ChatGPT

PX4 Simulation MVP Project Charter

Overview and Goal

This project aims to set up a **PX4 Software-In-The-Loop (SITL) simulation** environment using Gazebo for a quadcopter UAV. The **Minimum Viable Product (MVP)** will demonstrate a basic quadrotor (drone) model flying in simulation – specifically, performing a simple **takeoff, hover, and land** sequence. We will leverage the existing `px4-sim-suite` repository (which contains PX4, Gazebo models, and QGroundControl source) to build and run the simulation. The end goal is to confirm that the PX4 flight stack can control a virtual quadcopter in Gazebo, providing a foundation for further development.

MVP Scope and Features

- **Vehicle Type**: A single **quadrotor drone** model will be supported in the MVP. We will use a model that is standard in PX4 examples – for instance, the "**X500**" quadrotor (a generic quad used in recent PX4 Gazebo simulations) [1]. This is analogous to the classic "Iris" model used in many earlier PX4 SITL examples [2] [3]. Using a well-supported reference model ensures minimal configuration effort.
- **Simulation Environment**: The PX4 autopilot software will run in SITL mode on Ubuntu 24.04 (within WSL), interfacing with the **Gazebo** simulator. Gazebo will simulate the drone's physics and sensors in a virtual world, while PX4 runs the flight control algorithms. The project will use the **latest Gazebo (Ignition)** simulation engine, as this is the default on Ubuntu 22.04+ environments [4] (Gazebo "Harmonic" or newer). The `px4-sim-suite` repo includes the **PX4 Gazebo models** submodule, which provides all necessary vehicle and world definitions [5].
- **Flight Demonstration**: The MVP will focus on a **smoke test flight** – launching the simulation and performing an *autonomous takeoff to hover* and then a *landing*. This will verify end-to-end integration (PX4 ↔ Gazebo). The takeoff/land can be triggered by PX4 shell commands (e.g. using `commander takeoff` to initiate an automated takeoff [6], then `commander land` to descend). No complex missions or obstacle interactions are in scope – just basic flight stability in a controlled environment.
- **User Interface**: For MVP, direct command-line control and log observation will suffice. **QGroundControl (QGC)**, which is included in the repo, is optional – it can automatically connect to the simulated vehicle for manual control or visual monitoring [5]. However, since the agent will run in a headless WSL environment, we will primarily use console commands or simple scripts to control the vehicle. (Developers can still run QGC on their host machine to observe the simulation if needed.)

Out of Scope (for MVP): multi-vehicle simulation, non-quadrotor vehicle types, ROS integration, advanced missions or payload testing. Those can be added in future phases once the basic quadrotor simulation is validated.

Repository Structure and Components

The `px4-sim-suite` repository is a monorepo containing all the pieces needed for simulation development:

- **PX4 Autopilot (Firmware)** – Located under `px4/`, this is the full PX4 flight control source code (flight stack, modules, drivers, etc.). It will be built in SITL mode. The repo's `px4` folder is essentially a clone of the **PX4-Autopilot** codebase (with build system, source, and configuration files).
- **Gazebo Models** – Located under `px4-gazebo-models/`, this provides SDF model files, robot definitions, and world files for Gazebo simulation. It corresponds to the official PX4 Gazebo model repository [5]. Having this locally means the simulator has access to standard vehicle models like the X500 quadrotor, without needing to download from the internet at runtime.
- **QGroundControl** – Under `qgroundcontrol/`, the full QGC source code is provided. This can be built to run a ground control station GUI if needed. (For the headless MVP test, building QGC is not strictly required, but the source is available for completeness or future use.)
- **Documentation and Tools** – The repo contains documentation (`README.md`, `docs/`, `AGENTS.md`) and possibly scripts under `tools/` and `tests/` to guide how an autonomous agent or developer should work with the repository. The **AGENTS.md** likely outlines conventions for AI agents contributing to the project – the agent should review those guidelines once it checks out the repo.

This consolidated structure ensures the agent has offline access to all necessary code and models. The agent will primarily work in the `px4/` directory to build and run the SITL, and use the model files from `px4-gazebo-models/` when launching Gazebo.

