Billiard Simulation

1 Introduction

This exercise examines the motion of two disks falling within an enclosed container, where they interact with both each other and the container walls. The disks are influenced by gravity and friction forces, and their collisions demonstrate pure elastic behavior.

2 Equations of Motion and Their Approximations

Creating a functional billiard simulation requires incorporating and understanding multiple equations. For this simulation, we applied equations of motion and utilized the Euler method for numerical approximations.

An essential variable to define in this simulation is **dt**, which represents the time step. This variable determines the time interval for a disk to travel from one position to another at a given velocity. Initially, **dt** is assigned a value of 0.2; however, it may vary throughout the code. The factors influencing changes in **dt** will be discussed later.

Two important equations for this billiard simulation are:

$$\begin{aligned} dx/dt &= u \\ dy/dt &= v \end{aligned} \tag{1}$$

These equations are essential for approximating the disk's next position. Here, the variable u represents the x-velocity of the disk, while v represents the y-velocity. Using these equations, the position of the disk at the next time step can be calculated as follows:

$$\begin{aligned} xnprprediction &= xnr + dt * unr \\ ynprprediction &= ynr + dt * vnr \\ xnpbprediction &= xnb + dt * unb \\ ynpbprediction &= ynb + dt * vnb \end{aligned} \tag{2}$$

[1]In these equations, new variables are introduced: xnprprediction, xnr, and unr. In this simulation:

- xnprprediction represents the predicted x-position of the red disk after the next time step.
- xnr denotes the current x-position of the red disk.
- unr refers to the x-velocity of the red disk.

Similarly, for the y-direction, the variables ynprprediction, ynr, and vnr represent the predicted y-position, current y-position, and y-velocity of the red disk, respectively.
For the blue disk, analogous variables are denoted with a 'b', such as xnpbprediction, xnb, and unb, to indicate the x and y components for the blue disk.

The next crucial step in this simulation was to calculate the updated velocities of the disks over time. In this simulation, the disks are subject to the forces of gravity and friction, which cause their velocities to change dynamically. To account for these factors, we derived the equations for the velocities in the x and y directions as functions of time. By taking the time derivative of the velocities, we obtained the following equations:

$$\begin{aligned} du/dt &= -\frac{c_d}{m}||\mathbf{v}||u \\ dv/dt &= g - \frac{c_d}{m}||\mathbf{v}||v \end{aligned} \tag{3}$$

The equations are nearly identical, with $\mathbf{c_d}$ representing the drag coefficient, which serves as the friction coefficient in this simulation. In this case, its value is set to 0.25, as the disks are observed moving through air. Additionally, $||\mathbf{v}||$ denotes the magnitude of the velocity vector, combining the x(u) and y (v) velocity components. This magnitude is then multiplied by the respective velocity components (u) and (v) to account for the drag force in each direction.

The gravity constant ($\mathbf{g}$) is applied only to the y-component of velocity since gravity has no x-component and, therefore, does not influence the x-direction velocity.

Using Euler's method, we can derive another set of equations to calculate the velocities at the next time step. These equations incorporate the effects of drag and gravity to update the x and y velocity components dynamically as the simulation progresses.

$$\begin{aligned} unpbprediction &= unb + dt(-\frac{c_d}{m} * velocityb * unb) \\ vnpbprediction &= vnb + dt(-g + \frac{c_d}{m} * velocityb * vnb) \\ unprprediction &= unr + dt(-\frac{c_d}{m} * velocityr * unr) \\ vnprprediction &= vnr + dt(-g + \frac{c_d}{m} * velocityr * vnr) \end{aligned} \tag{4}$$

[1]New variables unpb, vnpb, and velocityb describe the velocity components for the blue disk:

- unpb represents the updated velocity in the x-direction for the blue disk.
- vnpb represents the updated velocity in the y-direction for the blue disk.

[1]variables introduced in (2) and (4) reference variables as they are written in the Matlab code.

