Programming Assignment 3
Report

Kanak Yadav, 20D070044

November 5, 2023

1 Approach

1. Find the ball closest to the cue ball.
2. Find the hole/pot that is closest to the selected ball.
3. If it is "feasible" proceed, else choose the next closest hole.
4. Simulate hitting the ball.
5. If the "oscillatory condition" occurs, choose a different ball; if only a single ball is left or otherwise if the condition doesn't occur, perform the action.

2 Implementation

2.1 Closest ball/hole

Used the Euclidean distance as the criteria. The ball with the least distance from the cue ball and the hole with the least distance from the selected ball are chosen.

2.2 Hole Feasibility

A hole if feasible, if the location where the cue ball is required to be when it makes contact with the target ball cannot be reached without colliding with the target ball at some other location first, i.e. a situation similar to Fig 1.

In code, this is checked by observing that the coordinates of the cue, ball, and the position of the cue for collision are such that the ball is bounded between them in either ascending or descending order and in either of the coordinates for the unfeasible cases.

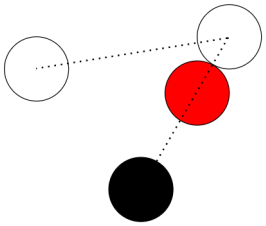

Figure 1: An unfeasible hole

2.3 Oscillatory condition

It was occasionally observed that while trying to pot a ball very close to a hole and also close to the boundary of the table, the cue ball would miss the target, bounce back, and return to approximately the same location it was in before.

Referring to the above condition as the oscillatory condition, it was tackled by using the get_next_state() function and checking the distance between the positions of the cue ball in the respective states.

As stated above, a different target is chosen in case this distance is less than a certain threshold (7 for the tests done locally).

2.4 Action

2.4.1 Force

The force required is calculated as

$$\text{Force} = 0.98 * \frac{\text{Total Distance}}{\text{Length of the table in the X direction}} \tag{1}$$

Where the total distance is the sum of the distance from the cue ball's initial to its final position and the distance between the target ball and the selected (feasible) hole.

The factor of 0.98 was found to work better as compared to 1 during local testing.

The length of the table in the X direction is approximated as the difference between the x-coordinates of the two holes that are located at the extremes.

2.4.2 Angle

First, we find the angle between the ball (xb, yb) and the hole (xh, yh) as:

$$\theta_1 = \tan^{-1}\left(\frac{x_h - x_b}{y_b - y_h}\right)$$

using the math.atan2() function, which returns values between -π an π.

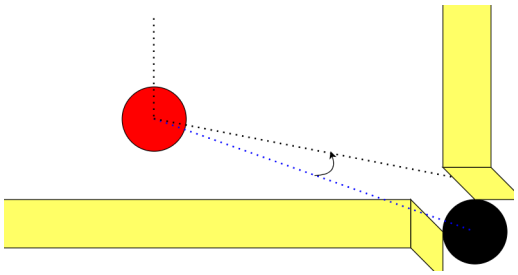

Figure 2: Angle correction

But, this angle may not always be indicative of the exact direction where we want the target ball to go, for example, consider the blue dotted line in Fig 2. Thus we perform an angle correction step on this angle to avoid such cases.

The angle correction is basically just a linear mapping of the component of the angle from 0-45 degrees to 0-45*factor degrees, and from 45-90 degrees to 45*(1+factor)-90 degrees, for each quadrant. After some local testing, the factor was determined to be 10/11.

Next, we find the final position of the cue ball required to provide momentum in the correct direction:

$$x'_c = x_b - 0.95(2r)\sin(\theta_1)$$

$$y'_c = y_b + 0.95(2r)\cos(\theta_1)$$

Where, (xc', yc') are the final coordinates of the cue ball and r is the radius of the balls.

The factor of 0.95 is used to tackle the noise added to the angle it was finalized after a few test runs.

Now, the angle at which the cue ball needs to go is calculated as:

$$\theta_2 = \tan^{-1}\left(\frac{x'_c - x_c}{y_c - y'_c}\right)$$

Where, (xc, yc) are the coordinates of the cue ball as given in the current state.

θ_2 is the angle at which the cue ball needs velocity (positive taken to be in the counter-clockwise direction from the negative Y axis). However, the angle required by the program to perform this action is the negative of θ_2, scaled to be in the range [-1, 1], because the game uses the angle to align the cue stick (instead of the velocity vector of the cue ball).

$$\text{angle} = -\frac{\theta_2}{\pi} \tag{2}$$

The function returns the tuple (angle, Force) obtained in the equations 2, and 1 respectively.