### \*\*1. Physical Layer\*\*

#### \*\*Description\*\*

This layer involves modeling drone parameters and behavior considering physical factors. OMNeT++ supports such tasks through frameworks like \*\*INET\*\* or \*\*Veins\*\*.

#### \*\*Implementation\*\*

- \*\*Drone Model\*\*:

- Create a module (NED file) for the drone, including parameters such as the number of rotors, weight, and battery capacity.

- Use these parameters in calculations, e.g., energy consumption.

- \*\*Energy Consumption\*\*:

- Add a C++ class for calculating battery discharge, depending on speed, weight, and maneuvers.

- Implement an event (`cMessage`) to decrement the battery charge during movements.

#### \*\*Approaches\*\*

- Use a custom `SimpleModule` to store and compute physical parameters.

- If advanced movement modeling is needed, consider integrating OMNeT++ with external engines like MATLAB or Python.

---

### \*\*2. Basic Protocol Layer\*\*

#### \*\*Description\*\*

The goal of this layer is to enable message exchange between drones and the operator, as well as handling basic commands (movement, stopping, interactions).

#### \*\*Implementation\*\*

- \*\*Message Protocol\*\*:

- Use OMNeT++ messages (`cMessage`) for data transmission.

- Define standard message types (e.g., `MOVE\_TO`, `STATUS`, `CHARGE`).

- \*\*Network Topologies\*\*:

- Use the INET Framework to create wireless network models.

- Leverage WiFi modules for communication simulation.

- Implement swarm topologies: centralized (through a leader) or decentralized.

- \*\*Command Handling\*\*:

- Write C++ handlers for basic commands.

- Use message queues to manage drone actions.

#### \*\*Approaches\*\*

- Categorize commands into groups (e.g., control, status requests).

- Simulate network delays and message losses for real-world scenarios.

---

### \*\*3. User Layer\*\*

#### \*\*Description\*\*

This layer allows users to write scripts that define the swarm's behavior based on the basic protocol.

#### \*\*Implementation\*\*

- \*\*User API\*\*:

- Create an interface for users to write behavior scripts.

- Scripts can be defined through `.ini` configuration files or an external interface.

- \*\*Script Execution\*\*:

- User-defined code should invoke commands from the basic protocol (`SEND\_COMMAND`, `GET\_STATUS`).

- Implement a standard interface to process user protocols.

- \*\*Visualization\*\*:

- Use OMNeT++ GUI to display the current state of drones and their trajectories.

- Add logging for analyzing the performance of user protocols.

---

### \*\*Additional Steps\*\*

1. \*\*OMNeT++ Frameworks\*\*:

- \*\*INET Framework\*\*: For implementing network communications.

- \*\*Veins\*\*: For modeling swarm interactions.

2. \*\*Customization\*\*:

- Write custom extensions for energy consumption models and user commands.

3. \*\*Optimization\*\*:

- Implement routing and coordination algorithms for improved energy efficiency.

- Use OMNeT++ statistical modules to evaluate results.

---

### 1. \*\*Navigation and Movement Commands\*\*

These commands control the position and movement of the drone:

- \*\*TAKEOFF\*\*: Initiates the drone's takeoff.

- \*\*LAND\*\*: Lands the drone.

- \*\*MOVE TO (x, y, z)\*\*: Moves the drone to a specified point by coordinates.

- \*\*HOVER\*\*: Keeps the drone hovering in place.

- \*\*ROTATE (yaw/pitch/roll)\*\*: Rotates the drone along its axes.

- \*\*SET VELOCITY (vx, vy, vz)\*\*: Sets the drone's speed in a specific direction.

- \*\*FOLLOW (target\_id)\*\*: Follows another object (e.g., another drone).

- \*\*RETURN TO BASE\*\*: Returns the drone to the starting point (or a designated base point).

---

### 2. \*\*State Management Commands\*\*

These commands are used to control and monitor the drone's state:

- \*\*STATUS\*\*: Requests the drone's current state (battery charge, coordinates, speed, altitude).

- \*\*STOP\*\*: Immediately halts all drone movements.

- \*\*CALIBRATE\*\*: Calibrates sensors (e.g., gyroscope, compass).

- \*\*SET MODE (e.g., manual, autopilot)\*\*: Switches between manual and autopilot modes.

- \*\*ARM\*\*: Activates the motors (preparing for takeoff).

- \*\*DISARM\*\*: Deactivates the motors.

---

### 3. \*\*Sensor Data and Monitoring\*\*

Modern drones are equipped with various sensors. These commands enable data collection and processing:

- \*\*GET POSITION\*\*: Requests the current coordinates.

- \*\*GET ALTITUDE\*\*: Requests the current altitude.

- \*\*GET BATTERY STATUS\*\*: Checks the battery level.

- \*\*GET SENSOR DATA\*\*: Retrieves data from various sensors (e.g., camera, LiDAR, ultrasonic sensors, barometer).

- \*\*START STREAM (e.g., video, telemetry)\*\*: Starts real-time video or telemetry streaming.

- \*\*STOP STREAM\*\*: Stops the ongoing data stream.

---

### 4. \*\*Inter-Drone Communication\*\*

These commands are useful for swarm coordination:

- \*\*BROADCAST (message)\*\*: Sends a message to all drones.

- \*\*SEND TO (target\_id, message)\*\*: Sends a command or message to a specific drone.

- \*\*SYNC POSITION\*\*: Synchronizes current coordinates with other drones.

- \*\*REQUEST NEIGHBOR LIST\*\*: Retrieves a list of nearby drones (based on network or physical topology).

- \*\*FORM FORMATION (shape)\*\*: Forms a specified shape (e.g., line, circle, triangle).

- \*\*LEAD (target\_id)\*\*: Transfers leadership to another drone.

---

### 5. \*\*Energy Management Commands\*\*

These commands focus on energy efficiency and safety:

- \*\*CHECK CHARGE\*\*: Checks the remaining battery charge.

- \*\*GO TO CHARGING STATION\*\*: Directs the drone to a charging station.

- \*\*OPTIMIZE ROUTE\*\*: Calculates the most energy-efficient route.

- \*\*POWER SAVE MODE\*\*: Switches to a power-saving mode (e.g., reduces sensor update frequency).

---

### 6. \*\*Emergency Commands\*\*

These commands ensure safe operation in critical situations:

- \*\*EMERGENCY LAND\*\*: Initiates immediate landing at the nearest safe location.

- \*\*RETURN TO HOME\*\*: Returns the drone to its starting point.

- \*\*SHUTDOWN\*\*: Immediately powers down all systems.

- \*\*ACTIVATE FAILSAFE\*\*: Activates fallback protocols (e.g., maintaining the current position).

- For collision detection need to know drone size - maybe enough radius of sphere around middle point of drone.

Functions need to be implemented in physical layer:

1. Collision detection:

\* collision as result of received command

\*Optionally: collision with another object in environment

2. Energy consumptions calculation (depends on weight, movement, battery capacity, antena's needs and sensors)

3. Movement in 3D

4. TCP/UDP protocols

5. Telemetry functions

6. Must contains different frequences for antena