- velocityb represents the magnitude of the velocity vector for the blue disk.

Similarly, variables denoted with a 'r' (e.g., unpr, vnpr, velocityr) represent the same components but correspond to the red disk.

3 Numerical Treatment of Collisions

3.1 Collision between Walls

When the disks interact with the walls, several variable values change to account for the collision dynamics. Coding for wall collisions involves following a series of steps to accurately update the disk's position and velocity after the impact.

1. **Detect the collision:** A condition must be written to indicate a collision with a wall. This condition will vary depending on which wall the disk collides with.

   - For the **right wall**, a collision occurs when the predicted x-position of the disk exceeds the maximum x value minus the disk's radius. The condition is:

     ```
     >> if xnprprediction > xmax - r %red ball collision with right wall
     ```

   - For the **left wall**, a collision happens when the predicted x-position of the disk is less than the sum of the minimum x value and its radius. The condition is:

     ```
     >> if xnprprediction < xmin + r %red ball collision with left wall
     ```

   - For the **top wall**, a collision occurs when the predicted y-position of the disk is greater than the maximum y value minus its radius. The condition is:

     ```
     >> if ynprprediction > ymax - r %red ball collision with top wall
     ```

     For the **bottom wall**, a collision happens when the predicted y-position of the disk is less than its radius. The condition is:

     ```
     >> if ynprprediction < r        %red ball collision with top wall
     ```

2. **Adjust the value of dt:** As discussed earlier, the value of dt will change when a collision occurs. The formula for calculating the new value of dt should be given by:

   $$dt = |distance|/|velocity| \tag{5}$$

   The position of the disk should be used according to which wall it is colliding with.

- If colliding with the right or left walls, the x position of the disk should be used to evaluate the distance to the wall.
- If colliding with the top or bottom walls, the y position of the disk should be used to evaluate the distance to the wall.

This ensures that the distance calculation accounts for the correct dimension (x or y) based on the wall the disk interacts with.

3. **Update x and y:** With a new value of dt, the prediction values for the x and y positions will no longer be accurate. To correct this, equation (2) should be used to update the positions in the x and y directions based on the new time step. It is helpful to rename the predicted position variables at this stage to reflect the updated values.

4. **Update u and v:** Similarly, the prediction values for the velocities will also need to be updated. (4) should be used to calculate the new velocities. Again, renaming the variables may be useful.

5. **Account for friction:** The final step is to account for the friction forces that the disk encounters when interacting with the wall. We use coefficients α and β, which control the damping effects of friction.

   α governs how much speed is lost in the horizontal direction.

   - If α is less than one, there is a loss of speed in the horizontal direction.
   - If α is greater than one, there is a gain in speed.
   - If $\alpha = 1$, there is no loss or gain in speed.

   In this simulation, α will always be a **negative value**, which simulates a loss of speed in the horizontal direction after a collision.
   β controls the amount of friction the disk experiences, influencing the velocities after the collision.

   Whether α and β affect the x or y velocities will depend on which wall the disk collides with.

   For example:
   For the left or right wall (horizontal walls), α will affect the x velocity and β will influence the magnitude of the velocity in the horizontal direction.
   For the top or bottom wall (vertical walls), α will affect the y velocity and β will influence the magnitude of the velocity in the vertical direction.

3.2 Collision between Disks

Collisions with disks can be treated in similar steps.

1. **Detect the collision:** The two disks will collide with each other when the magnitude of the distance between them is less than twice the radius.

2. **Recalculate dt:** The equation to calculate dt has some important distinctions,

$$dt = (distancemag - 2r)/velocitiesrnb, \tag{6}$$

   where 2r is subtracted from the magnitude of the distance (distancemag) to account for the radius of the two disks. The magnitude of the velocities of the two object is the velocitiesrnb.

3. **Finding Collision:** Using the new dt, collision location can be calculated. (2) is sufficient in calculating this.

