Illuminated Drone Swarm Simulation

Numerical Programming Final Project

Andria Beridze

January 25, 2026

1 Project Abstract

This project implements a numerical simulation of a synchronized drone swarm consisting of $N = 1000$ quadcopters. The system is designed to perform autonomous formation control, trajectory transitions, and dynamic target tracking. The core of the simulation is a **Physics Engine** based on Second-Order Ordinary Differential Equations (ODEs), solved using the **Runge–Kutta 4 (RK4)** numerical integration method.

The simulation addresses three specific autonomous tasks:

1. **Static Formation:** Aggregating scattered drones into a predefined shape (e.g., a handwritten name).
2. **Transition Flight:** Smoothly reconfiguring the swarm from the first shape to a second shape ("Happy New Year!").
3. **Dynamic Tracking:** Real-time tracking of a moving target extracted from a video feed while preserving swarm cohesion.

2 System Pipeline

The data flow follows a strict *Input* $\rightarrow$ *Process* $\rightarrow$ *Output* pipeline:

2.1 Stage A: Input & Preprocessing

- **Input:** Raw data is ingested from either static images (grayscale JPEGs) or video files (MP4).
- **Processing:**
  - *Image Processing:* Adaptive thresholding is applied to separate the signal (ink or object) from the background.
  - *Point Extraction:* The system identifies (x, y) coordinates of relevant pixels.
  - *Downsampling:* A random sampling technique reduces the extracted points to exactly N coordinates to match the drone count.
- **Output:** A target matrix $T \in \mathbb{R}^{N\times 3}$, representing the desired destination for every drone.

2.2 Stage B: Physics Engine (The "Brain")

- **Input:** Current state (position x, velocity v) and target state T.
- **Processing:** The system calculates the net force acting on every drone based on three physical rules: **Attraction** (move to target), **Damping** (stabilize motion), and **Repulsion** (avoid collisions).
- **Output:** Acceleration vectors a for every drone.

2.3 Stage C: Numerical Solver (The "Legs")

- **Input:** Current state (x, v) and computed accelerations a.
- **Processing:** The **Runge–Kutta 4 (RK4)** integrator estimates the state of the system at time $t + \Delta t$. This method samples derivatives at four points within the timestep to minimize truncation error.
- **Output:** Updated state $(x_{\text{new}}, v_{\text{new}})$.

3 Mathematical Model

The simulation models the drones as point masses moving in 3D space.

3.1 State Definition

For a swarm of N drones, the state of the i-th drone at time t is defined by its position $x_i(t)$ and velocity $v_i(t)$:

$$x_i(t) \in \mathbb{R}^3, \quad v_i(t) \in \mathbb{R}^3, \quad i = 1, \ldots, N \tag{1}$$

3.2 Governing Equations

The motion is governed by Newton's Second Law ($F = ma$), formulated as a second-order ODE with velocity saturation:

$$\begin{cases} \dot{x}_i(t) = v_i(t) \, \min\left(1, \dfrac{v_{\max}}{\|v_i(t)\|}\right) \\ \dot{v}_i(t) = \dfrac{1}{m} \left(F_{\text{att}} + F_{\text{rep}} + F_{\text{damp}}\right) \end{cases} \tag{2}$$

where $v_{\max}$ is the maximum allowable velocity magnitude and m is the mass of a drone.

3.3 Force Components

The total force acting on drone i is defined as:

$$F_{\text{total}} = F_{\text{att}} + F_{\text{rep}} + F_{\text{damp}}$$

3.3.1 Attraction Force (F_{att})

A proportional controller that pulls the drone toward its assigned target $T_i(t)$:

$$F_{\text{att}} = k_p \left(T_i(t) - x_i(t)\right) \tag{3}$$

3.3.2 Damping Force (F_{damp})

A viscous friction term that stabilizes motion:

$$F_{\text{damp}} = -k_d \, v_i(t) \tag{4}$$

3.3.3 Repulsion Force (F_{rep})

A collision-avoidance mechanism that activates when the inter-drone distance $d_{ij} = \|x_i - x_j\|$ is smaller than the safety radius R_{safe}:

$$F_{\text{rep}} = \sum_{j \neq i} f_{ij} \tag{5}$$

with

$$f_{ij} = \begin{cases} k_{\text{rep}} \dfrac{x_i - x_j}{\|x_i - x_j\|^3}, & \text{if } \|x_i - x_j\| < R_{\text{safe}} \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

4 Numerical Methods

To simulate the continuous dynamics in discrete time, the system uses the **Runge–Kutta 4 (RK4)** integration method, chosen for its superior stability and fourth-order accuracy.

