Aerial Robotics Kharagpur Documentation Template

Neha Dalmia

Abstract— **This task involved creating a billiard game like simulation in which a ball is initially hit by a cue thereby deciding its initial trajectory and collisions with the remaining balls and the walls are assumed to be perfectly elastic. The angle of collision with a ball determines the trajectory of the ball hit and this ball becomes the one in motion. In this task I used the slope of the line joining the centers of the colliding balls to obtain the direction in which the ball hit will be travelling. In case of wall collisions the slope is simply multiplied by -1 to obtain the changed path. the condition for collisions betweens walls were r_1+r_2 is equal to C_1C_2. The conditon for collision with wall is whether the distance between the center of the ball and wall plane is equal to the radius of the ball. However, these conditions have to be relaxed a little as due to extremely high and extremely small slope values, the values of y and x xoordinates respectively change by a very high high magnitude and hence the ball might enter the wall or the ball it has collided with as well. Hence the conditions have been relaxed to if r_1+r_2 is less than or equal to C_1C_2 and in case of wall collisons if distance between the center and wall plane coordinates are less than or equal to the radius of the ball. With these situations and certain cases like infinite slope and zero slope conditions having been kept in mind the code has been written.**

Predicting the trajectory of an object in different situations based on certain extraneous factors can have wide applications when it comes to predicting path of a flying object in different wind speeds and similar flying conditions.

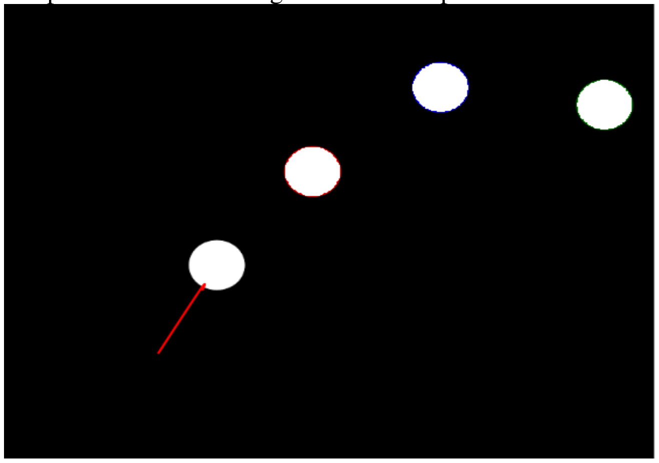
I. INTRODUCTION

In this problem statement we were given images of a billiard pool table with the cue and ball to be hit indicated and we find pairs of collisions assuming perfectly elastic ball-ball collisions and ball-table collisions. The balls' centers might not be in-line and this would have lead to two divergent ball movements but we assume that in case of ball-ball collision, the incoming ball stops and the incident-ball moves with the same velocity as the incoming ball.

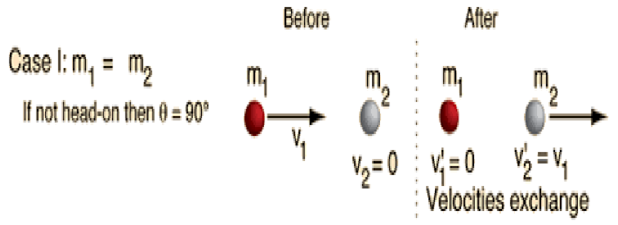
I decided to make a simulation actually showing the various collisions taking place as opposed to just calculating and displaying the coordinates of the collision. The ball in motion is shown in white and the stationary balls in red. I have done this using the basic concept of elastic collisions ie the ball hit will move in the direction of the slope joining the centers of the colliding balls.

II. PROBLEM STATEMENT

In the problem statement we have been given an image representing a billiard table with a cue and different balls. We have to identify the ball being hit by the cue and predict the collisions that will take place between this ball and the other balls. In these collisions, we have assumed that the ball initially in motion comes to rest after colliding and there happens a complete transfer of velocity to the ball which was hit. In real world scenarios, this kind of collision will take place only if the collision is perfectly elastic ie no friction exists between the colliding balls and both kinetic energy and momentum are conserved and if the two balls have the same mass. The problem statement has been designed taking into consideration these 2 assumptions. I have taken a simulation based approach of dealing with this problem, in which the collisions are actually visible to the viewer. Below is an example of the billiard board setup we have been given in the problem statement.



The case of elastic collision we have been required to use can be illustrated by the image below:



The problem statement does not explicitly mention the number of collisions that need to be recorded or the time duration till which we have to continue the collision. Thus we demonstrate it via an infinite loop till a counter reaches a particular value. The demonstration via visual display of collisions has not been explicitly mentiones as well.

III. RELATED WORK

Other approaches to solving this problem can involve heavy calculation based approaches rather than a visual approach. These will involve calculating the coordinates where collisions can occur using a reference coordinate system with the coordinates of the wall planes being stored as well as storing coordinated of non colliding balls. The challenges of visual display are omitted in method.

