. 23. All shots of the near pocket algorithm with a cluster distribution with a high threshold (1).

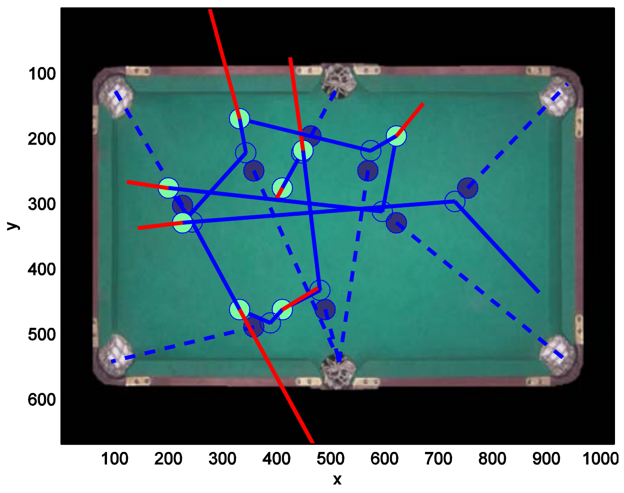


Fig. 24. All shots of the all-pocket algorithm with a cluster distribution with a low threshold (0.0001).

The statistics compared for all tables include the minimum, maximum, average sink rates and a high sequential shot count. We specifically define a new comparison index, the high sequential shots count. This index counts successful consecutive sink shots greater than seven. This is used to compare the efficiency of different algorithms in rendering high numbers of consecutive shots.

Tables 2 and 3 use different algorithms but keep the same object ball distribution, sparse distribution, and threshold values, from 0.0001 to 1. The statistics indicate that the nearest pocket algorithm perform moderately inferior to the all-pocket algorithm in all threshold value ranges in both the average and maximum sink rates.

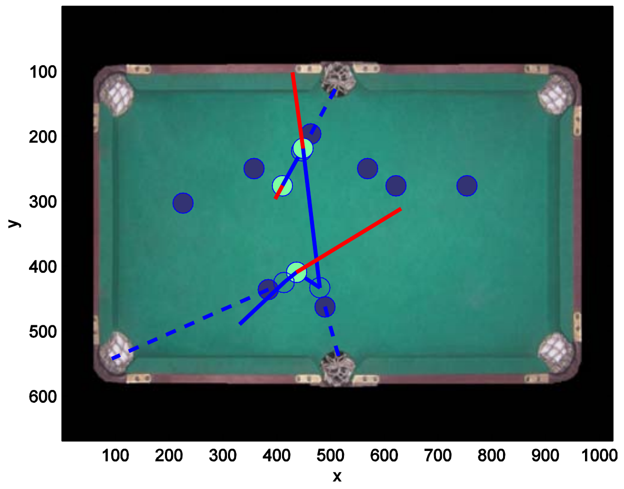


Fig. 25. All shots of the all-pocket algorithm with a cluster distribution with a high threshold (1).

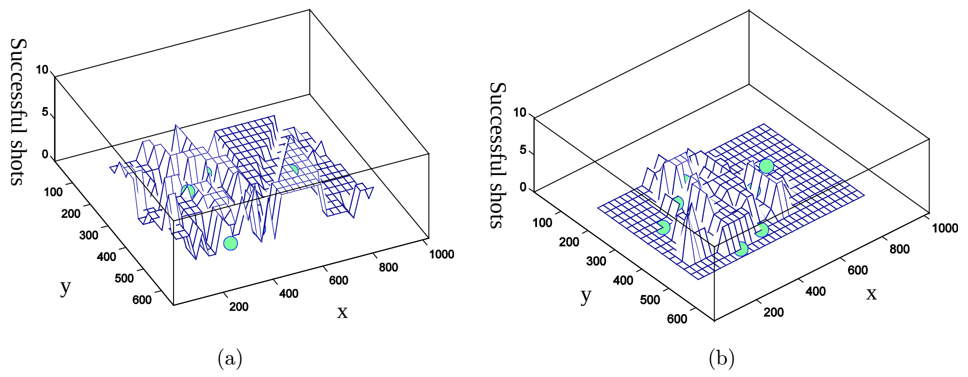


Fig. 26. Sequential sink rate distribution profile of the all-pocket algorithm with low (a) and high (b) thresholds in a sparse configuration.

