**Documentation for “Hit a Ball to The Fixed Target” Project**

**Overview**

This project combines image processing, numerical simulation, and animation to model, detect, and visualize the motion of balls in a 2D environment. The workflow includes detecting circular objects in an image, calculating their trajectories for newly created ball to hit the target balls using physical principles, and creating an animation to visualize their motion and interactions. The implementation relies on Python with key libraries like OpenCV and NumPy for computational efficiency and visualization.

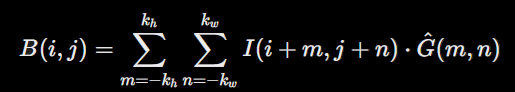
**1. Circle Detection**

**Objective**

To identify circular objects in a given input image and determine their locations and radii. This is a prerequisite for simulating their motion.

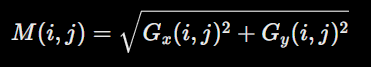
**Key Methods**

1. **Image Preprocessing**: The input image is read and converted to grayscale to simplify processing. This step reduces computational complexity and improves edge detection.
2. **Gaussian Blurring**: Noise in the image is reduced by applying a Gaussian blur. The custom gaussian blur function convolves this kernel with the image.
3. The kernel is computed as:

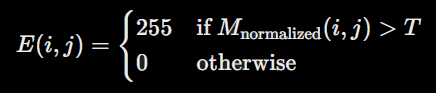


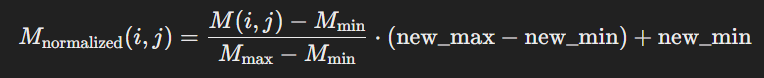
Where:

B(i,j)B(i, j)B(i,j): The resulting blurred intensity at pixel (i,j)(i, j)(i,j).  
I(i+m,j+n): The intensity of the pixel in the neighborhood of (i,j), with m, n as offsets.  
G(m,n): The normalized Gaussian kernel value at offset (m,n).  
kh,kw: The half-dimensions of the kernel (height and width).

1. **Edge Detection**: A custom Sobel operator is applied to compute the image gradient in the x and y directions. The gradient magnitude is:  
   

A Binary edge map is created by using threshold:

  
where Mnormalized is



1. **Contour Detection**: Contours are identified in the binary image, and points belonging to each contour are extracted.
2. **Circle Fitting**: A least-squares optimization algorithm is used to fit a circle to the contour points. The circle parameters (cx,cy,r)(c\_x, c\_y, r) are iteratively refined to minimize the residual error. The residuals measure the difference between the distance of each point to the estimated circle center and the estimated radius.

**Outputs**

* **Detected Circles**: Each circle is represented as (cx,cy,r)(c\_x, c\_y, r), where cx,cyc\_x, c\_y are the circle's center coordinates, and r is its radius.
* **Canvas Visualization**: A dimmed version of the original image with detected circles overlaid and a grid for reference.

**2. Trajectory Calculation**

**Objective**

To compute the trajectory of a ball considering gravity and air resistance, enabling simulation of its motion from an initial position to a target.

**Mathematical Models**

1. **Ball Motion Ode**:
   * The ball's position (x,y)(x, y) and velocity (vx,vy)(v\_x, v\_y) evolve over time according to:

dvx/dt = - (k/m) \* vx \* sqrt(vx2 + vy2)  
dvy/dt = - g - (k/m) \* vy \* sqrt(vx2 + vy2)

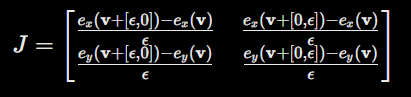
1. **Numerical Integration**: The Runge-Kutta 4th Order (RK4) method is employed to numerically integrate these differential equations. For each time step h=0.01:

y1 = y0 + (⅙) (k1 + 2k2 + 2k3 + k4) where  
k1 = hf(x0, y0); k2 = hf[x0 + (½)h, y0 + (½)k1]  
k3 = hf[x0 + (½)h, y0 + (½)k2]; k4 = hf(x0 + h, y0 + k3)

The Runge-Kutta 4th-order method (RK4) used in the code is not A-stable. It is conditionally stable, meaning that its stability depends on the step size h. If h is too large for a given problem, particularly for stiff equations, RK4 may become unstable. But as h=0.01, relatively small step size for this problem, RK4 can guarantee stability of the process.

1. **Shooting Method**: An iterative approach is used to compute the initial velocity v0 that results in the ball reaching a target (xt,yt)(x\_t, y\_t) within time T. Errors in position are minimized by solving:

Ex = Xfinal – Xtarget; Ey = yfinal – ytarget

The Jacobian matrix approximates the relationship between velocity changes and errors, enabling corrections:  


**Outputs**

* **Trajectories**: For each target ball, a trajectory is calculated and stored as a sequence of (x,y,vx,vy)(x, y, v\_x, v\_y) states.

**3. Animation**

**Objective**

To create a visual representation of the simulated trajectories and interactions between the balls.

**Process**

1. **Setup**: The detected circles are drawn on the initial canvas. A red ball is introduced at the bottom center of the canvas.
2. **Frame-by-Frame Update**: For each time step, the red ball's position is updated based on the computed trajectory. The footprint of the red ball's trajectory is drawn to visualize its path.
3. **Collision Detection**: If the red ball's distance from a target ball's center is less than the sum of their radius, a collision is detected, and the target ball is removed.
4. **Video Generation**: Frames are saved to create a video illustrating the motion and interactions.

**Outputs**

* **Animation**: A smooth animation showing the red ball’s motion and its collisions with other balls. which you can see in Task1/outputs folder. To run the program for personal tests one should run animation class and change value of image\_path variable.

**Comparison of numerical methods**

As an additional numerical experiments, code provides implementation of rk4, rk2 and Euler methods be used to solve Ball Motion Ode, for simplicity, implementations are presented in a different, NP\_experiments.py file. From the output:  
Solving with Euler method:  
EULER method: max error = 0.242665, mean error = 0.242665  
EULER method took 0.002268 seconds for calculations.

Solving with RK2 method:  
RK2 method: max error = 0.000204, mean error = 0.000204  
RK2 method took 0.004848 seconds for calculations.

Solving with RK4 method:  
RK4 method: max error = 0.000000, mean error = 0.000000  
RK4 method took 0.011410 seconds for calculations.  
  
we can conclude that Euler method is the fastest, but the error is too large to be used for such projects where accuracy is important, same can be said about rk2 method, which also has an noticeable error. RK4 can be the best in this situation. Despite it takes 0.01 seconds for a single calculation(method call), error is 0.