**Documentation for the “Intercept a Moving Ball” Project**

**Overview**

This project implements a complete workflow for detecting, analyzing, and animating the motion of balls in a 2D environment. The system has three main components. Ball detection of a thrown ball and trajectory reconstruction using BallDetector, Simulation of a ball throw to reach a specified target point and hit previously detected ball using BallThrower and Visualization and animation of the observed and simulated trajectories using Animator.

**1. Ball Detection (BallDetector)**

**Objective**

Extracts the trajectory of a ball from a video by detecting the ball's position frame-by-frame and Estimates the ball's initial velocity and drag coefficient (k/m) using optimization methods.

**Key Methods**

1. **\_process\_video()**: Processes each video frame, detects the ball using contour analysis, and records its trajectory in meters. Ratio of meters:pixels is taken as 1:100.
2. **detect\_ball()**: Uses frame differencing and thresholding to identify the largest contour (assumed to be the ball). As the first task includes manually written detection algorithms, this one uses cv2 functions for it, including gaussian blur, threshold and findContours.
3. **gradient\_descent\_initial\_velocity()**: Computes the ball's initial velocity vxv\_x and vyv\_y by minimizing the difference between observed and simulated trajectories, uses formula:   
   vx(k+1)​=vx(k)​−η(​∂E/∂vx ​; vy(k+1)​=vy(k)​−η​(∂E/∂vy) ​  
   takes initial velocity as (0,0) and dt=0.01, with learning rate of 0.01 and maximum iterations of 1000 and calculates error using formula: E=∑((xobs​−xsim​)2+(yobs​−ysim​)2)
4. **shooting\_method()**: Refines the drag coefficient (k/m) by aligning the final point of the simulated trajectory with the observed one. Initial guess is 0.1 with step size of 0.05 and error threshold of 0.001, error is calculated using the last point of the estimated and the original trajectories.
5. **estimate\_trajectory()**: Simulates the ball's trajectory over time using the calculated initial conditions and physical models. Solves trajectory’s ODE(ball motion) using ode solver, which on it’s side uses manual rk4 implementation.

**Mathematical Models**

1. **Ball Motion ODE**: Governs the position (x,y)(x, y) and velocity (vx,vy)(v\_x, v\_y):

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

1. **Runge-Kutta 4th Order (RK4)**: Numerical integration to solve the differential equations for motion:

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.

**Outputs**

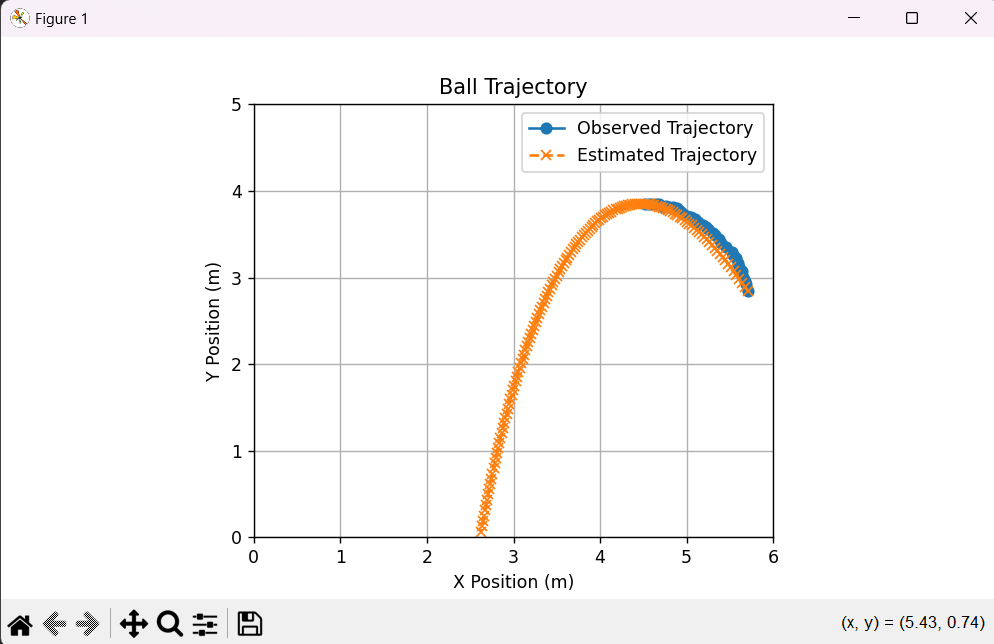
1. **Observed Trajectory**: A sequence of (t,x,y)(t, x, y) points.
2. **Estimated Trajectory**: A refined sequence of trajectory points matching the observed motion.
3. **Parameters**:
   * Example 1