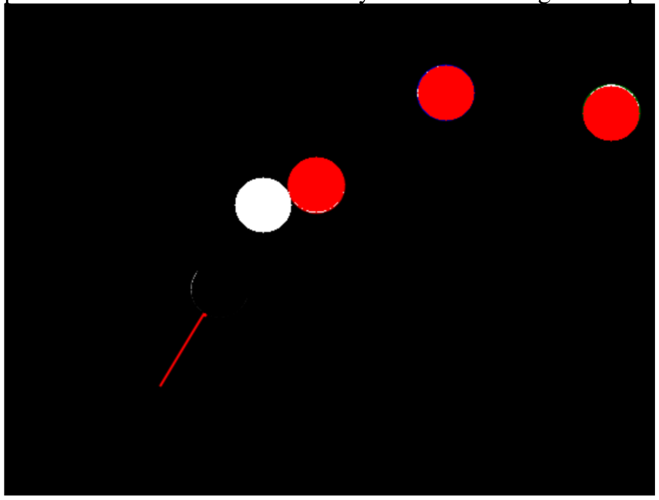
IV. Initial Attempts

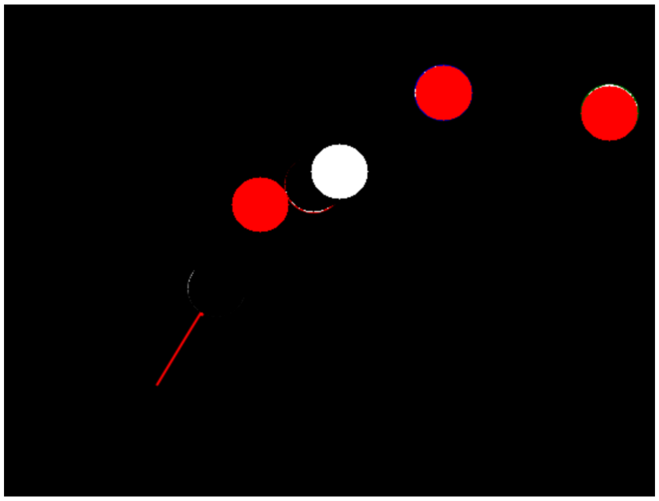
My initial approach to solving this problem was very similar to the final approach I ended up using. Initially instead of showing the movement of the ball and shifting of motion between balls, I was simply calculating the co-ordinates at which collisions were taking place and using lines to represent the paths that had been taken. However for a large number of collisions this display looked extremely messy. Hence I decided to show the movement of the balls by drawing coloured and black balls at the same position.

V. Final Approach

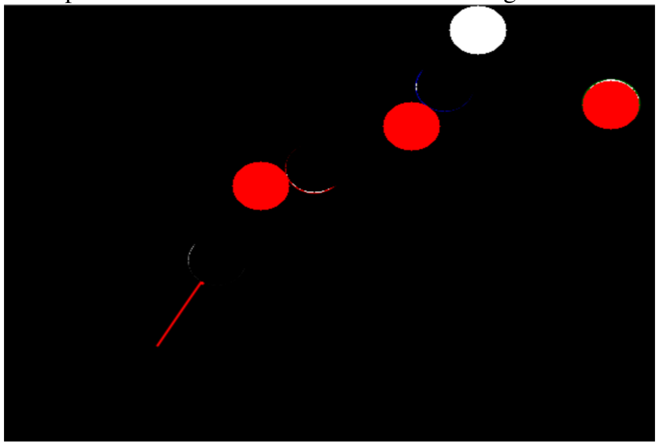
General Overview of my approach: In my approach, I am visually displaying the path of the moving ball and the collisions that take place during its motion. My approach starts with first finding the initial slope along which the ball will move by detecting the cue using **Hough Lines** and finding the ball closest to either of its end coordinates. This ball is currently our 'x' ball which is going to move. The balls are detected using **Hough Circles** and we thus obtain their center coordinates and radii as well. The slope of the line along which it is going to move is obtained using the end coordinates of the cue. The motion of the moving ball throughout the code is showing by drawing the moving ball in white and redrawing it in black at the same position using **cv2.circle** method of opencv.

Ball with ball collisions: To detect ball collisions I used the geometric concept that **distance between centers of the balls is less than or equal to the sum of their radii**. However, due to the availability of only integer coordinates and non unity increasing of x and y coordinates, a compensation of approximately 10 pixels had to be made to ensure proper detection in a wide variety of cases including the ones in which a slight overlap between the two balls occurs as well. After the collision is detected, the line joining their centers is obtained and the ball which was hit is made the 'x' ball which now traverses this line. After the 'x' ball is updated, its made to travel 4 steps along this new line to ensure that in the continuation of the loop it is not recolided with the previous ball. This is more likely to occur during overlaps.





