Task2 Documentation

Giorgi Kezevadze

January 2025

1 Problem formulation

Given a cropped video of a moving ball, the task was to predict the motion of the ball on the cropped part of the frames to intercept it by throwing another ball at time t of the cropped part of the video.

2 Algorithm

2.1 Extract trajectory of a moving ball

1. Detect edges on each frame of the image with canny operator.
2. Cluster the edges using DBSCAN and assign a centroid to obtain the center of the moving ball, representing its position on each frame.
3. Evaluate the positions on each frame to determine the overall trajectory of the ball.

2.2 Predict the cropped frames

1. Extract the last two positions from the calculated trajectory and compute the velocity on the last frame of the cropped video as the initial velocity v_0 on the next frames, using the finite difference method.

$$v_0 = \frac{x_n - x_{n-1}}{\Delta t}$$

2. Use RK4 on the ball motion model to restore the missing positions with the initial velocity v_0 and the boundary position values. The number of frames to restore is optional and set by parameter n in the code.
3. Choose a random frame t in the restored part of the video and extract the position on that frame from the trajectory as the target position of the thrown ball.

2.3 Intercept the moving ball

1. Choose a random point on the video frame.
2. Apply the shooting method to the ball motion ODE, with the initial position being the randomly chosen point.
3. Set the tolerance for convergence as tolerance = radius of the thrown ball + radius of the target ball, if $error < tolerance$, the two balls collide, meaning the trajectory trail converged.
4. Use the RK4 method to solve the ODE with the guessed initial velocity.
5. Apply Newton's method for adjusting the initial velocity guess. Save the converged trajectory in the position array for each ball.

2.4 Animate

Finally, animate the resulting trajectories by looping over their arrays and displaying them frame by frame to create an animation. This visualization helps evaluate the error in computing the missing positions.

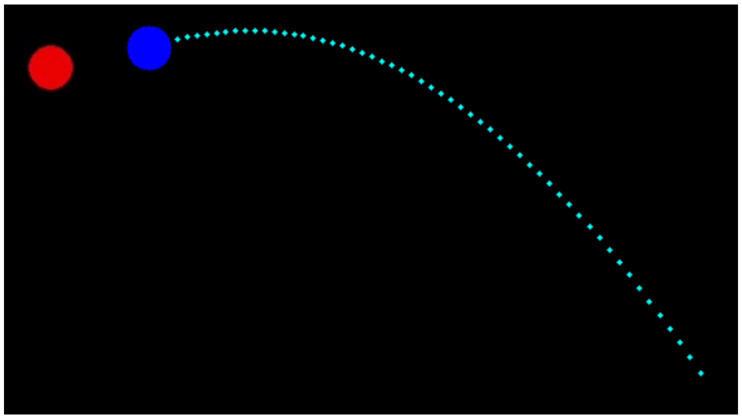

Figure 1: Original trajectory

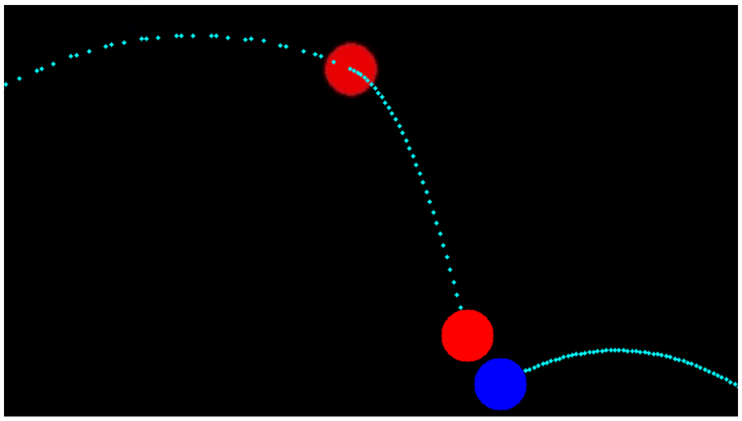

Figure 2: Restored trajectory

3 Numerical methods

3.1 Ball motion ODE

As the ODE model, I used the ball motion ODE for the thrown ball as it was compulsory for this task. I chose $k/m = 0.0001$ and $g = -9.8(pixel/frame^2)$.