Moreover, the high sequential shot count is much higher with the all-pocket algorithm than the nearest algorithm.

Tables 4 and 5 use different algorithms but keep the same object ball distribution, cluster distribution, and threshold values, from 0.0001 to 1. The efficiency of the all-pocket algorithm over that of nearest algorithm can be observed immediately. Under a harder ball configuration with greater distances from the pockets, the performance of the nearest pocket algorithm falls far lower than that of the all-pocket algorithm. The average sink rate falls to under half of that of the all-pocket algorithm in Table 5, and the high sequential sink count is reduced to 12, compared to 297 for the all-pocket algorithm. The clustered ball configuration can be genuinely hard to sink consecutively, even with very low threshold values. The all-pocket algorithm overcomes this difficulty by searching through all post-collision paths, targeting all accessible pockets for optimal positions for the next consecutive shot. The post-collision search space also covers all possible combinations of accessible pockets and

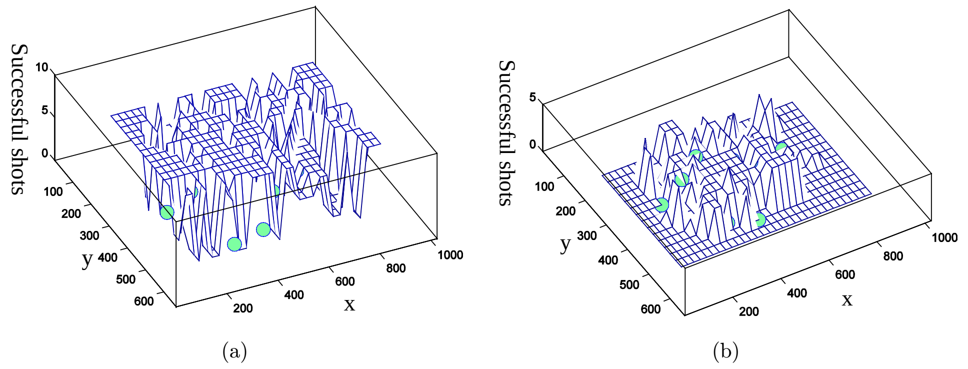
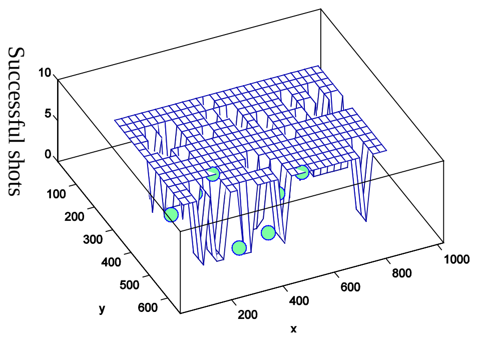


Fig. 27. Sequential sink rate distribution profile of the all-pocket algorithm with low (a) and high (b) thresholds in a cluster configuration.

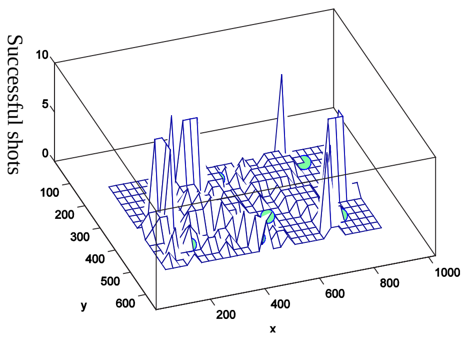
object balls for maximum tolerance values. This feature automatically enhances the capability of the all-pocket algorithm in tackling both easy and hard ball configurations. The dominance of the all-pocket algorithm over the nearest pocket algorithm continues even for the medium threshold value of 0.5, as in Tables 4 and 5. This phenomenon is even more interesting, as the 0.5 threshold value is a normal value range regularly observed by high- to medium-efficiency players. The all-pocket algorithm value can be enhanced even more when combined with the visual guidance system in this paper. Using our visual aid system can enhance both the aiming and reposition accuracy. This is equivalent to a simulation with lower threshold values.

