Autonomous Drone Fleet Coordination Simulator

Difficulty Level	Full
Suggested Language	Java
Maximum Team Size	3
Skill Set	Java, GUI, graphics

1. Introduction

In modern agriculture and environmental research, fleets of drones are increasingly deployed for **precision crop monitoring, mapping, and spraying**. Coordinating these drones in real-world environments is complex and costly; therefore, **simulation** offers a safe and efficient way to test coordination and control algorithms before hardware implementation.

This project focuses on the design of an **Autonomous Drone Fleet Coordination Simulator**, which models the **collective behavior of multiple drones** flying in a shared 3D environment. Each drone behaves as an **independent autonomous agent** capable of movement, target tracking, communication, and collision avoidance.

The simulator will allow you to explore how **object-oriented programming concepts** (encapsulation, modularity, inheritance, composition) can be used to build complex interacting systems from simpler class components.

2. Problem Description

The project is to design and implement a simulation that models the **coordinated movement of a fleet of drones** operating in a 3D area. Each class encapsulates a specific physical or computational aspect of the drone fleet, forming a hierarchical workflow that guides the simulation. The system integrates **file-based initialization and result storage**, ensuring reproducibility and parameter traceability across simulation runs.

3. Solution Design (Tentative)

The **Autonomous Drone Fleet Simulation Platform** models the collective and coordinated operation of multiple quadrotor drones for agricultural field surveying and precision spraying. The system is designed as a modular and object-oriented architecture in which each class corresponds to a distinct physical or computational process derived from the mathematical model.

At the start of the simulation, the **Simulator** reads configuration data (e.g., number of drones, Δt, environment size, drone parameters) from an **input file** and initializes the environment, drone fleet, and communication channels. Each **Drone** instance operates as an autonomous entity possessing its own control systemmand communication module. Drones receive target waypoints or formation commands and adjust thrust and orientation accordingly.

The simulation advances in discrete time steps (Δt).
At each time step:

1. The **Controller** computes the thrust and desired attitude required to achieve target trajectories.
2. The **Drone** class applies these control inputs to its dynamic model, computing acceleration and angular velocity through Newton–Euler equations.
3. The **FormationManager** modifies drone behavior based on neighbor positions to maintain formation or coverage.
4. The **CollisionAvoidance** module applies repulsive potential fields to prevent drone overlap.
5. The **CommunicationModule** simulates the stochastic exchange of state information among nearby drones.
6. The **Simulator** integrates translational and rotational motion equations for all drones.
7. The **Logger** records all relevant parameters (positions, velocities, thrusts, sensor readings, communication events) and save them in output files.

This hierarchical flow ensures that physical laws, control logic, and cooperative algorithms are consistently applied to each drone at every simulation cycle.

Simulator Class

The Simulator acts as the core numerical engine of the system. It manages simulation time, updates drone states using numerical integration, and synchronizes inter-drone communications and logging. The translational motion of each drone is governed by discrete-time Newtonian dynamics:

$$v_{t+\Delta t} = v_t + a_t\,\Delta t$$
$$p_{t+\Delta t} = p_t + v_{t+\Delta t}\,\Delta t$$

where:

- p_t: position vector at time t(m)
- v_t: velocity vector (m/s)
- a_t: acceleration (m/s²), obtained from total forces acting on the drone

For rotational dynamics, the simulator integrates Euler's rotational equations:

$$\omega_{t+\Delta t} = \omega_t + \dot{\omega}_t\,\Delta t$$
$$R_{t+\Delta t} = R_t\,\mathrm{Exp}(\omega\,\Delta t)$$

where:

- ω: angular velocity vector (rad/s)
- R: rotation matrix describing drone orientation

The Simulator updates all drones simultaneously in each step, applying translational and rotational integration to maintain consistency between kinematic and dynamic states. It also coordinates message passing through the **CommunicationModule** and initiates logging procedures.

Drone Class

Drone represents an individual quadrotor in the simulation, encapsulating its physical parameters (mass, inertia, aerodynamic constants) and dynamic behavior. Each drone computes its motion based on internal forces and feedback from the controller and formation manager.

Mathematical Modelling:
The translational dynamics follow Newton's Second Law:

$$m\,a = mg + RT + F_{\text{aero}} + F_{\text{rep}} + F_{\text{form}}$$

where:

- m: drone mass (kg)
- $g = [0,0,-9.81]^T$: gravitational acceleration vector (m/s²)
- R: rotation matrix converting body frame to world frame
- $T = [0,0,T_z]^T$: net thrust vector (N)
- F_{aero}: aerodynamic drag force (N)
- F_{rep}: repulsive force for collision avoidance (N)
- F_{form}: formation control force (N)

Rotational motion follows:

$$\dot{\omega} = I^{-1}\,(\tau - \omega \times (I\omega))$$

where:

- I: inertia tensor (initialized, e.g. (0.02, 0.02, 0.04))
- ω: angular velocity
- τ: torque

The **aerodynamic drag** is computed as:

$$F_{\text{aero}} = -k_d\,v$$

where $k_d$is the drag coefficient and v is the drone's velocity vector.