Estimated k/m=0.18300996477505654

Initial Velocity: vx = -3.41 m/s, vy = 5.07 m/s

Drag Coefficient (k/m): 0.1830

Ball Radius: 0.2831 m



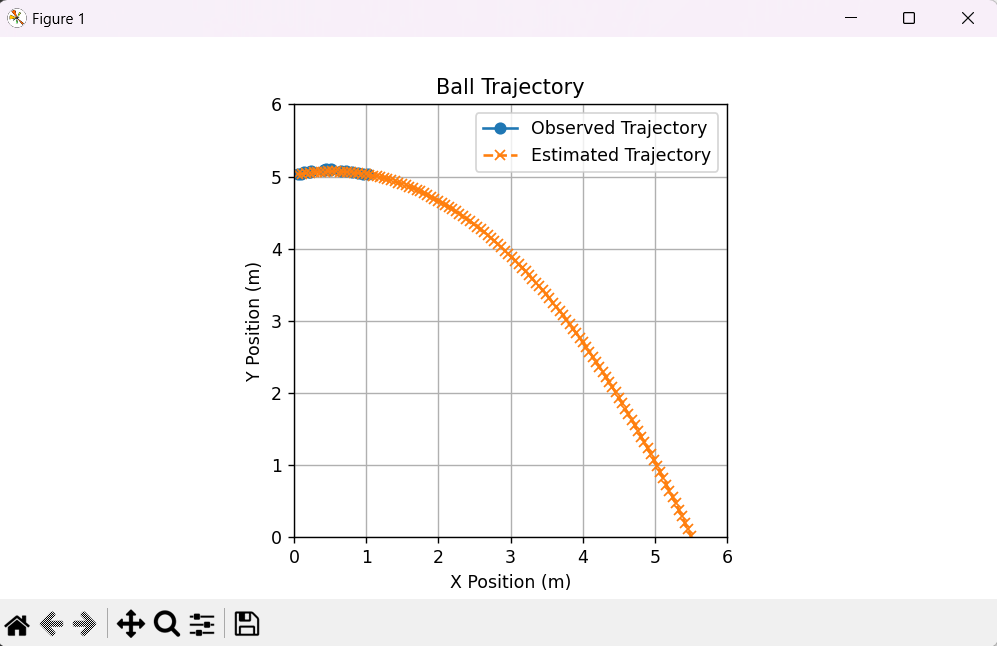
* + Example 2

Estimated k/m=0.027051696461635643

Initial Velocity: vx = 5.23 m/s, vy = 0.90 m/s

Drag Coefficient (k/m): 0.0271

Ball Radius: 0.3289 m



**2. Ball Throw Simulation (BallThrower)**

**Objective**

Simulates a ball throw designed to hit a specific target point extracted from the observed trajectory.

