Illuminated Drone Show Simulation

Numerical Programming Final Project

NP 2025

Ramaz Botchorishvili
Kutaisi International University
r.botchorishvili@kiu.edu.ge

January 2026

Abstract

This report presents a comprehensive simulation of an illuminated drone light show using numerical methods. The project models drone swarm dynamics as an Initial Value Problem (IVP) consisting of coupled ordinary differential equations governing position and velocity. The system incorporates spring-damper dynamics for target tracking, velocity saturation for realistic motion constraints, and repulsive forces for collision avoidance. The IVP is solved using the 4th-order Runge-Kutta (RK4) method, providing $O(h^4)$ global accuracy. The simulation demonstrates three distinct phases: static-to-static transitions forming handwritten text, image-based shape formation, and dynamic tracking of animated GIF frames. All core numerical algorithms are implemented from first principles using only NumPy for array operations, demonstrating mastery of numerical methods including spatial hashing for efficient neighbor queries, greedy assignment for target matching, and complete image processing pipelines for edge detection.

Contents

1 Introduction

1.1 Background

Illuminated drone shows have emerged as a spectacular form of entertainment and artistic expression, replacing traditional fireworks displays at major events worldwide. Companies like Intel and EHang have demonstrated swarms of thousands of synchronized drones forming complex three-dimensional shapes in the night sky. These shows require precise coordination algorithms to ensure smooth transitions between formations while maintaining safe separation distances between aircraft.

The mathematical foundation of drone swarm coordination lies in numerical methods for solving differential equations, optimization algorithms for path planning, and real-time control systems. This project applies concepts from numerical programming to simulate such a drone light show in a 2D environment.

1.2 Project Goals

The primary objectives of this project are:

1. **Model drone dynamics** as an Initial Value Problem using coupled ODEs for position and velocity
2. **Implement RK4 integration** from scratch to solve the IVP with high accuracy
3. **Design collision avoidance** using repulsive force fields between nearby drones
4. **Extract target formations** from images using edge detection algorithms
5. **Demonstrate three operational modes**: static-to-static, static-to-image, and dynamic tracking

1.3 Three Subtasks

The simulation consists of three distinct phases (subtasks):

Subtask 1: Transition from initial grid formation to a handwritten name

Subtask 2: Transition from the name to a greeting image ("Happy New Year")

Subtask 3: Dynamic tracking of a moving target extracted from an animated GIF

2 Mathematical Model

2.1 Initial Value Problem Formulation

The motion of each drone is governed by a second-order ODE system, reformulated as a first-order IVP. Let $\mathbf{x}_i(t) \in \mathbb{R}^2$ denote the position and $\mathbf{v}_i(t) \in \mathbb{R}^2$ the velocity of drone i at time t.

2.1.1 Velocity Equation with Saturation

The position evolves according to velocity with a maximum speed constraint:

$$\frac{d\mathbf{x}_i}{dt} = \mathbf{v}_i \cdot \min\left(1, \frac{v_{\max}}{|\mathbf{v}_i|}\right) \tag{1}$$

This ensures that regardless of the computed velocity, the actual displacement never exceeds $v_{\max}$ per unit time.

2.1.2 Acceleration Equation (Spring-Damper with Repulsion)

The velocity evolves according to a combination of forces:

$$\frac{d\mathbf{v}_i}{dt} = \frac{1}{m}\left[k_p(\mathbf{T}_i - \mathbf{x}_i) + \mathbf{f}_{rep,i} - k_d\mathbf{v}_i\right] \tag{2}$$

where:

- m is the drone mass
- $k_p(\mathbf{T}_i - \mathbf{x}_i)$ is the spring force pulling toward target $\mathbf{T}_i$
- $k_d\mathbf{v}_i$ is the damping force (viscous friction)
- $\mathbf{f}_{rep,i}$ is the repulsive force from nearby drones

2.2 Repulsive Force Model

To prevent collisions, drones within a safety radius R_{safe} exert repulsive forces on each other:

$$\mathbf{f}_{rep,i} = \sum_{j \neq i, |\mathbf{x}_i - \mathbf{x}_j| < R_{safe}} k_{rep} \cdot \frac{\mathbf{x}_i - \mathbf{x}_j}{|\mathbf{x}_i - \mathbf{x}_j|^3} \tag{3}$$

This inverse-square law creates strong repulsion at close range that diminishes rapidly with distance, similar to electrostatic repulsion between like charges.

2.3 Model Parameters

Table 1 summarizes the physical parameters used in the simulation.