The poor performance of both algorithms at the high threshold value of 1 can be attributed to early sequential shot termination due to high shot perturbation by low-skill players who aim for the optimally selected shots. The algorithms cannot control this, so it is beyond the scope of this paper. The all-pocket algorithm has proven itself suitable for guiding users in both *a priori* shots and optimal post-collision shot placement. Our visual guide system thus uses this algorithm to plan shots and guide repositioning.

Tables 6 and 7 present the simulation results of the multi-objective algorithm. The same threshold values used to test the other algorithms are adapted to evaluate the performances on the cluster and sparse distributions. Table 6 shows that the multi-objective algorithm has a higher sink rate than the all-pocket algorithm, with a small threshold value of 0.0001 under the cluster distribution in Table 5, which is an interesting phenomenon. The high sink rate for both the cluster and sparse configurations in Tables 6 and 7 can be related to the higher flat zone areas with eight consecutive shots on the left of Fig. 21. The high frequency of the high sink counts of the multi-objective algorithm can be attributed to the higher number of co-linear alignments of the cue, object and pockets. This is a direct result of the multi-objective optimization process. To achieve this arrangement, the algorithm somehow sacrifices the tolerance angle optimal values. As the algorithm uses the tolerance



(a)



(b)

Fig. 28. Sequential sink rate distribution profile of the multi-objective algorithm with low (a) and high (b) thresholds in a cluster configuration.

angle as a stop condition, only low threshold values can allow small tolerance angle shots to continue shooting. When the threshold value increases, many shots with smaller tolerance angles cannot surpass the thresholds and must terminate early. The middle threshold value (0.5–1) thus has a small average sink rate and high sequential sink counts in Tables 6 and 7, compared to that in Tables 3 and 5. Because the threshold is set with a unit of degree, a small threshold value is apparently not practical. This phenomenon makes the multi-objective algorithm less practical than the all-pocket algorithm in the middle threshold range, which normally occurs in real-world gaming scenarios with or without our guidance system. We thus choose to use the all-pocket algorithm to test our guidance system during shot planning and cue placement control.

5.4. *Comparing real game play statistics with/without using guidance system*

From the above simulation results obtained by testing various algorithms under different threshold values and ball configuration, the all-pocket algorithm has been deemed suitable for integration into our novel reposition guidance system for real-world game play training. We integrate this algorithm within the shot planning stage of the guidance system design. Given the calibrated ball deceleration rate, we can estimate the initial speeds to drive a cue to obtain the optimal positions for subsequent shots, as calculated in Sec. 3. Section 3 only generates the required optimal cue positions to sink the maximum number of object balls in subsequent shots. Equation (32) is used to estimate these optimal speeds, given the calibrated decelerations from Sec. 5.2, and the optimal distance of the follow-up shots, taken from the optimal shot planning algorithm in Sec. 3.

Given different driving speeds (forces), players with different skill levels are arranged to test our guidance system. The start cue position is deliberately picked to be uniformly distributed on the table. The maximum number of successful sequence shots a user can render given a fixed start cue position varies and depends on user skill levels. Different start cue positions have different numbers of maximum successful sequence shots for the same users. A distribution profile of the maximum number of successful sequence shots can be evaluated for various cue positions on the pool table. This is done for users with different skill levels to determine their performance. The optimal shot planning algorithm is exercised for every uniformly distributed start cue position. Given the pre-calculated best position of sequential shots, the estimated speed counting in the calibrated deceleration distance is displayed on the graphical interface to help users start playing the game. The number of continuous successful shots is recorded at these uniform grid locations for users with different skill levels. Tables 8–11 further summarize these numbers as performance comparison indices.

Tables 8 and 9 show the sequential shot statistics as functions of different threshold values of the all-pocket algorithm with sparse and cluster distributions with guidance,

Table 8. Sequential shots statistics as function of different threshold values of all pocket algorithm with sparse distribution with guidance.

Proficiency level	Number of minimum sequential shots count	Number of maximum sequential shots count	Average sequential shots count	High sequential shots count
High	0	8	3.81	145
Medium	0	7	1.50	0
Low	0	3	0.58	0

Table 9. Sequential shots statistics as function of different threshold values of all pocket algorithm with cluster distribution under guidance.

