System Overview
This project implements a MAVSDK-based offboard trajectory-following application for PX4. A user-defined 3D trajectory in local NED coordinates is executed using offboard position and velocity setpoints, targeting applications such as aerial 3D printing on platforms like the Freefly Alta X. The system runs external to the flight controller and communicates with PX4 over MAVLink via MAVSDK. The system relies exclusively on PX4 offboard position and velocity interfaces and is largely vehicle-agnostic and compatible with other PX4-supported platforms.

Architecture
A user-provided `.txt` file of positional waypoints is converted into a dense trajectory with position and velocity setpoints, either via linear interpolation with bounded velocity or through offline smoothing using `Ruckig` to enforce acceleration and jerk limits. Trajectory loading is handled by `traj_io.py`, which defines a waypoint as a 3D position and velocity and treats the trajectory timestep `dt` stored in the CSV as authoritative at runtime.

Offboard execution is implemented in `offboard.py` and the trajectory follower in `controller.py`. These scripts handle connection, health checks, arming, offboard entry, trajectory execution via MAVSDK, and landing. Control handoff is managed through PX4 offboard mode, with standard PX4 failsafes allowing immediate pilot takeover if offboard mode is exited or setpoint streaming stops. A top-level CLI (`run_traj.py`) allows selection of the MAVSDK connection URL and trajectory file.

A supporting state interface (`state.py`) was developed during an earlier attempt at online trajectory generation using `Ruckig`, which required live state updates and PX4 timestamps. This approach was not pursued further to avoid additional complexity and scope expansion.

Trajectory Generation and Dynamic Feasibility
Trajectory design itself is largely outside the scope of this challenge and is assumed to be handled by the end user, particularly given application-specific constraints such as material properties in 3D printing. Since the challenge specifies only XYZ waypoint input, this implementation prioritizes position tracking, with velocity, acceleration, and jerk treated as secondary constraints. The simple generator uses linear interpolation with bounded velocity, while the optional `Ruckig`-based generator additionally enforces acceleration and jerk limits to produce smoother cornering behavior.

Results and Discussion
The results shown below were obtained using the `jMAVSim` simulator with the default `iris` vehicle model. The `jMAVSim` simulator was chosen due to its relatively lightweight implementation and low latency when developing on a PC through WSL.

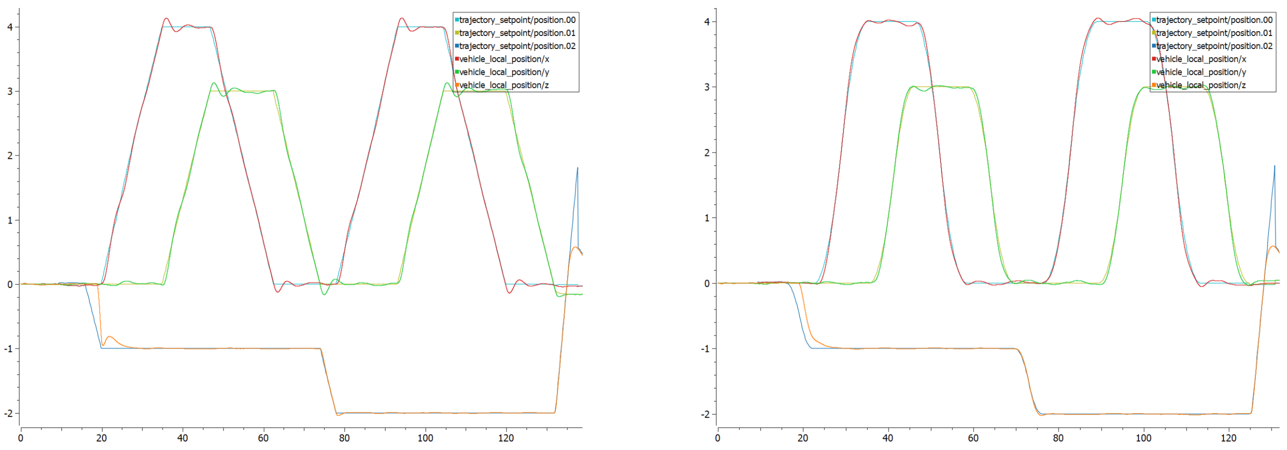

Figure 1: Trajectory tracking results in SITL. Left: tracking of simple trajectory within linearly-interpolated velocities. Right: tracking of a smoothed, jerk-bounded trajectory.

As observed in the results, positional tracking along a single axis is generally accurate, but damped oscillations are present during cornering maneuvers and when settling after direction changes. Using a smoothed, jerk-bounded trajectory with more gradual corners reduces the magnitude of these oscillations, but does not eliminate them entirely. Since the default PX4 position controller is used, this behavior is expected as the controller gains are tuned conservatively for stability rather than aggressive tracking performance. Controller gains were not modified due to the scope of this challenge and PX4 parameter porting for the submission. It is reasonable to expect that modest increases in controller gains would further improve tracking performance. More advanced approaches could incorporate vehicle dynamics during trajectory generation or perform online trajectory generation.

Safety
Waypoints and trajectories are validated before execution, and all trajectories are bounded by user-defined velocity (and optionally acceleration and jerk) limits. The vehicle is only armed after PX4 commander health checks pass (with GCS connection intentionally overridden). Offboard mode is entered only after a valid local position estimate is available and errors are caught. The drone lands and disarms automatically.

Path to Hardware Deployment
Transitioning from simulation to real flight would involve incremental testing with increasing trajectory complexity, improved trajectory generation methods, tuning PX4 position and velocity controller gains for the target platform on fixed apparatus and in free-flight, validating failsafe behavior, and integrating pilot override mechanisms.

Acknowledgment of Use of AI
As with many modern software projects, this work benefited from limited use of generative AI. AI assistance was restricted to mundane tasks such as file and CLI argument parsing, simple mathematical operations, inline comment suggestions, and documentation cleanup. No agentic AI was used for code generation. The design and implementation of all core features remain my own work.