Development Environment Setup

Setting up a correct development environment is crucial. We assume **Ubuntu 24.04** running in **WSL2** (Windows Subsystem for Linux) as the platform. The following steps and resources will prepare the environment:

- **Dependencies Installation**: Run the PX4 provided setup script to install all required dependencies (compilers, ROS2 if needed, Gazebo simulator, etc.). PX4's standard Ubuntu setup script (`Tools/setup/ubuntu.sh`) will install the necessary Gazebo packages by default [7] (on Ubuntu 22.04+ this installs the newer Gazebo "Harmonic" version and related libraries). The agent should execute this script in a new WSL Ubuntu environment to quickly get all dependencies. (Note: If the script is outdated for Ubuntu 24.04, refer to updated PX4 documentation or manually install the `gazebo` / `gz` packages for Ubuntu 24.04.)
- **Building PX4 SITL**: Navigate to the `px4/` directory and use the PX4 build system to compile the SITL target. For example, to build and launch the default quadrotor simulation in one step, use:

  ```
  cd px4
  make px4_sitl gz_x500
  ```

  This single command will compile the PX4 Autopilot for SITL and then start the PX4 SITL process along with Gazebo, loading the X500 quadrotor model and a default world [8] [1]. The `gz_x500` target uses Gazebo (Ignition) with the X500 drone model. (If needed, other vehicle targets are available with the `gz_` prefix in the same fashion [1], but X500 quad is our focus.)

- **Headless Mode**: Since WSL may not have a GUI or hardware acceleration for graphics, running Gazebo in headless mode is advisable. This avoids launching the Gazebo GUI, running only the physics server. To do this, prepend the build command with `HEADLESS=1`. For example:

  ```
  HEADLESS=1 make px4_sitl gz_x500
  ```

  This will start the simulation without any Gazebo window [9]. The PX4 SITL console will still appear, allowing interaction via the PX4 shell. Headless mode uses fewer resources and is suitable for automated testing or environments without displays [9].
- **Networking**: By default, PX4 SITL uses UDP ports to communicate with external tools (e.g., port 14550 for telemetry). QGroundControl running on the Windows host should automatically detect the SITL on WSL (assuming WSL network is accessible) [5]. If the agent or developer wants to use QGC for monitoring, simply launch QGC on Windows and it should connect to the simulated vehicle. No special configuration is needed if using the default settings. (If that fails, ensure Windows firewall allows UDP 14550, or configure the PX4 `SYS_COMPANION` parameter as needed – but usually default works out-of-box).
- **QGroundControl (optional)**: For the agent's purposes, QGC is not required to achieve the MVP. However, if a developer wishes to visualize the drone's state or manually control it, they can use QGC. The latest QGC can be downloaded from the official site or built from source. (On Windows, simply installing QGC and running it is the easiest way to interface with the WSL-hosted SITL.) The repo's included QGC source can be compiled on Ubuntu if a Linux GUI is available, but this requires Qt and is beyond MVP scope.

Development Workflow and Branching Strategy

We will follow a structured Git workflow to implement and deliver the MVP features. Key guidelines for development:

- **Branching Model**: All development should occur on feature branches, *not* directly on the `main` branch. For each discrete task or feature (e.g., adding a launch script, updating docs, etc.), the agent should create a new branch off `main`. We suggest a naming convention for clarity, for example:
- Feature implementations: `feature/<brief-description>` (e.g., `feature/px4-gazebo-iris`)
- Bug fixes or minor changes: `fix/<description>` (e.g., `fix/sitl-launch-path`)
- Documentation updates: `docs/<description>` (if needed)
  Use lower-case and hyphens or underscores to separate words. This helps maintain consistency. (If an external system pre-defines branch names or IDs for the agent, the agent should follow that instead – since "agents usually don't get to pick the name of the branch they create a PR against," the naming convention may be predetermined by the platform orchestrating the agent. In absence of an imposed name, use the above convention as a guideline.)
- **Commits**: Make frequent, small commits with clear messages. Each commit message should succinctly describe *what* change is made and *why* if not obvious. For example: `"Add script to automate takeoff/land in SITL"` is more helpful than `"update code"`. This will aid reviewers in understanding the changes. If possible, follow a consistent style (such as Conventional Commit format with prefixes like `feat:`, `fix:`, etc.), but clarity is the priority.
- **Pull Requests**: Once a feature branch achieves its goal (e.g., the simulation can successfully take off and land), the agent will open a Pull Request (PR) to merge into `main`. The PR description should outline the changes, reference any relevant issue/ticket, and include instructions on how

to test the new functionality. Because this project involves an AI agent contributing, PRs will be a critical point for a human maintainer to review and approve the changes. The agent should be prepared to respond to feedback or make modifications if the PR tests fail.