Proficiency level	Number of minimum sequential shots count	Number of maximum sequential shots count	Average sequential shots count	High sequential shots count
High	0	8	4.4	221
Medium	0	8	2.7	101
Low	0	5	0.86	0

Table 10. Sequential shots statistics of players at different proficiency levels of sparse object balls distribution without guidance.

Proficiency level	Number of minimum sequential shots count	Number of maximum sequential shots count	Average sequential shots count	High sequential shots count
High	0	8	2.16	50
Medium	0	7	0.81	1
Low	0	2	0.261	0

Table 11. Sequential shots statistics of players at different proficiency levels of cluster object balls distribution without guidance.

Proficiency level	Number of minimum sequential shots count	Number of maximum sequential shots count	Average sequential shots count	High sequential shots count
High	0	7	1.58	1
Medium	0	4	0.742	0
Low	0	3	0.363	0

respectively. Tables 10 and 11 are the sequential shot statistics of players at different proficiency levels with sparse and cluster object ball distributions without guidance, respectively. Table 12 is the performance enhancement percentage between Tables 8 and 10 with/without the author's visual guiding system in a sparse configuration, while Table 13 is the performance enhancement percentage between Tables 9 and 11 without/with author's visual shot planning guiding system in a cluster configuration.

Table 12. Performance enhancement percentage between Tables 8 and 10 with/without author's visual guiding system in sparse configuration.

Proficiency level	Maximum # of sequence shots (%)	Average # of sequence shots	High sequential shots count
High	0	43.3	65.51
Medium	0	46	NA
Low	50	55.17	0

Table 13. Performance enhancement percentage between Tables 9 and 11 with/without author's visual guiding system in cluster configuration.

Proficiency level	Maximum # of sequence shots (%)	Average # of sequence shots	High sequential shots count
High	12.5	64	99
Medium	50	72	100
Low	40	57.79	0

In Table 8, the maximum successive sink rate ranges from 3 to 8 for low- and high-proficiency players, and the average sequential shots count ranges from 0.58 to 3.81 for low- and high-proficiency players. The high sequential shot count ranges from 0 to 145. All results in Table 8 are acquired using the sparse ball configuration. Table 10 shows the results of playing without the author's guidance system under the same conditions as Table 8. The maximum successive sink rate ranges from 2 to 8 for low- to high-proficiency players, and the average sequential shots count ranges from 0.261 to 2.16 for low- to high-proficiency players. Table 12 lists the performance enhancement percentage using the maximum number of sequence shots and average number of sequence shots under a sparse configuration. The statistics shows that high-proficiency player using the system plays 43.3% better than a high-proficiency player who does not use the system; a low-proficiency player plays 55.17% better using our guidance system. This is reasonable, as a low-skill player has poor control in both aiming and reposition control. The performance enhancement can thus be higher than that of the high-proficiency players. The high-proficiency players can also benefit from using our system and algorithm, though with lesser improvements. The high-proficiency players also perform better in their high sequential shot count index by 65.51%. The extent of the enhancement is higher than the other indices. Table 13 lists the performance enhancement percentage in terms of maximum number of sequence shots and average number of sequence shots under cluster configuration. The statistics shows that high proficiency player benefits from using our guidance system by 64% of average number of sequence shots compared to those without using the guidance system. The low proficiency player benefits from using our guidance system by 57.79% compared to those without using the guidance system also by average number of sequence shots. The high proficiency player performs

Table 14. Error percentage between collision model and system in different table states with varying skill level.

Proficiency level	Sparse configuration		Cluster configuration	
	Maximum # of sequence shots (%)	Average # of sequence shots (%)	Maximum # of sequence shots (%)	Average # of sequence shots (%)
High	0	5.69	0	24.13
Medium	25	66.6	0	53.44
Low	62.5	85.64	37.5	85.17

better in high sequential shots count index for 99 percent, while the medium player performs about 100 percent better. This is attributed to the effectiveness of the all pocket algorithm in helping both level of players conquering such hard configuration. However, the low skill player has no improvement. This could be caused by the high perturbation of their strikes in rendering the guided shots, again beyond the scope of this paper.