Variable Dependencies:

- Initialized: m, I, k_d, p_0, v_0, R_0
- Computed: $F_{\text{aero}}, F_{\text{rep}}, F_{\text{form}}, T, \tau$

This class thus bridges the physical drone model with the software environment, updating its state based on real-time force computations.

Controller Class

The **Controller** class is responsible for computing the control thrust that drives the drone toward a specified target position or waypoint. It implements the **PID/PD control law** directly as formulated in the mathematical model.

For translational motion:

$$e_p = p_{desired} - p_{actual}$$
$$e_v = v_{desired} - v_{actual}$$

These represent how far the drone is from the goal in position and velocity.

$$a_{cmd} = k_p\, e_p + k_d\, e_v + g$$

where

- k_p, k_d= position and velocity gain constants (initialized parameters),
- g= gravity compensation vector [0, 0, -9.81].

To hover or move, the drone must produce thrust equal and opposite to net forces (gravity + acceleration).

$$T_{inertial} = m\, a_{cmd}$$

and

$$T_{body} = R^T T_{inertial}$$

Extract vertical thrust component (along body z-axis):

$$T_z = T_{body,z}$$

where

- m= drone mass (initialized),
- a_{cmd}= desired linear acceleration (computed).

Torque Control:

$$\tau = k_R(R_{target} - R) + k_\omega(\omega_{target} - \omega)$$

This relation ensures that the drone accelerates toward the target while minimizing error and damping oscillations. The proportional term $k_p(p_t - p)$corrects positional error, the derivative term $k_d(v_t - v)$stabilizes velocity changes.

The **Controller** thus represents the control system within the simulation's architecture, converting positional discrepancies into physical thrust values that are then passed to the **Drone** for motion computation.

Formation Manager Class

The **FormationManager** maintains spatial coordination among multiple drones, ensuring that they maintain desired relative positions within a formation or move cohesively as a group. It implements the **consensus-based formation control law**, which mathematically governs how each drone adjusts its motion based on its neighbors:

$$F_{form,i} = -\sum_{j \in N_i} [k_p(p_i - p_j) + k_v(v_i - v_j)]$$

In this model, $N_i$represents the set of drones within communication range of drone i. The proportional term aligns the drones' relative positions, while the velocity term equalizes their speeds to maintain smooth group movement.

The **FormationManager** applies these computed formation forces to each drone's dynamic model, influencing its acceleration and direction. By embedding this mathematical relationship, the class enables decentralized coordination — each drone acts based on local information rather than global commands, reflecting realistic swarm behavior.

Collision Avoidance Class

It applies repulsive forces to prevent collisions between drones.

$$F_{rep,i} = \sum_{j \in N_i} k_{rep} \frac{(p_i - p_j)}{\| p_i - p_j \|^2}$$

Communication Module Class

The **CommunicationModule** mathematically models radio-based data exchange among drones, integrating spatial and stochastic effects into the coordination system. It evaluates communication success probabilistically using:

$$\text{CommSuccess} = rand() > p_{loss}$$

and determines connectivity based on spatial distance constraints:

$$\| p_i - p_j \| < R_{comm}$$

Only drones that satisfy this condition can share position and velocity data during each simulation step. This mathematical integration ensures that information flow is dynamic and environment-dependent, introducing realistic uncertainty into the control and formation algorithms.

Thus, the **CommunicationModule** provides the underlying structure for data coupling between agents in the distributed mathematical model.

Environment Class

The **Environment** class defines the mathematical boundaries of the simulation space and ensures that drones remain within the permissible operational area. The positional constraints are modeled as:

$$p_x, p_y \in [0, width] \times [0, height]$$

When a drone reaches a boundary, corrective measures are applied, such as reflecting velocity or repositioning within bounds.

In this way, the **Environment** enforces the physical limits of the system and can be extended to include obstacle regions or restricted zones by defining additional mathematical boundary conditions.

Logger Class

The **Logger** functions as the data analysis and evaluation component of the simulator. It records numerical outputs derived from the mathematical state variables of all drones, such as position vectors p_i, velocities v_i, and distances $|| p_i - p_j ||$. From these, it computes system-level performance metrics defined mathematically as:

$$\text{Average Spacing} = \frac{1}{N(N-1)} \sum_{i,j} || p_i - p_j ||$$

$$\text{Collision Count} = | \{(i,j) : || p_i - p_j || < d_{min}\} |$$

$$\text{Communication Success Rate} = \frac{\text{Successful Messages}}{\text{Total Messages}}$$

These expressions allow quantitative assessment of the fleet's coordination efficiency, communication robustness, and safety during the simulation. The logger class saves data in two files:

- positions.csv: stores p, v, T_z for all drones per timestep
- metrics.txt: stores final performance summaries

The **Logger** thereby bridges the raw mathematical results with meaningful performance interpretations.