Ball with wall collisions: To detect collisions of balls with the respective walls, I used the fact that whell a ball collides with a wall, the **distance of the center of the ball with the wall plane will be equal to the radius** of the ball. We also allow values less than the radius in case of slight overlapping due to x or y values increasing with high magnitude. After this collsion the slope of the ball is multiplied by -1 to indicate the change in direction. In certain wall collision cases, the increase or decrease of x and y values will be reversed. This is dealt with using the variable **choice** which decides whether the x coordinate increases or decreases and is updated during wall collisions accordingly. The wall plane coordinates are 0 to screen height or width.



Updating Coordinates: The coordinates of the **center** of ball 'x' need to be updated in every iteration. This is done by calling the method **alter**. This involves checking for numerous cases. As we know, the coorinates have only integer values so approximations have to me made. Here we first check if the slope is tending to infinity or not. If not, we check whether it's range is within -1 and 1. If yes, this implies that the increase in x coordinate will be much more significant than the y coordinate. Hence, depending on the sign of the slope and on choice, we increment or decrement the y coordinate by one and increment or decrement the x coordinate using the change in y coordinate and the slope value. For slopes not falling in this range, we change the x coordinate by unity depending on choice and change the y coordinate using the slope depending on its magnitude and sign. In case our slope value is **tending to infinity** we just increment or decrement the y coordinate depending on the value of variable special.

Approximations: The first major approximation I have taken is that **slope angles greater than 86.2° are taken to be 90°** and hence the slope is taken to be infinity. Another approximation that I have to inevitably concede to due to the lack of availabilty of floating point coordinates is the **restriction in the slope values**. If we change the y coordinate by one, the x coordinate can only be changed by integral numbers ie 1,2,3... Similar situation arises when change the x coordinate by 1. Thus the values taken up the slope can be ....$\frac{1}{4}$,$\frac{1}{3}$,$\frac{1}{2}$,0,1,2,3,4.....This restricts the movement of our ball considerably and can lead to slight inaccuarcies. High slope values lead to higher magnitude of decrement or increment of the x and y center coordinates as well. It is possible for a ball to move 11 spaces in x or y direction in simply one iteration. Hence a ball whose center is at a distance of 5 units from another ball at an instant can be inside the ball by one unit in the very next iteration. Same applies for wall collisions and hence alterations have been made in standard geometric formulae to accommodate these **overlapping errors** which are very likely to occur. This tremendous increase in slope magnitudes also increases the velocity of the moving wall which fundamentally violates the laws of physics. To stop this from seeming very obvious, a waitKey has been used to nullify this effect as much as possible.

Miscellaneous problems faced: One notable problem faced by me was the successive recognition of collisions between the same two balls as the coordinates were not updated before the next collision check. This problem was magnified in case of slight overlaps where despite the coordinate update, the distance moved by the ball might not be large enough owing to changed slope value to take it out of the other ball completely and hence will be treated as a separate collision. To avoid this, after every collision the ball being hit has its center coordinates updated by 3 iterations to make it sufficiently far from the ball it was hit by yet not far enough to miss any actual collisions that may occur. Initially I was faced with a lot of errors as I had set the data type of the numpy array of the image as uint16 and yet my calculations involved negative values as well as large numbers. The determination of the increase or decrease values of x and y coordinates also required a lot of attention due to the rounding off of the integers of the slope.

VI. Results and Observation

As opposed to this visual approach we could've used Hough Lines and Hough Circles to just read the values of the coordinates of centers and radii of circles and end points of the cue and instead of displaying the movement of the ball just displayed the coordinates at which the collision would have taken place. We could have done this using the same basic program structure but in this case the slope could have taken a much larger range of values due to floating point increment and decrement till even two decimal places. Also, infinity would not have had to be approximated in this case.

Despite some errors arising due to approximation I chose the visual approach as opposed to the approach mentioned above. This is because I felt the whole essence of collision can be understood only when we can see it happening and hence I chose the **visual appeal and demonstration despite slight errors due to approximations**. This was also because on calculating, the errors were not coming out to be very significant, apart from the error due to the approximation to infinity.

VII. Future Work

The major problems in my algorithm arise due to the large number of approximations involved. These problems can be solved if we can show the movement of the ball in a graph based background where the circle can be drawn in floating point coordinates as well. The current algorithm too can be improved by using better calculations to make the approximations more fitting to the entire procedure. A floating-point coordinate plotting system will be ensure a close to error free visual approach to the problem statement.

Conclusion

The task required us to stimulate a billiard ball around a billiard board using the concept of perfectly elastic collisions between balls of the same mass. I used a visual approach in which I displayed the ball as it moved across the board and as it collided with the other balls. This task enables one to get very well acquainted with opencv as welll as helps us to identify collisions of circles which can be helpful if a drone is used as a tracking device or as a monitor for detecting car crashes on roads etc.