UAV Control via Cellular Automata (Master’s Project)

A research codebase exploring **Cellular Automata (CA)** for fixed-wing UAV longitudinal dynamics and control.
We embed a **PID controller** (with anti-windup) into CA update rules, model **actuator dynamics** (first-order lag, rate, saturation), inject **turbulence** using an Ornstein–Uhlenbeck (OU) process, and tune PID gains with a **Genetic Algorithm (GA)**. The framework runs disturbance/maneuver/failure scenarios and exports publication-ready **plots** and **tables**.

Why CA?

Classical control studies often depend on ODE/CFD pipelines—accurate but heavy. A CA model trades fine-grained aerodynamics for a **lightweight, discrete environment** that still exhibits **emergent, qualitative behaviors** (e.g., damped recovery after a pitch-up). This lets you iterate on controller ideas and robustness testing **quickly**, then decide which hypotheses merit higher-fidelity simulations.

Features

- **CA state**: attitude `a`, stability `s`, speed `v`; local diffusion + control coupling.
- **Turbulence**: OU process with adjustable intensity schedule.
- **Actuator**: first-order lag + rate limit + saturation.
- **PID**: anti-windup; failure-window toggles for robustness tests.
- **GA tuner**: elitism, tournament selection, crossover, multiplicative mutation (log-space friendly).
- **Experiments**: pitch-up maneuvers, turbulence blocks, controller failure windows.
- **Metrics**: overshoot, time-to-recover, stability variance, control effort, crash flag.
- **Artifacts**: CSVs + PNGs for direct use in theses/papers.
- **Modular**: easy to extend (add metrics, controllers, or scenarios).

Repository Structure

uav_ca_masters/ ├─ src/ │ ├─ ca/ │ │ ├─ grid.py # CA grid/state (a, s, v) + neighbors │ │ └─ update.py # CA update rules + crash condition │ ├─ dynamics/ │ │ ├─ actuator.py # first-order actuator w/ rate & saturation │ │ └─ noise.py # OU turbulence process │ ├─ control/ │ │ └─ pid.py # PID with anti-windup │ ├─ opt/ │ │ └─ ga.py # genetic algorithm (elitism + tournaments) │ ├─ experiments/ │ │ ├─ sim.py # scenario runner │ │ └─ scenarios.py # turbulence/maneuver helpers │ └─ analysis/ │ ├─ metrics.py # compute metrics; returns scalars + timeseries │ └─ plots.py # matplotlib plots (timeseries & GA history) ├─ scripts/ │ ├─ run_demo.py # quick baseline vs "GA" candidate; exports plots/CSVs │ └─ run_ga_search.py # GA tuner (edit pop/gens; extend as needed) ├─ outputs/ # generated CSVs & figures ├─ configs/ │ └─ default.yaml # example config for long runs ├─ tests/ │ └─ test_metrics.py # minimal test of metrics keys ├─ requirements.txt └─ README.md # this file

---

Quickstart (Windows/PowerShell)

From the project root (folder containing `src` and `scripts`):

```
python -m venv .venv
.\.venv\Scripts\Activate.ps1

python -m pip install --upgrade pip setuptools wheel
python -m pip install -r requirements.txt

# If you run scripts directly, ensure they can import src/
# Option A: run as module
python -m scripts.run_demo
# Option B: or run script after adding this to top of scripts/run_demo.py:
# import os, sys; sys.path.insert(0,
os.path.abspath(os.path.join(os.path.dirname(__file__), '..')))
# Then:
# python .\scripts\run_demo.py

Artifacts appear in outputs/:
- CSVs: timeseries_baseline.csv, timeseries_ga.csv, metrics_summary.csv
- Plots: baseline_*.png, ga_*.png (attitude, stability, cmd/effort)

## Reproducible Runs

Set seed in scripts/run_demo.py and GA search scripts.
Keep every table in CSV (do not hand-edit tables in Word).
Use consistent figure sizes/labels; re-generate on any change.

## Running GA Tuning

Open scripts/run_ga_search.py and edit:

best, hist = ga_optimize(eval_fn, pop_size=30, gens=30, seed=11)


Then run:

python -m scripts.run_ga_search

This writes outputs/ga_history.csv and prints best (Kp, Ki, Kd); re-evaluate the
winner on multiple seeds using experiments/sim.py + analysis/metrics.py, summarize
with mean ± std.
```