Table 1: Simulation Parameters

Symbol	Parameter	Value	Unit
m	Drone mass	1.0	kg
k_p	Spring constant (position gain)	25.0	N/m
k_d	Damping coefficient	12.0	N·s/m
k_{rep}	Repulsion strength	50.0	N·m^2
R_{safe}	Safety radius	4.0	m
$v_{\max}$	Maximum velocity	100.0	m/s
Δt	Time step	0.05	s
n	Number of drones	1200	–

3 Numerical Methods

3.1 4th-Order Runge-Kutta Integration

The IVP system (Equations 1 and 2) is solved using the classical 4th-order Runge-Kutta method (RK4). For a general ODE $\frac{dy}{dt} = f(t, y)$, RK4 computes:

$$k_1 = f(t_n, y_n) \tag{4}$$

$$k_2 = f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_1\right) \tag{5}$$

$$k_3 = f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_2\right) \tag{6}$$

$$k_4 = f(t_n + h, y_n + hk_3) \tag{7}$$

$$y_{n+1} = y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{8}$$

3.1.1 Error Analysis

RK4 has a local truncation error of $O(h^5)$ and a global error of $O(h^4)$. For our time step $\Delta t = 0.05$s, this provides excellent accuracy while maintaining computational efficiency for 1200 drones simulated over thousands of frames.

3.1.2 Implementation

The coupled system is integrated simultaneously:

Algorithm 1 RK4 Integration Step

function STEP($\mathbf{x}, \mathbf{v}, \mathbf{T}$)
  $k_{1x}, k_{1v} \leftarrow \text{deriv}(\mathbf{x}, \mathbf{v})$
  $k_{2x}, k_{2v} \leftarrow \text{deriv}(\mathbf{x} + \frac{\Delta t}{2}k_{1x}, \mathbf{v} + \frac{\Delta t}{2}k_{1v})$
  $k_{3x}, k_{3v} \leftarrow \text{deriv}(\mathbf{x} + \frac{\Delta t}{2}k_{2x}, \mathbf{v} + \frac{\Delta t}{2}k_{2v})$
  $k_{4x}, k_{4v} \leftarrow \text{deriv}(\mathbf{x} + \Delta t \cdot k_{3x}, \mathbf{v} + \Delta t \cdot k_{3v})$
  $\mathbf{x} \leftarrow \mathbf{x} + \frac{\Delta t}{6}(k_{1x} + 2k_{2x} + 2k_{3x} + k_{4x})$
  $\mathbf{v} \leftarrow \mathbf{v} + \frac{\Delta t}{6}(k_{1v} + 2k_{2v} + 2k_{3v} + k_{4v})$
  return $\mathbf{x}, \mathbf{v}$
end function

3.2 Spatial Hashing for Collision Detection

Naive collision detection requires $O(n^2)$ distance calculations. We implement spatial hashing to reduce this to $O(n)$ average case.

3.2.1 Hash Function

The 2D space is discretized into grid cells of size $c = R_{safe}$:

$$h(x, y) = \left(\left\lfloor \frac{x}{c} \right\rfloor, \left\lfloor \frac{y}{c} \right\rfloor\right) \tag{9}$$

3.2.2 Neighbor Query Algorithm

For each drone, we only check the 3×3 neighborhood of cells:

Algorithm 2 Find Neighbor Pairs using Spatial Hashing

```
function FindNeighborPairs(positions, radius)
    cells ← BuildSpatialHashGrid(positions, radius)
    pairs ← [ ]
    for each drone i do
        (c_x, c_y) ← cell of drone i
        for dx, dy ∈ {−1, 0, 1} do
            for each drone j in cell (c_x + dx, c_y + dy) do
                if j > i and |x_i − x_j| < radius then
                    pairs.append(i, j)
                end if
            end for
        end for
    end for
    return pairs
end function
```

3.3 Greedy Assignment Algorithm

To assign drones to target positions, we use a greedy algorithm:

$$\text{For each drone } i: \quad j^* = \arg\min_{j \text{ unassigned}} \|\mathbf{x}_i - \mathbf{T}_j\| \tag{10}$$

This provides $O(n^2)$ complexity compared to $O(n^3)$ for the optimal Hungarian algorithm, with acceptable results for our application.

4 Image Processing

Target formations are extracted from input images using edge detection. All algorithms are implemented from mathematical first principles.