The performance deviations between the ideal simulation based on the collision model and those operated by users under system guidance can be characterized by the successive sink rate. The performance deviation error between the ideal model and the system depends on users with different skill levels. Table 14 presents such error as a reduction percentage from the sink rate of an ideal player exercising the all-pocket algorithm on two different table states. Two indices are compared: the maximum successive and average sink rates. The ideal player is simulated with an extremely low threshold value of 0.0001 (degree) exercising the all-pocket algorithm. The maximum and average sink rates of such player are extracted from Tables 3 and 5 for the sparse and cluster table states. The performance deviation error is then calculated using the difference percentage of the indices between the ideal and real players with high to low skill proficiencies. Tables 8 and 9 present the sequential shot statistics of different real players exercising the system guidance. Table 14 indicates that the average error percentage is lowest for the high-skill player with both configurations, while the error percentage for the cluster configuration is actually higher. This phenomenon is quite reasonable, as the high-skill player can successfully render most shots, according to the analysis model with minor mistakes. This proves that our system provides a solid test ground for our collision model. The reliability of the integrated system is fully verified and found consistent with the optimal theory. The cluster table state generally has more difficulty succeeding, and bears higher error rates for high-skill players.

6. Conclusion

A design for a billiards training system is proposed to integrate a deflection model, considering restitution effects during collision for tutoring and enhancing the fun of a pool game. To accurately predict the actual rebound motion of two colliding balls, the

Table 15. Reposition strategy performance comparison.

Shot type	Definition	Best used when	Optimization index	Computation complexity
Nearest pocket	Always search for nearest pocket during optimization process	Object balls are close to the jaw of pockets	Tolerance error analysis value using the nearest pocket versus object and cue locations	$O(MN)$ where M is the number of object balls and N is the number of post collision search points
All pocket	Calculate all pockets for optimal post collision paths	Better in cluster and high difficulty pool table states, and about medium high performance in the easy configuration	Tolerance error analysis value using all the pockets versus object and cue locations	$O(MNK)$ where M is the number of object balls, N is the number of post collision search points, and K is the number of pockets
Multi-objective	Calculate all pockets for optimal post collision paths	Best for very low threshold or high proficiency player	Combined tolerance error and inverse of attack angle	$O(MNK)$ where M is the number of object balls, N is the number of post collision search points, and K is the number of pockets

collision physics are analyzed to predict the accurate deflection direction. Three novel gaming strategies are simulated to investigate the effects of the target object ball and pocket combinations and the multi-objective optimization criterion on gaming performance. The first algorithm considers the nearest pocket for every selected target object ball, examining optimal post-collision positions. The second algorithm considers all pocket and target object ball combinations during both the pre- and post-collision optimal shots selection processes. The third algorithm considers a multi-objective optimization process for optimal cue repositioning control. The objective functions considered include the tolerance angle and the inverse of the attack angle. The goal is to find an optimal combination of both quantities, as we wish to maximize the tolerance angle and minimize the attack angle. The simulations are conducted based on a collision model considering the restitution effects. Different threshold values are added to emulate the real-world proficiency levels of different players. Two different object ball distributions with different difficulties are also selected by testing the different algorithms under different threshold values. The difficulty of ball distributions is a new term defined in this work and evaluated as the sum of the distance from object balls to their nearest pockets. The first ball configuration used to test the algorithms is a sparse distribution, with every ball around the jaws of each pocket; the second configuration is a more clustered distribution with each ball farther from each pocket.

