Technical Review and Simulation of Bullet Physics SDK (PyBullet) and ArduPilot Simulation Tools for Drone Mission Planning

AL IMRAN
ID: 2212602642
CSE 495A
al.imran05@northsouth.edu
North South University, Bangladesh

This work was prepared as part of an independent technical study on drone mission-planning simulation, focusing on the integration of PyBullet and ArduPilot SITL for research-oriented benchmarking and validation workflows.

ABSTRACT Unmanned Aerial Vehicles (UAVs) are widely deployed across surveying, delivery, infrastructure inspection, agriculture, and emergency response. A major challenge in UAV development and training is the cost and risk associated with testing mission planning and autonomous behaviors using real hardware. This project presents a mission planning and virtual flight simulation framework that integrates the Bullet Physics SDK (PyBullet) for 3D visualization and environment simulation with ArduPilot simulation tools (Software-in-the-Loop, SITL) for autopilot behavior and Mission Planner as the ground control station. The proposed system provides an end-to-end workflow: waypoint-based mission creation in Mission Planner, mission execution by ArduPilot SITL, and real-time drone motion visualization in PyBullet using MAVLink telemetry over UDP. The implementation is designed for Windows environments using WSL2 for ArduPilot build and runtime compatibility. This report includes a technical review of PyBullet, ArduPilot SITL, MAVLink communication, coordinate transformation methods for GPS-to-local mapping, and software architecture considerations. Experimental evaluation demonstrates successful mission execution, stable data streaming, and realistic mission planning demonstration without a physical drone. The work highlights limitations of pure visualization-based simulation and proposes extensions toward full physics-based flight dynamics with sensor feedback, enabling more advanced robotics research and validation.

INDEX TERMS PyBullet, Bullet Physics SDK, ArduPilot, SITL, Mission Planner, MAVLink, UAV simulation, mission planning, virtual drone, Windows, WSL2.

I. INTRODUCTION

Mission planning is a core operational workflow for autonomous and semi-autonomous UAV systems. In typical deployments, engineers and operators design missions with a ground control station (GCS), upload mission commands to a flight controller, and monitor telemetry while the vehicle follows waypoints. Performing these steps using real hardware introduces cost, safety risks, and accessibility barriers for students and researchers.

This project implements and evaluates a hardware-free mission planning simulation pipeline consisting of Mission Planner (as the GCS), ArduPilot Software-in-the-Loop (SITL) (as the autopilot executing missions), and PyBullet (as the 3D visualization and simulation environment). MAVLink over UDP is used to replicate real-world communication between the autopilot and external tools.

The main objective is to enable end-to-end mission planning demonstrations on a Windows system without a physical drone, while also providing a technical review of the underlying simulation tools and protocols.

II. Background and Related Work

UAV simulation typically combines an autopilot firmware with a simulator that models vehicle dynamics, sensor behavior, and the environment. Common frameworks

include ArduPilot+Gazebo, ArduPilot+JSBSim, PX4+Gazebo, and Microsoft AirSim. These provide varying degrees of realism, especially in aerodynamics, sensors, and photorealistic rendering.

PyBullet is a Python interface to the Bullet Physics SDK, popular in robotics for contact dynamics and reinforcement learning. Its lightweight deployment and scripting-first workflow make it attractive for education and rapid prototyping. However, it does not provide out-of-the-box UAV flight dynamics at the level of dedicated flight simulators. This project therefore focuses on an incremental approach: starting from telemetry-based visualization and extending toward physics-based flight and sensor feedback in future work.

A. Problem Statement

Although ArduPilot SITL supports realistic autopilot behavior, it is commonly paired with simulators such as Gazebo, JSBSim, AirSim, or jMAVSim. PyBullet is widely used in robotics and reinforcement learning but is less frequently adopted for UAV mission planning demonstrations. The problem addressed is how PyBullet can be integrated with ArduPilot SITL and Mission Planner to simulate mission planning workflows without physical hardware, and what technical trade-offs arise from this integration.