- **Code Review & Testing**: All code will be reviewed before merging. The agent should ensure that any new code (scripts, config changes, etc.) are tested locally in the WSL environment. For instance, if the agent writes a takeoff script, it should run it to confirm the drone actually takes off in simulation. Automated tests (if set up in a CI pipeline) should pass as well. If the repository includes a test suite (see `tests/` directory), the agent should run those tests to avoid regressions.
- **Documentation**: Update documentation as features are added. For the MVP, this means documenting how to run the simulation and the expected outcome. The repository's `README.md` or `docs/AGENTS.md` should be updated to include instructions for the basic simulation usage (commands to build, how to invoke the takeoff/land test, etc.). This ensures that future developers or agents can easily reproduce and build upon the MVP.

Throughout development, refer to any guidelines in `AGENTS.md` (there may be multiple, e.g. in `docs/` or `tools/`). These likely contain project-specific conventions for automated agents (for example, how to format PR titles or how to log progress). Adhering to those will smooth the integration of the agent's contributions.

Steps to Reach MVP

To achieve the MVP goal (a quadcopter that can launch, hover, and land in simulation), the following roadmap is proposed:

1. **Environment Preparation** – *Timeline: Day 1*
2. Install dependencies on the Ubuntu 24.04 (WSL) system using the PX4 setup script or manual apt commands (including Gazebo).
3. Verify that Gazebo (Ignition) is installed by running `gz --version` or similar. Also install any needed build tools (CMake, Ninja, etc., many of which the PX4 script covers).

4. Build QGroundControl (optional): If a decision is made to use QGC in Linux, set up Qt and attempt a build in WSL. (Skip if using QGC on Windows host or if not needed for MVP test.)

5. **Build and Run PX4 SITL** – *Timeline: Day 1-2*

6. In the `px4/` directory, run the `make px4_sitl gz_x500` command to compile PX4 and launch the simulation [8]. The first build may take some time as it compiles the PX4 code. Confirm that the build finishes without errors and that Gazebo starts.
7. If running headless (`HEADLESS=1`), Gazebo's server will run in background. You should see the PX4 shell (`pxh>` prompt) in the console, indicating PX4 is up and waiting for the simulator to connect [10] [11]. Once you see messages about GPS fusion starting (or no warnings about missing simulator), the connection is established and the vehicle is ready to arm [12].

8. **Troubleshooting**: If PX4 does not connect to Gazebo, or you see a timeout warning [13], ensure that the Gazebo server started properly. You might need to run the `simulation-gazebo` script from `px4-gazebo-models` manually [14] or check network settings. Also confirm that the `PX4_GZ_MODEL` environment variable or submodule paths are correct so PX4 can find the X500 model. (Given the submodule is present, this should be automatic.)

9. **Basic Flight Test (Takeoff/Land)** – *Timeline: Day 2*

10. With the simulation running (PX4 shell active and vehicle initialized on the ground), command the vehicle to take off. In the PX4 shell, enter:

    ```
    commander takeoff
    ```

    This will switch the vehicle to **Takeoff mode** and begin an automatic arming and throttle-up sequence [6]. PX4 will ramp up the quad's motors and lift off to a default altitude (typically around 2.5 meters for the takeoff command).
11. Let the drone hover for a short period (a few seconds). Observe console output for any errors. If QGroundControl is connected, you should see the drone's state change to "armed" and altitude rising.
12. Next, command the drone to land by entering:

    ```
    commander land
    ```

    The vehicle should smoothly descend and disarm after touch down. Monitor the console for confirmation messages (e.g., *Landing detected*, *Disarming*).
13. Success criteria: The drone takes off, hovers, and lands without manual intervention beyond the commands, and without the simulation or PX4 crashing. This confirms the core integration works.

14. If anything goes wrong (for example, the vehicle refuses to arm or take off), check that the vehicle is in a position mode (by default, `commander takeoff` should handle mode switching). Also ensure that home position was initialized (PX4 usually needs a GPS lock or simulated position; the `[ecl/EKF] commencing GPS fusion` message indicates the state is good [15]). In simulation, the GPS is fake but provided after a few seconds. If needed, one can manually unlock by commanding `commander arm` then `commander takeoff` as a backup, or set the parameter `COM_ARM_WO_GPS` to allow arming without GPS in a pinch – but this usually isn't necessary for the standard iris/X500 models.

15. **Automation and Verification** – *Timeline: Day 3*

16. Automate the smoke test if possible. For example, write a simple script or modify a PX4 startup script to execute the takeoff and land commands after a delay. This could be a Python script using MAVLink/MAVSDK to issue takeoff/land commands, or even an expect script feeding input to the PX4 shell. Automation will allow the test to be run headlessly and repeatedly.
17. Validate the automation: run the script in the WSL environment and ensure the sequence completes. This can be integrated into a CI workflow or as a demo for the PR.