**Key Methods**

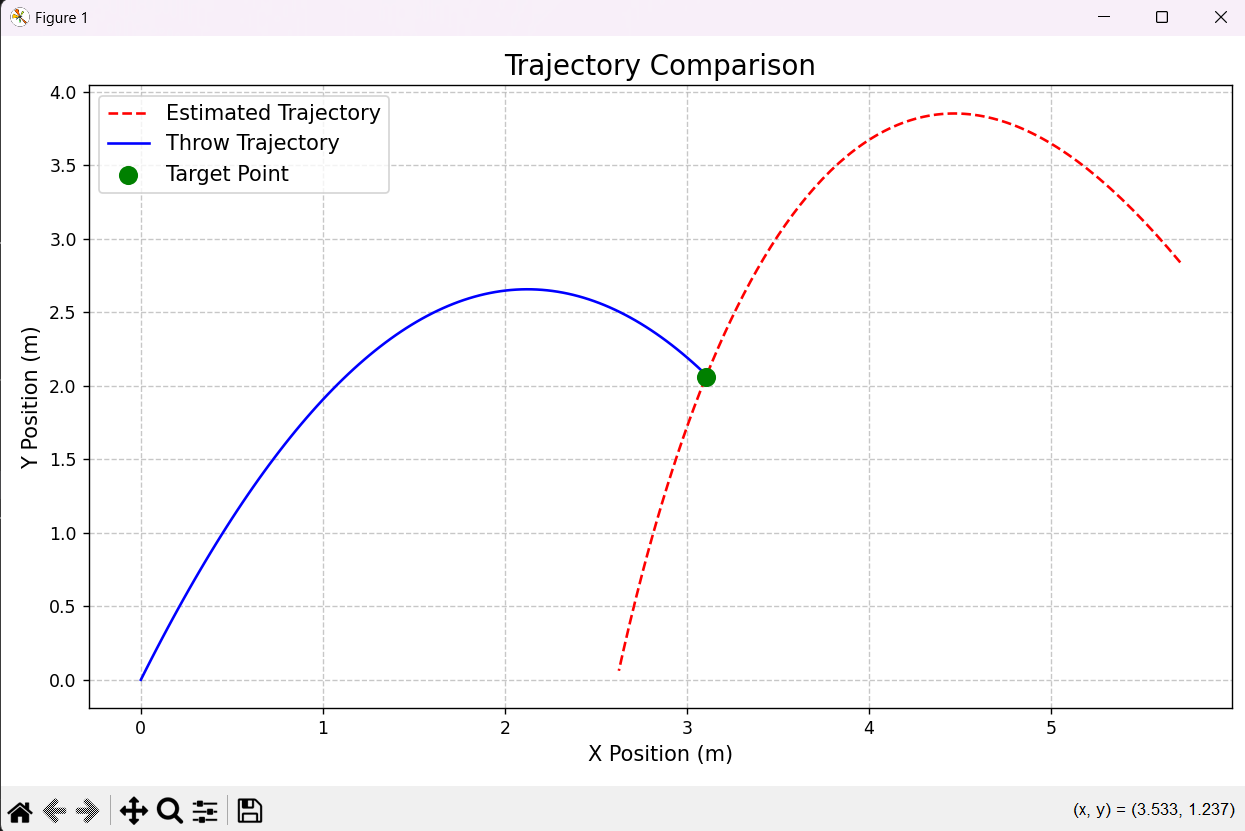
1. **\_select\_target\_point()**: Chooses a random target point from the second half of the observed trajectory. (second half is chosen for better visualization of correctly chosen trajectory, as if the dt is too short, ball is thrown with enormous speed and trajectory seems linear at a glance)
2. **\_calculate\_velocity()**: Iteratively determines the initial velocity (vx,vy)(v\_x, v\_y) needed to reach the target point by minimizing errors. dt = 0.01 and k/m is taken as 0.01 as an example. Error calculation is: Ex = xtarget - xend; Ey = ytarget - yend. Velocities update during 100 iterations with error threshold at 0.01 with: vx += 0.1 \* Ex / ttarget; vy += 0.1 \* Ey / ttarget
3. **\_estimate\_trajectory()**: Computes the simulated trajectory of the thrown ball using RK4 integration to solve trajectory ODE(ball motion)

**Outputs**

1. **Simulated Throw Trajectory**: A sequence of (t,x,y)(t, x, y) points for the ball's motion toward the target point.
2. **Initial Velocity**: Components vxv\_x and vyv\_y of the thrown ball.
3. **Target Point**: The chosen goal coordinates (x,y)(x, y) to be hit by the ball.
   1. Example 1:

Target Point: (1.0800000000000007, 3.104110757167739, 2.0603780941512024))