Table 15 summarizes the pros and cons of different reposition strategies and compares their performances. The table compares the three algorithms based on indices, including their definitions, usage occasion, optimization index, and computation complexity. The nearest pocket algorithm searches for the nearest pocket during the optimization process. This algorithm can outperform other algorithms while the object balls are near pockets. Its complexity is in the polynomial order of the power of two. The all-pocket algorithm adds one extra search to the pocket combinations. The all-pocket algorithm outperforms the other two algorithms in the high-difficulty ball and cluster configurations, in the medium- to high-threshold value ranges. Its performance is medium among the three algorithms in the easy configuration. Although the multi-objective algorithm outperforms the other algorithms in the extremely low threshold value range, it is not comparable to other algorithms for medium to high threshold values. This indicates that the all-pocket algorithm is suitable for application in the general gaming scenario, as the extremely low threshold value is rarely observed in real gaming scenarios for users who may or may not use the guidance system; it is thus recommended for integration with the visual guide system. Table 15 also summarizes the complexity of each algorithm, indicating that the complexity of both the all-pocket and multi-objective algorithms approximately reach the polynomial order of the power of three.

We then opt to adopt the all-pocket algorithm in the integration process with the visual guide system due to its optimal simulation performance. Our visual guide system uses a vision system for the cue ball, object ball locations and cue stick velocity tracking. Users can adjust the cue stick's aiming direction and hitting velocity on the pool table according to the visual guides displayed on a PC monitor. The guidance information is repeatedly updated and superimposed onto the image stream of the pool table. A least square error calibration process correlates the real and virtual world pool ball coordinates to precisely calculate the guidance line and cue stick speed. Adding cue placement control, including aiming direction, ghost ball, and optimal cue position for the following shot, is analyzed and displayed on a GUI. The all-pocket game strategy is selected to apply the maximum tolerance angle search sequentially on the first and subsequent shots along the post-collision paths, considering the restitution factor.

Experimental results of a maximum tolerance angle shot planning strategy using our training facility, tested by users with different skill levels, outperformed the results without guidance for the same user set on several indices. A high-skill player generally exhibits lesser enhancements, while a low-skill player exhibits greater enhancements in the different ball configurations. This is intuitively reasonable, as low-skill players have poor stroke control and ball placement judgment. They can benefit more from the aiming and ball placement guides in our system than the easier ball configurations. The performance statistics from high-skill players using the guidance system exhibit lower deviation error than the simulation results for different ball configurations. This not only proves the reliability of our training device in selecting proper shot sequences using the all-pocket shot planning algorithm, but it also proves the consistency between the experiments and the theory of shot planning strategies when guiding users towards optimal performance.

Future research will include exploring the search space on the post-collision path. A greedy search method is currently used on the path. A Monte Carlo search method on the post-collision path sequence will be conducted. The effect of ordering the attacked object balls in a Monte Carlo search will also impact the optimal performance. These methodologies will be studied in future papers.

Acknowledgments

The author wishes to thank the financial support of National Science Council of Taiwan for completion of this work under contract number of NSC 99-2622-E-029-010-CC3 and NSC 99-2220-E-029-003.

References

1. C. Shih, Zero tolerance cue angle analysis and its effect on successive sink rate of a low cost billiard reposition control tutoring system, *Knowledge-Based Systems* **30** (2012) 17–34.
2. M. Jouaneh and P. Carnevale, The development of an autonomous robotic system for playing mini-golf, *IEEE Robotics & Automation Magazine* **10**(2) (2003) 56–60.
3. P. Grogono, Mathematics for snooker simulation, personal communication (2001).
4. F. C. A. Groen, G. A. den Boer, A. van Inge and R. Stam, Chess playing robot, *IEEE Transactions on Instrumentation and Measurement* **41**(6) (1992) 911–914.
5. L. B. Larsen, P. M. Jensen, K. Kammersgaard and L. Kromann, The automated pool trainer—a multi modal system for learning the game of pool, in *Intelligent Multimedia, Computing and Communications: Technologies and Applications of the Future by IEEE Computer Society Press* (2001), pp. 90–96.
6. Z. Lin, The study of a billiard robot, Thesis, Department of Mechanical and Electro Mechanical Engineering, TKU, Taiwan, R.O.C. (2003).
7. S. C. Chua, E. K. Wong, A. W. C. Tan and V. C. Koo, Decision algorithm for pool using fuzzy system, in *Proc. Int. Conf. Artificial Intelligence in Engineering & Technology (lCA1ET 2002)*, Kola Kinabalu, Malaysia, 17–18 June 2002, pp. 370–375.
8. B. R. Cheng, J. T. Li and J. S. Yang, Design of the neural-fuzzy compensator for a billiard robot, in *Proc. 2004 IEEE Int. Conf. Networking, Sensing & Control*, Taipei, Taiwan, 21–23 March 2004.
9. W. Leckie and M. Greenspan, Pool physics simulation by event prediction 1: Motion transitions, *International Computer Games Association Journal* **28**(4) (2005) 214–222.
10. W. Leckie and M. Greenspan, Pool physics simulation by event prediction 2: Collisions, *International Computer Games Association Journal* **29**(1) (2006) 24–31.
11. C. Chou, M. Tien and J. Wu, Billiards wizard: A tutoring system for broadcasting nine-ball billiards videos, *IEEE Int. Conf. Acoustics, Speech, and Signal Processing*, April 2009, pp. 1921–1924.
12. J. Lander, Physics on the back of a cocktail napkin, *Game Developer* (September, 1999), pp. 1–4.
13. T. Jebara, C. Eyster, 1. Weaver. T. Starner and A. Pentland, Stochastics: Augmenting the billiard experience with probabilistic vision and wearable computers, in *Proc. Int. Symp. Wearable Computers* (Cambridge, Massachusetts, 1997), pp. 138–145.