Given the state vector $Y = [x, v]$ and the differential equation $\dfrac{dY}{dt} = f(t, Y)$, the RK4 update rule is:

$$k_1 = f(t, Y_n) \tag{7}$$

$$k_2 = f\left(t + \frac{\Delta t}{2}, Y_n + \frac{\Delta t}{2} k_1\right) \tag{8}$$

$$k_3 = f\left(t + \frac{\Delta t}{2}, Y_n + \frac{\Delta t}{2} k_2\right) \tag{9}$$

$$k_4 = f(t + \Delta t, Y_n + \Delta t k_3) \tag{10}$$

$$Y_{n+1} = Y_n + \frac{\Delta t}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{11}$$

In our code, the function `rk4_step` implements this logic to update both positions and velocities simultaneously.

5 Implementation Details

This section documents the specific software implementation of the mathematical models described above. The codebase is organized into modular components within the `src/` directory.

5.1 Configuration Parameters (`src/config.py`)

The simulation behavior is controlled by a set of global constants. These parameters tune the physics engine and stability of the swarm.

5.2 Preprocessing Algorithms (`src/preprocessing.py`)

The `get_target_points` function converts a raw raster image (JPEG) into a set of 3D coordinates.

1. **Adaptive Thresholding:** The input image is converted to binary using Gaussian adaptive thresholding. This handles varying lighting conditions better than global thresholding.
2. **Morphological Opening:** A noise reduction step using an erosion followed by dilation to remove isolated noise pixels (speckles).
3. **Coordinate Extraction:** The indices (i, j) of all non-zero pixels are extracted.

Variable	Value	Description
N_DRONES	1000	Total number of agents in the swarm.
DT	0.01 s	Time step for the numerical solver.
TOTAL_TIME	40.0 s	Duration of the simulation.
MASS	1.0 kg	Mass of a single drone (m).
V_MAX	5.0 m/s	Maximum velocity saturation limit (v_{max}).
K_P	2.0	Attraction gain (k_p). Higher values = faster convergence.
K_D	1.5	Damping coefficient (k_d). Higher values = less oscillation.
K_REP	10.0	Repulsion gain (k_{rep}). Strength of collision avoidance.
R_SAFE	0.8 m	Safety radius (R_{safe}). Distance to trigger repulsion.

Table 1: Key Simulation Parameters

4. **Random Sampling:** To match the target points to N_{DRONES}, we perform random sampling without replacement (if $N_{pixels} > N_{drones}$) or with replacement (if $N_{pixels} < N_{drones}$).

5. **Scaling & Centering:** The points are centered at $(0, 0)$ and scaled by a factor (e.g., 0.35) to fit within the physical simulation boundaries.

5.3 Vectorized Physics Engine (`src/physics.py`)

To ensure the simulation runs efficiently with $N = 1000$ drones, the force calculations are **vectorized** using NumPy, avoiding slow Python loops.

5.3.1 Function: `compute_forces`

This function calculates the net acceleration for all drones simultaneously.

- **Input:** `positions` ($N \times 3$), `velocities` ($N \times 3$), `targets` ($N \times 3$).
- **Pairwise Distances:** Instead of iterating $O(N^2)$ times, we use broadcasting to compute the difference matrix:

```
diff_matrix = positions[:, None, :] - positions[None, :, :]
```

  This results in an $(N \times N \times 3)$ tensor containing the vectors between every pair of drones.

- **Masking:** A boolean mask `dist_matrix < R_SAFE` identifies neighbors that are too close, and repulsion is only calculated for these pairs.

5.3.2 Function: `velocity_saturation`

Implements the saturation logic:

$$v_{new} = v_{old} \cdot \min\left(1, \frac{V_{MAX}}{\|v_{old}\|}\right) \tag{12}$$

This ensures no drone exceeds the physical speed limit of the motors.

5.4 Video Processing (`src/video_processing.py`)

For Task 3, dynamic targets are extracted from video frames.

- **Background Subtraction:** We use `cv2.createBackgroundSubtractorMOG2` to isolate moving foreground objects from the static background.
- **Temporal Sampling:** To synchronize the video (typically 30 FPS) with the simulation (100 FPS), the targets are updated at specific intervals, allowing the physics engine to interpolate the motion smoothly.

5.5 Main Execution Scripts

- `main_task1.py`: Solves the Initial Value Problem (IVP) where targets are static.
- `main_task2.py`: Solves a transition problem. The initial state Y_0 is the final state of Task 1, and the target state T is the "Happy New Year" formation.
- `main_task3.py`: Solves the dynamic tracking problem where $T = T(t)$ changes at every frame.