4. **Collision Velocities:** Using the new dt, collision velocities can be calculated. (4) is sufficient in calculating this.

5. To determine the exit velocities, we first need to calculate the velocity with which the disks collide. We use the velocities from step 4 to determine the exit velocities. Using these velocities, we can find the vector that predicts the exit velocity. The equation to determine the velocity component is:

$$vrin = (vrin \cdot n) \cdot n + (vrin \cdot tau) \cdot tau \tag{7}$$

   Here, n represents the normal vector of the velocity, and tau is the tangential component. The dot product must be taken to find the velocity vector in. The normal vector can be calculated by finding the unit vector of the difference in the distances between the two disks, producing a column vector:

$$\begin{bmatrix} n_1 \\ n_2 \end{bmatrix}$$

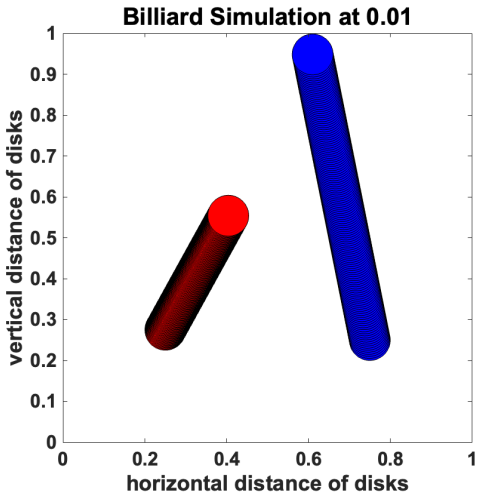
   Tau can be found by using the components of the normal force and be written as:

$$\begin{bmatrix} n_2 \\ -n_1 \end{bmatrix}$$

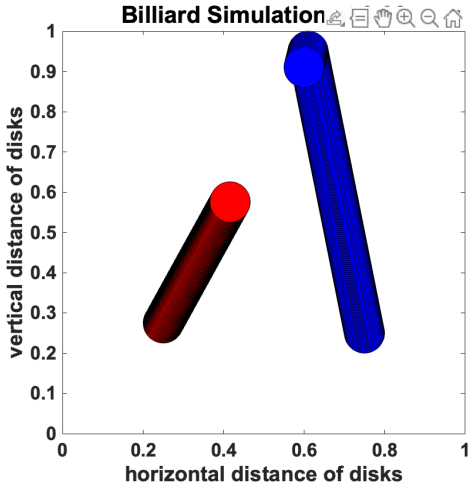
   Finally, the exit velocities can be found by calculating the dot product between the velocity and the normal and tau vectors. However, the blue and red disks will exchange their normal component velocities, forcing them to exchange velocities. Since we are dealing with a pure elastic collision, we can ignore any damping or friction due to the collision.

$$\begin{aligned} vrout &= (vrin \cdot n) \cdot n + (vrin \cdot tau) \cdot tau \\ vbout &= (vbin \cdot n) \cdot n + (vbin \cdot tau) \cdot tau \end{aligned} \tag{8}$$

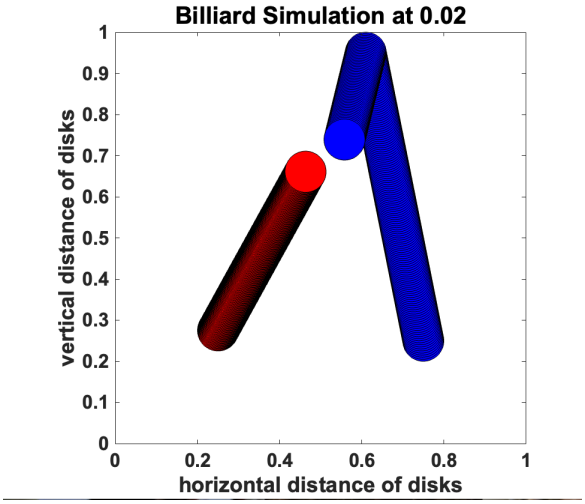
Since this is in the tau and n reference frame, we need to shift it back to the i and j reference frame by using the dot product between the velocity out and the i and j components, where i corresponds to the x velocity and j corresponds to the y velocity.



