DroneMission — AeroHack

Short technical report: unified mission planning and simulation for aircraft and spacecraft.

1. Overview

The framework provides a single planning engine used for both aircraft (UAV / fixed-wing) and spacecraft (CubeSat-style LEO) missions. Both domains share: decision variables, constraints interface, objective interface, and one solver. Mission configuration is centralized in *src/mission_settings.py*; the full pipeline is run via *python -m src.run_all*.

2. Architecture

2.1 Core planning engine (src/core/)

Shared abstractions used by both aircraft and spacecraft planners:

• **Decision variables** (variables.py): what the planner chooses (e.g. waypoint order, segment times, observation/downlink windows).

• **Constraints** (constraints.py): common interface returning feasible/violation; each domain registers its constraint set.

• **Objective** (objective.py): single interface to score a plan (minimize time/energy or maximize science value).

• **Solver** (solver.py): one planning method (constraint-based search/heuristic) for both domains.

2.2 Aircraft module (src/aircraft/)

• **Model** (model.py): point-mass kinematics, turn rate/bank limits, energy (fuel for planes, battery for drones). Wind as callable; waypoint altitude correction and depletion detection.

• **Constraints**: energy/endurance, maneuver limits, geofencing (no-fly polygons), altitude envelope.

• **Planner**: builds variables, constraints, objective; corrects waypoint altitudes; returns ordered route with timestamps and energy remaining per waypoint.

• **Simulation**: runs planned trajectory; Monte-Carlo over wind seeds for robustness (success rate, total-time range).

2.3 Spacecraft module (src/spacecraft/)

• **Orbit & visibility** (orbit.py): two-body propagation; ground-track and pass computation; observation and contact time windows.

• **Constraints**: pointing/slew (min time between activities), power/duty (max active time per orbit).

• **Planner**: builds variables, constraints, science-value objective; returns 7-day schedule.

• **Schedule** (schedule.py): time-ordered activities (observations, downlinks); science value from targets observed and downlinked.

3. Mission settings (summary)

All configurable parameters live in *src/mission_settings.py*.

Aircraft

• Default route: Vilnius Airport → Warsaw Chopin → Berlin Brandenburg → Lisbon Portela (waypoints with optional altitude).

• Vehicle type: Plane or UAV; fuel tank or battery capacity (J); consumption (J/s); min/max/default altitude (m).

• No-fly zones (list of polygons); Monte-Carlo: number of seeds and RNG seed.

Spacecraft

• Orbit altitude (km); ground targets (lat, lon, science value); ground station (lat, lon).

• Schedule duration (days); min slew time (s); max active time per orbit (s).

4. Outputs and validation

Full pipeline (*python -m src.run_all*) writes:

• **outputs/aircraft_mission.json**: planned route (lat, lon, alt_m, t, energy_used), energy_remaining_at_waypoints, crash_depletion if any, constraint_checks, Monte-Carlo robustness.

• **outputs/aircraft_mission_plot.png**: flight path and state vs time.

• **outputs/spacecraft_mission.json**: 7-day schedule, activities, mission_value_metrics, constraint_checks (slew, power).

• **outputs/spacecraft_schedule.csv**: activities table (type, start_t, end_t, duration_s, target_idx).

Validation:

• Monte-Carlo: *python validation/run_monte_carlo.py* prints success rate and total-time range over wind seeds.

• Unit tests: *pytest tests/ -v*; coverage: *pytest tests/ --cov=src --cov=pygame_viz --cov=webapp --cov-report=html* (htmlcov/index.html).

5. User interfaces

• **Web app** (Flask): http://127.0.0.1:5000 — Aircraft and Spacecraft tabs; editable waypoints/params; Plan + Save; map and globe; APIs for planning and mission start.

• **Pygame viz**: map with OSM tiles, waypoints, Start mission, replay; spacecraft Full Earth globe; same planning/simulation logic as run_all.