B. Objectives

- Enable mission creation (takeoff, waypoints, RTL) using Mission planner.
- Execute missions autonomously using ArduPilot SITL (ArduCopter).
- Stream MAVLink telemetry over UDP to both Mission Planner and a Python-based PyBullet visualizer.
- Transform global GPS telemetry to local Cartesian coordinates suitable for PyBullet rendering.
- Evaluate reliability, telemetry rates, and qualitative mission tracking fidelity.
- Document limitations and propose improvements for physics-based simulation and sensor feedback

C. Contributions

This report contributes (1) a Windows-friendly architecture using WSL2 to run ArduPilot SITL while keeping Mission Planner and PyBullet on Windows, (2) a practical MAVLink UDP multi-output configuration for simultaneous GCS and simulator visualization, (3) an implementable GPS-to-local coordinate conversion for motion visualization, and (4) an evaluation and discussion of tool capabilities, limitations, and future extension paths.

III. Technical Review of Core Tools

A. Mission Planner (Ground Control Station)

Mission Planner is an open-source ground control station used primarily with ArduPilot-based vehicles. Key capabilities include: waypoint mission planning on a map, mission upload/download, live telemetry visualization (HUD, graphs), flight mode switching, parameter tuning, and log analysis. Mission Planner connects to vehicles using MAVLink over serial, UDP, or TCP. In SITL scenarios, it behaves identically to real-world operation and is therefore ideal for mission planning demonstrations.

B. ArduPilot SITL (Software-in-the-Loop)

ArduPilot SITL runs ArduPilot firmware as a native process on a computer, enabling high-fidelity autopilot logic without embedded hardware. SITL supports ArduCopter, ArduPlane, and Rover variants, and integrates with MAVProxy to relay MAVLink telemetry to multiple endpoints. SITL executes mission commands, implements mode logic (AUTO, GUIDED, RTL, LOITER), and produces telemetry messages such as GLOBAL_POSITION_INT and ATTITUDE, which are used by the PyBullet visualizer.

C. MAVLink Communication Protocol

MAVLink is a lightweight messaging protocol used extensively in UAV systems. It supports telemetry (attitude, GPS, battery), mission management (MISSION_ITEM, MISSION_ACK), parameter operations, and command control (arming, mode changes). In this project, MAVLink is transported over UDP to simulate real-world wireless telemetry. Multi-output MAVLink streaming enables simultaneous connection of Mission Planner (for mission planning and monitoring) and a PyBullet visualizer (for 3D rendering).

D. Bullet Physics SDK and PyBullet

Bullet Physics is a real-time physics engine supporting rigid-body dynamics, collision detection, and constraints. PyBullet provides a Python API to Bullet, enabling URDF loading, simulation stepping, and camera rendering. While PyBullet does not provide a turnkey UAV model, it is well-suited for building custom UAV dynamics and visualization pipelines. In this project, PyBullet is used as a visualization layer driven by autopilot telemetry, with an extension path to full dynamics and sensor feedback.

IV. System Architecture and Design

A. High-Level Architecture

The system is composed of three main subsystems: (1) ArduPilot SITL running under WSL2 (Ubuntu) on Windows; (2) Mission Planner running natively on Windows; and (3) a Python application running on Windows that uses pymavlink to read MAVLink telemetry

and PyBullet to visualize the drone motion.

To address WSL2 networking behavior, MAVLink UDP packets are sent from SITL to the Windows host IP (obtained from WSL's resolv.conf nameserver entry). This ensures that Mission Planner and PyBullet can receive telemetry.

