
Projectile Motion

Kopaliani Mate

Introduction

Edge detection Algorithm

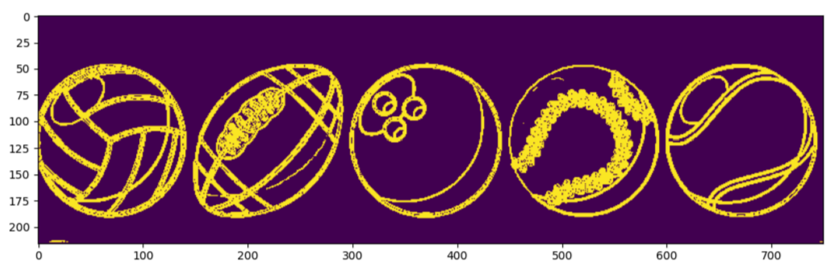
- Transform image to grayscale
- Perform Gaussian blurr
- Use Sobel Operator on the image and calculate Magnitude

$$S_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \quad S_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{1}$$

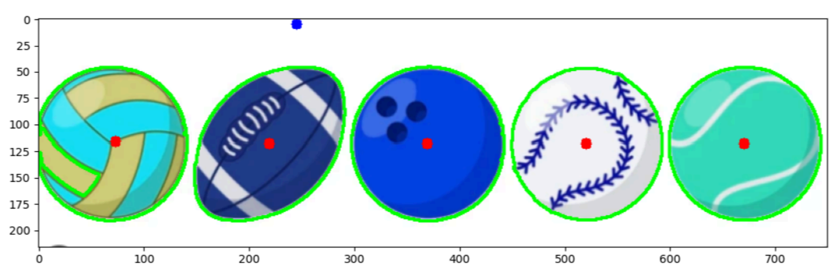
- calculate the Magnitude

$$\begin{gathered} h_x = I * S_x \quad h_y = I * S_y \\ M = \sqrt{h_x^2 + h_y^2} \end{gathered} \tag{2}$$

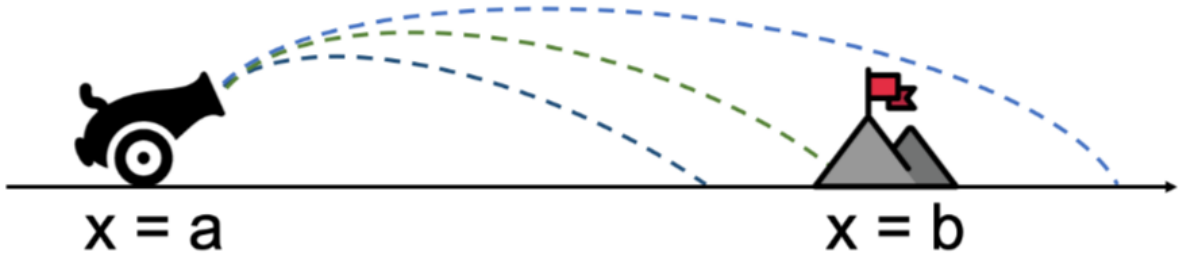
- do binary thresholding



- use DBSCAN algorithm to detect the balls and also the starting points where we can shoot from (in blue)



Shooting Method



Projectile Motion give as Second order ODE

$$\frac{d^2y}{dt^2} = -g \tag{3}$$

where

- $g = 9.81$
- $y(t)$ is the position of the ball

initial conditions $y(a) = a_0,\ y(b) = b_0$

which in our case is starting position of random point and the end position which is the center of the ball we want to hit.

Solution

we solve by shooting method. first transform the ODE into first order

$$\begin{aligned} \frac{dy}{dt} &= v \\ \frac{dv}{dt} &= -g \end{aligned} \tag{4}$$

written in vector form

$$\begin{aligned} S(t) &= \begin{bmatrix} y(t) \\ v(t) \end{bmatrix} \\ \frac{dS(t)}{dt} &= \begin{bmatrix} 0 & -1 \\ 0 & -g/v \end{bmatrix} S(t) \end{aligned} \tag{5}$$

```python
def F(t, s):
  return np.dot(np.array([[0, 1], [0, -9.8 / s[1]]]), s)
```

we use RK4, Backward Euler and Euler to find the optimal velocity in shooting method

- formulation of Backward Euler (***A*-stable** method)

$$S_{n+1} = S_n + hF(t_{n+1}, S_{n+1}) \tag{6}$$

since we have S_{n+1} on both sides we are calculating next iteration by minimizing the residual

$$R(S_{n+1}) = S_{n+1} - S_n - hF(t_{n+1}, S_{n+1}) \tag{7}$$

using newton method to achieve $R(S_{n+1}) = 0$ with jacobian

$$J(S_{n+1}) = I - h\frac{\partial F(t_{n+1}, S_{n+1})}{\partial S_{n+1}} \tag{8}$$

- formulation of RK4

$$S_{n+1} = S_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{9}$$

$$\begin{aligned} k_1 &= F(t_n, S_n) \\ k_2 &= F\left(t_n + \frac{h}{2}, S_n + \frac{h}{2}k_1\right) \\ k_3 &= F\left(t_n + \frac{h}{2}, S_n + \frac{h}{2}k_2\right) \\ k_4 &= F(t_n + h, S_n + hk_3) \end{aligned} \tag{10}$$

- formulation of Euler

$$S_{n+1} = S_n + hF(t_n, S_n) \tag{11}$$

where

- $F(t, S)$ is the system of equations defined as $F(t, s) = \frac{dS}{dt}$
- h is a step size
- t_n is the current time
- S_n is the state vector at time t_n

analysis and comparison of methods

- Backward Euler

```
target coordinates: 520, 118
last trajectory point: 520.0, 117.99999999981083
target coordinates: 670, 118
last trajectory point: 670.0, 117.9999999996071
target coordinates: 219, 118
last trajectory point: 219.0, 117.9999999999709
target coordinates: 73, 116
last trajectory point: 73.0, 115.99999999999659
target coordinates: 369, 118
last trajectory point: 369.0, 117.99999999991633
```

- Rk4 output

```
target coordinates: 520, 118
last trajectory point: 520.0, 117.99999999972351
target coordinates: 670, 118
last trajectory point: 670.0, 117.99999999722058
target coordinates: 219, 118
last trajectory point: 219.0, 118.00000000000273
target coordinates: 73, 116
last trajectory point: 73.0, 116.00000000000102
target coordinates: 369, 118
last trajectory point: 369.0, 117.99999999993452
```

- Euler output

```
target coordinates: 520, 118
last trajectory point: 520.0, 117.99999999974534
target coordinates: 670, 118
last trajectory point: 670.0, 117.99999999959255
target coordinates: 219, 118
last trajectory point: 219.0, 118.00000000000273
target coordinates: 73, 116
last trajectory point: 73.0, 115.99999999999864
target coordinates: 369, 118
last trajectory point: 369.0, 118.0000000001346
```

- Rk4 method achieves high accuracy, with errors on the order of 10^{-10} to 10^{-12}.
  - but the errors result in less consistent form compared to backward euler.
- Euler method also reaches target coordinates with error on the order of 10^{-10} to 10^{-12} aswell.
  - Regular Euler also has non consistent error.
- Backward Euler
  - The errors are consistently small across all target coordinates (i.e 115.99999999999864 *vs* 116.00000000000102)

References

heplful article for ODE
https://pythonnumericalmethods.studentorg.berkeley.edu/notebooks/chapter23.02-The-Shooting-Method.html