$$\frac{dx}{dt} = v_x, \quad \frac{dy}{dt} = v_y$$

$$\frac{dv_x}{dt} = -\frac{k}{m} v_x \sqrt{v_x^2 + v_y^2}$$

$$\frac{dv_y}{dt} = -g - \frac{k}{m} v_y \sqrt{v_x^2 + v_y^2}$$

- t: Frame number
- $x(t)$: Horizontal position of the ball at frame t
- $y(t)$: Vertical position of the ball at frame t
- $v_x(t)$: Horizontal velocity of the ball at frame t
- $v_y(t)$: Vertical velocity of the ball at frame t
- g: Gravity $(pixel/frame^2)$
- k: The drag coefficient
- m: The mass of the ball

3.2 Runge-Kutta 4th Order (RK4) Method

The Runge-Kutta method estimates the velocity and acceleration at four points:

$$\mathbf{k}_1^v = \mathbf{a}(\mathbf{v}_n), \quad \mathbf{k}_1^r = \mathbf{v}_n$$

$$\mathbf{k}_2^v = \mathbf{a}(\mathbf{v}_n + \frac{1}{2}\mathbf{k}_1^v \Delta t), \quad \mathbf{k}_2^r = \mathbf{v}_n + \frac{1}{2}\mathbf{k}_1^v \Delta t$$

$$\mathbf{k}_3^v = \mathbf{a}(\mathbf{v}_n + \frac{1}{2}\mathbf{k}_2^v \Delta t), \quad \mathbf{k}_3^r = \mathbf{v}_n + \frac{1}{2}\mathbf{k}_2^v \Delta t$$

$$\mathbf{k}_4^v = \mathbf{a}(\mathbf{v}_n + \mathbf{k}_3^v \Delta t), \quad \mathbf{k}_4^r = \mathbf{v}_n + \mathbf{k}_3^v \Delta t$$

The final updates for velocity and position are:

$$\mathbf{v}_{n+1} = \mathbf{v}_n + \frac{1}{6}(\mathbf{k}_1^v + 2\mathbf{k}_2^v + 2\mathbf{k}_3^v + \mathbf{k}_4^v)\Delta t$$

$$\mathbf{r}_{n+1} = \mathbf{r}_n + \frac{1}{6}(\mathbf{k}_1^r + 2\mathbf{k}_2^r + 2\mathbf{k}_3^r + \mathbf{k}_4^r)\Delta t$$

3.3 Newton's method for adjustment

As the adjustment method I used Newton's method over bisection for convenance. I thought adjusting the delta variable instead of upper and lower limits of velocity was much easier. To refine the velocity $\mathbf{v} = (v_x, v_y)$ using Newton's method, we compute the Jacobian matrix J via finite differences:

$$J_{:,i} = \frac{(\mathbf{r}^* - \mathbf{r}_{perturbed}) - (\mathbf{r}^* - \mathbf{r})}{\Delta}$$

where: - $\mathbf{r}^*$ is the target position, - $\mathbf{r}$ is the final position computed with the current velocity, - $\mathbf{r}_{perturbed}$ is the final position computed with a perturbed velocity component $v_i + \Delta$.

The velocity update $\Delta\mathbf{v}$ is then obtained by solving:

$$J\Delta\mathbf{v} = \mathbf{r}^* - \mathbf{r}$$

If J is singular, we set $\Delta\mathbf{v} = \mathbf{0}$. Otherwise, we compute:

$$\mathbf{v} \leftarrow \mathbf{v} + \Delta\mathbf{v}$$