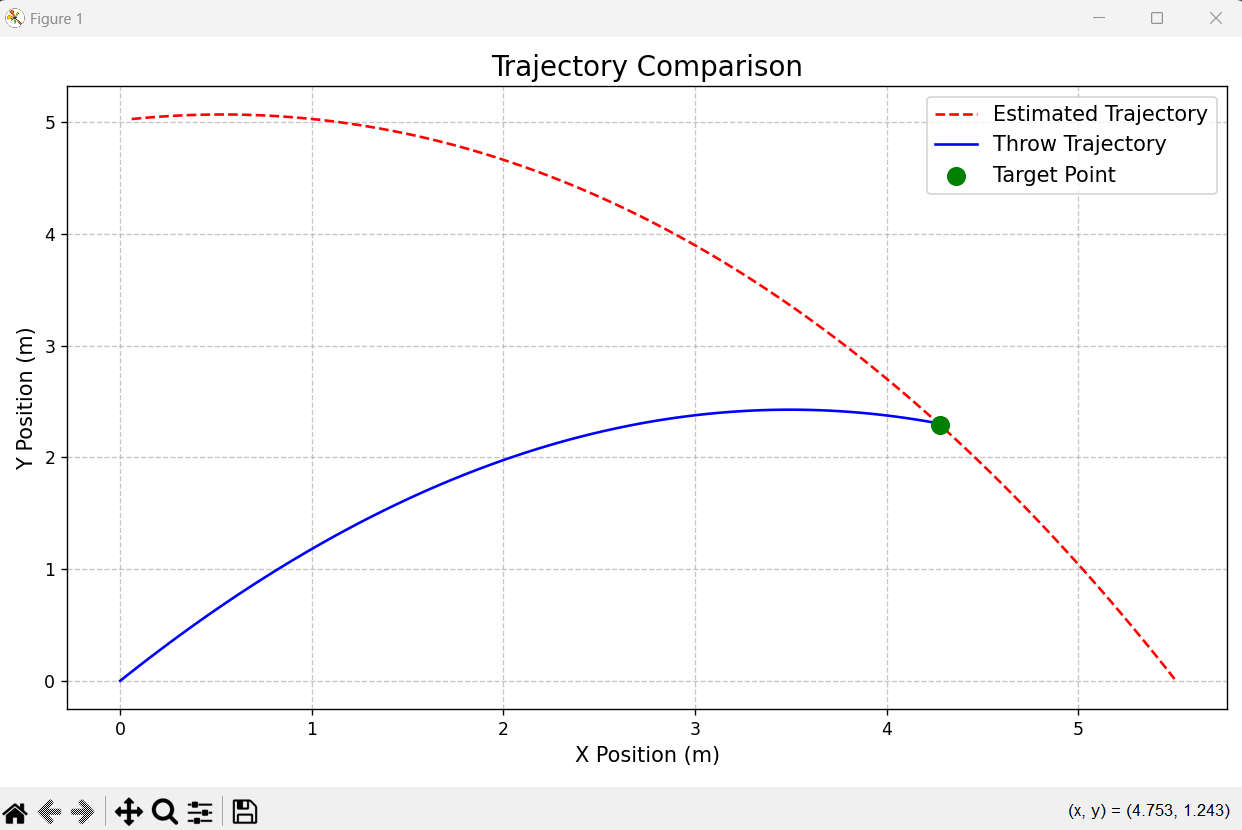
Initial Velocity: vx = 2.96, vy = 7.33



* 1. Example 2:

Target Point: (0.8600000000000005, 4.276544362129806, 2.2943214058925854)

Initial Velocity: vx = 5.13, vy = 7.02



**3. Animation (Animator)**

**Functionality**

Visualizes the observed and simulated trajectories of the balls and compares them in an animated plot.

**Key Methods**

1. **init\_animation()**: Prepares the figure by setting axis limits, gridlines, and adding the trajectories and target point.
2. **update(frame)**: Updates the positions of the observed and thrown balls for the current frame in the animation.
3. **animate()**: Generates a frame-by-frame animation comparing the observed and simulated motions.

**Outputs**

* **Animation**: A dynamic visualization showing the trajectories of the observed and simulated balls and their alignment with the target point. Animations are stored in Task2/outputs folder. To run code for personal tests one should run animator class and change value of video\_path variable.