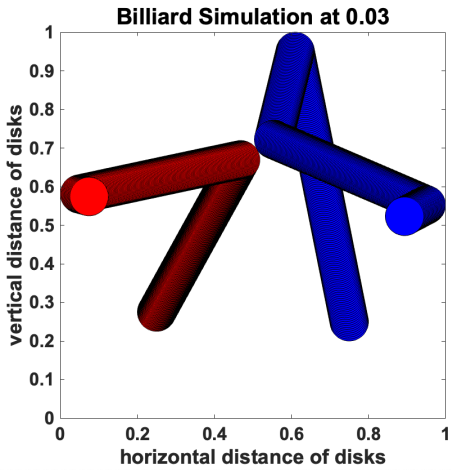
(a) blue disk before collision with the upper wall



(b) blue disk after collision with the upper wall



(c) disks before colliding with each other



(d) disks before colliding with each other

Figure 1: Comparing disk behavior before and after collision

4 Results

Referring to Figure 1, we can visualize the behavior of the ball before and after the collision. In Figures 1(a) and 1(b), we can observe the movement of the blue disk. The blue disk approaches the top wall along a linear path, steadily gaining momentum until it contacts the wall. Upon collision, the ball rebounds in the opposite direction, with its motion mirroring the path it took before impact. Although this rebound is visible, it's important to note that the speed of the ball has experienced a slight reduction due to the damping effect that is not directly shown in the figures.

In Figures 1(c) and 1(d), we can observe the behavior of the disks during and after their collision with each other. As the two disks approach one another, they collide, and due to the nature of the elastic collision, they exchange their velocities. This results in the disks moving in opposite directions, each reflecting the velocity of the other prior to the collision. This exchange of velocities follows the principle of conservation of momentum, and as a result, both disks continue their motion in different trajectories after the impact. The figures help to illustrate how the energy is conserved and how the disks behave in a dynamic interaction.

5 Conclusion

This code can serve a wide range of purposes, from educational tools to applications in game development. One of its primary uses is in demonstrating pure elastic collisions, a fundamental concept in physics. By simulating disks that collide with one another and with walls in a perfectly elastic manner, students can visualize how objects exchange velocities without energy loss. This makes it a useful resource for physics students to explore the behavior of colliding objects under ideal conditions.

Moreover, this simulation offers a clear demonstration of Newton's laws of motion, especially the second and third laws. Newton's second law, **F=ma**, is clearly observed in the behavior of the falling disks as they accelerate due to gravity. The force acting on the disks results in the change in their velocity, allowing students to visualize the relationship between force, mass, and acceleration. Additionally, the third law of motion, which states that every action has an equal and opposite reaction, is demonstrated in the simulation through friction forces and the drag resistance encountered by the disks as they move through the air. The interaction between the disks and the walls also provides a direct example of action and reaction forces.

In a more applied context, simulations like this could play a crucial role in the development of video games and other interactive digital environments. Video games often require realistic object interactions and collision mechanics, especially in physics-based games or sports simulations. For example, in a game where players shoot basketballs, the principles of velocity, angle, and elasticity demonstrated in this code could be used to simulate the ball's motion and its interaction with the hoop. As players shoot the basketballs, the simulation of the ball's trajectory, speed, and collisions with other balls or

walls could mirror real-world behavior, offering a more immersive and accurate gaming experience.

Many games, involving any form of object-based interaction, would benefit from this type of code. By adjusting the physics parameters, game developers could use this simulation to create more dynamic and realistic game environments, where objects interact in ways that closely resemble real-world physics.