4.1 Gaussian Blur

Noise reduction is performed using 2D Gaussian convolution:

$$G(x, y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right) \tag{11}$$

The convolution operation:

$$(f * G)(x, y) = \sum_i \sum_j f(i, j) \cdot G(x - i, y - j) \tag{12}$$

4.2 Sobel Edge Detection

Gradient computation uses Sobel operators:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \tag{13}$$

Gradient magnitude and direction:

$$|\nabla I| = \sqrt{G_x^2 + G_y^2}, \quad \theta = \arctan\left(\frac{G_y}{G_x}\right) \tag{14}$$

4.3 Canny Edge Detection

The complete Canny algorithm consists of four stages:

1. **Gaussian Blur**: Reduce noise using Equation 11
2. **Gradient Computation**: Apply Sobel operators
3. **Non-Maximum Suppression**: Thin edges by keeping only local maxima in gradient direction
4. **Hysteresis Thresholding**: Connect edges using dual thresholds (low=50, high=150)

5 Implementation

5.1 Software Architecture

The simulation is implemented in Python 3.12 with the following structure:

- `drone_show_math.py`: Pure mathematical implementation (692 lines)
- `drone_show.py`: Optimized version using scipy for comparison
- `MATH_IMPLEMENTATION_NOTES.md`: Documentation of algorithms

5.2 Dependencies

- **NumPy**: Array operations and vectorized mathematics
- **OpenCV**: Image/video I/O only (no algorithmic functions)
- **Pillow**: GIF frame extraction

5.3 Three Subtasks Implementation

5.3.1 Subtask 1: Initial → Handwritten Name

Drones start in a uniform grid and transition to positions forming a handwritten name. Target points are extracted from `handwritten_name.png` using edge detection.

5.3.2 Subtask 2: Name → Greeting Image

From the name formation, drones smoothly transition to form the greeting "Happy New Year" extracted from `greeting.png`.

5.3.3 Subtask 3: Dynamic GIF Tracking

Drones track a moving target extracted from an animated GIF (41 frames). The assignment is updated every 6 physics steps to balance responsiveness and smoothness.

6 Results

6.1 Output Summary

The simulation produces the following outputs:

Table 2: Simulation Output Statistics

Phase	Frames	Duration (s)	Description
Phase 1	110	3.7	Grid → Handwritten Name
Phase 2	110	3.7	Name → Greeting
Phase 3	246	8.2	Dynamic GIF Tracking
Total	**466**	**15.5**	Combined simulation

6.2 Video Outputs

Eight video files are generated:

- `drone_show_math_phase1.mp4`: Subtask 1 only
- `drone_show_math_phase2.mp4`: Subtask 2 only
- `drone_show_math_phase3.mp4`: Subtask 3 only
- `drone_show_math_combined.mp4`: All phases combined
- (Plus 4 equivalent files from the optimized implementation)

6.3 Performance

- **Drones**: 1200 simultaneous particles
- **Frame rate**: 30 FPS output
- **Runtime**: Approximately 2 minutes for full simulation
- **Collision-free**: Zero collisions observed (repulsive forces effective)

7 Conclusion

This project successfully demonstrates the application of numerical methods to simulate an illuminated drone light show. Key achievements include:

1. **IVP Formulation**: Modeled drone dynamics as coupled ODEs with spring-damper physics and velocity saturation
2. **RK4 Implementation**: Solved the IVP using 4th-order Runge-Kutta integration implemented from first principles, achieving $O(h^4)$ accuracy
3. **Collision Avoidance**: Implemented inverse-square repulsive forces with spatial hashing for $O(n)$ neighbor queries
4. **Image Processing**: Built complete edge detection pipeline including Gaussian blur, Sobel gradients, and Canny algorithm
5. **Pure Mathematical Implementation**: All core algorithms implemented using only NumPy array operations, demonstrating understanding of the underlying mathematics
6. **Three Operational Modes**: Successfully demonstrated static-to-static transitions and dynamic target tracking

The simulation produces smooth, collision-free transitions between formations with 1200 drones over a 15-second video, validating the effectiveness of the chosen numerical methods and physical model.

7.1 Future Work

Potential extensions include:

- 3D extension with perspective projection
- Optimal assignment using the Hungarian algorithm
- Real-time interactive control
- Hardware deployment on actual drone swarms

References

1. Burden, R.L., Faires, J.D. (2015). *Numerical Analysis*, 10th Edition. Cengage Learning.
2. Canny, J. (1986). A Computational Approach to Edge Detection. *IEEE TPAMI*, 8(6), 679-698.
3. Reynolds, C.W. (1987). Flocks, Herds, and Schools: A Distributed Behavioral Model. *SIGGRAPH '87*.

A Source Code Repository

The complete source code is available at:
https://github.com/devichikovani/np-final-project