14. C. Shih, P. A. Hsiung, C. H. Wan, C. S. Koong, T. K. Liu, Y. Yang, C. H. Lin and W. C. C. Chu, Integration of a vision-based tracking platform, visual instruction, and error analysis models for an efficient billiard training system, *Optical Engineering* **48**(2) (2009) 027202-1-11.
15. S. C. Chua, E. K. Wong and V. C. Koo, Pool balls identification and calibration for a pool robot, in *Proc. Int. Conf. Robotics, Vision, Information and Signal Processing (ROVISP 2003)*, Penang, Malaysia, January 2003, pp. 312–315.
16. S. C. Chua, E. K. Wong and V. C. Koo, Performance evaluation of fuzzy-based decision system for pool, *Applied Soft Computing* **7** (2007) 411–424.
17. H. Nakama, 1. Takaesu and H. Tokashiki, Basic study on development of shooting mechanism for billiard robot, *JSME, Robotic Workshop, IAIF8* (2001).
18. M. Greenspan, J. Lam, M. Godard, I. Zaidi, S. Jordan, W. Leckie, K. Anderson and D. Dupuis, Toward a competitive pool-playing robot, *Computer* **41**(1) (2008) 46–53.
19. M. Smith, PickPocket: A computer billiards shark, *Artificial Intelligence* **171**(16–17) (2007) 1069–1091.
20. T. Braunl, Research relevance of mobile robot competitions, *IEEE Robotics & Automation Magazine* **6**(4) (1999) 32–37.
21. K. Kiguchi, K. Watanabe and T. Fukuda, Design of a human-like desired grasping force planner for object manipulation by robot manipulators, *Machine Intelligence Robotic Control* **3**(4) (2002) 167–173.
22. K. Hashimoto and T. Noritsugu, Modeling and control of robotic yoyo with visual feedback, *Proc. IEEE Int. Conf. Robotics and Automation*, Vol. 3 (1996), pp. 2650–2655.
23. H. Nakai, Y. Taniguchi, M. Uenohara, T. Yoshimi, H. Ogawa, F. Ozaki, J. Oaki, H. Sato, Y. Asari, K. Maeda, H. Banba, T. Okada, K. Tatsuno, E. Tanaka, O. Vamaguchi and M. Tacyhimon, Volleyball playing robot, in *Proc. IEEE Int. Conf. Robotics and Automation*, Vol. 2 (1998), pp. 1083–1089.
24. J. Hoffman and E. Malstrom, Teaching a miniature robotic manipulator to play chess, *Robotica* **1**(4) (1983) 197–203.
25. L. Acosta, J. J. Rodrigo, J. A. Mendez, G. N. Marichal and M. Sigut, Ping-pong player prototype: A pc-based, low-cost, ping-pong robot, *IEEE Robotics & Automation Magazine* **10**(4) (2003) 44–52.
26. J. Francois and L. J. Dussault, AI optimization of a billiard player, *Journal of Intelligent and Robotic Systems* **50** (2007) 399–417.