18. Gather logs or console output from the test run to include in documentation or for analysis. PX4 generates a log file (`*.ulg`) for each run (in `px4/logs/` by default). These can be reviewed with Flight Review or the PX4 log tools, but for MVP it's enough to note that the flight was completed.

19. **Documentation and Cleanup** – *Timeline: Day 3*

20. Update the repository README or docs to include instructions on how to run the SITL demo. Clearly explain any prerequisites (e.g., "install Gazebo via `ubuntu.sh` script, then run the make command, then use PX4 shell commands to fly"). Include the exact commands for takeoff/land so that even someone unfamiliar can reproduce the test.

21. If there were any code changes or additions (perhaps adding a launch script or configuration file), ensure they are committed and pushed. Double-check that no large unnecessary files are added (the repo is already heavy with submodules, so avoid adding binaries).

22. Review compliance with coding standards or agent guidelines (line lengths, license headers if needed, etc.). Ensure all new code is appropriately commented for clarity.

23. **Open Pull Request** – *Timeline: Day 4*

24. Once satisfied, the agent will push the feature branch to the remote and create a Pull Request to merge into `main`. The PR should be titled descriptively (e.g., "MVP: PX4 Gazebo SITL with X500 quad – takeoff/land demo") and include a summary of changes.
25. In the PR description, describe how maintainers can run the simulation themselves. For example: "To test, run `HEADLESS=1 make px4_sitl gz_x500`, then at the PX4 console type `commander takeoff` and `commander land`. The drone should lift off and land as expected." Mention that the documentation has been updated accordingly.
26. Attach or link any relevant logs or screenshots (if GUI was used) to show evidence of the successful flight. For instance, a screenshot of QGroundControl showing the drone in air, or a snippet of the console output showing the altitude change.
27. After creating the PR, the agent should monitor feedback or CI results and be ready to make adjustments.

By following the above steps, we ensure the MVP is delivered in a systematic, verifiable manner. The timeframe is relatively short (a few days) since much of the heavy lifting (PX4, models, etc.) is using existing components – our work is mainly integration and testing.

Additional Resources and References

To assist the development process, below are key resources with more detailed information:

- **PX4 SITL Gazebo Documentation** – Official PX4 guide for Gazebo simulation (both Gazebo Classic and newer Ignition). This covers installation, running simulation, and common usage. In particular, see the sections on starting a quadrotor in Gazebo [2] [3] and using the PX4 shell to take off [6]. The new Gazebo (Ignition) documentation lists the supported `make` targets (e.g., `gz_x500` for the X500 quad) [1] and how models/worlds are provided via submodule [5]. These guides also discuss headless mode and standalone mode if you need to run PX4 and Gazebo separately [9] [13].
- **QGroundControl User Guide** – Instructions on installing and using QGC can be found on the QGroundControl website or the QGC guide. This is useful if you want a GUI to monitor the simulation or to manually control the vehicle. (Note: for Windows, simply downloading the installer is the quickest way to get QGC running [16].)
- **PX4 Dev Guide – Source Code Management** – The PX4 dev guide has a section on git usage and examples [17] which might provide insight into best practices for branching and contributing. While our project has its own conventions, it's good background on how the PX4 community handles contributions.
- **MAVSDK (Optional)** – PX4's MAVSDK library allows writing Python/C++ scripts to control the drone via MAVLink (e.g., arm, takeoff, land programmatically) [18]. We aren't required to use MAVSDK for MVP, but if automation via script is desired, MAVSDK is a high-level option. The MAVSDK documentation and examples (such as offboard control from Linux [18]) could be a helpful reference if we go down that route.

With these resources and a clear development plan, the agent should have everything needed to implement the MVP successfully. The focus will be on careful integration and testing rather than creating new complex functionality from scratch. Once this MVP is in place, we will have a solid foundation to expand the simulation capabilities, add more vehicle types or scenarios, and integrate advanced features in subsequent iterations.

---

1 5 8 9 13 14 Gazebo Simulation | PX4 Guide (main)
https://docs.px4.io/main/en/sim_gazebo_gz/

2 3 6 7 10 11 12 15 Gazebo Classic Simulation | PX4 Guide (main)
https://docs.px4.io/main/en/sim_gazebo_classic/

4 17 18 Simulation | PX4 Guide (main)
https://docs.px4.io/main/en/simulation/

16 Download and Install - QGC Guide - QGroundControl
https://docs.qgroundcontrol.com/master/en/qgc-user-guide/getting_started/download_and_install.html