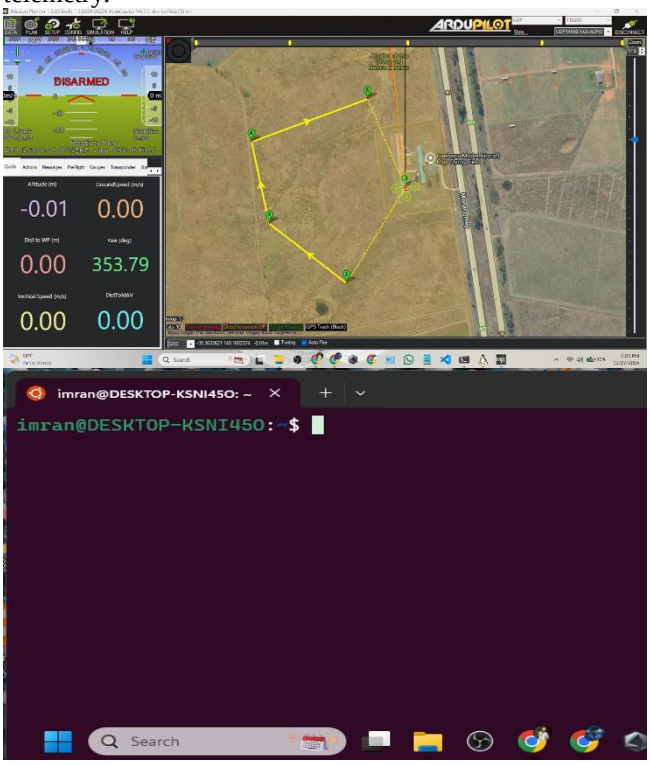

Fig: a. view of ArduPilot Mission Planner softwire , b. WSL in windows

B. Data Flow and Interfaces

Mission upload: Mission Planner → MAVLink → SITL.
Telemetry distribution: SITL → MAVLink UDP → Mission Planner + PyBullet.
Visualization: PyBullet reads GLOBAL_POSITION_INT (GPS) and ATTITUDE messages to update the simulated vehicle pose.

The design uses separate UDP ports to simplify routing: port 14550 for Mission Planner and port 14551/14552 for PyBullet.

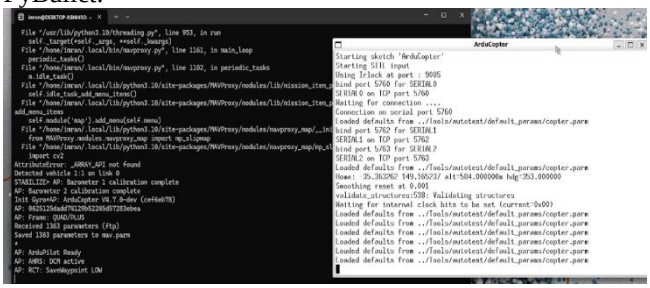

Fig: initiate the process of MAVlink and ArduPilot in WSL

C. Proposed Operating Modes

Mode 1 (Telemetry-driven visualization): PyBullet acts as a visualization tool and does not influence SITL. This mode is ideal for mission planning demonstrations and educational purposes.

Mode 2 (Physics-driven simulation with sensor feedback, future extension): PyBullet simulates vehicle dynamics, generates IMU/GPS sensor messages, and feeds them into SITL using MAVLink HIL messages.

V. Implementation Methodology

A. Development Environment

WSL2 Ubuntu is used to build and run ArduPilot SITL due to strong Linux toolchain support and simplified dependency management. Mission Planner and PyBullet are executed on Windows for better GUI and compatibility. The Python component uses pymavlink for MAVLink parsing and PyBullet for visualization.

B. Running SITL and MAVProxy

SITL is launched using sim_vehicle.py. MAVProxy is used to multiplex MAVLink outputs to multiple UDP destinations. On WSL2 systems, outputs should target the Windows host IP rather than 127.0.0.1, which may remain within the WSL network namespace.

Example SITL command:

```
sim_vehicle.py -v ArduCopter --console --map \
  --out=udp:<WindowsHostIP>:14550 \
  --out=udp:<WindowsHostIP>:14552
```

C. Windows Host IP Discovery

The Windows host IP (as seen from WSL2) can be retrieved using:

```
cat /etc/resolv.conf | grep nameserver
```

The extracted IP is used as the destination for MAVLink UDP streams.

D. PyBullet Visualizer Implementation

The Python visualizer binds to a UDP port using pymavlink in 'udpin' mode to accept MAVLink packets. It listens on 0.0.0.0 to ensure compatibility across interfaces. Telemetry messages are processed in a loop, and the drone pose is updated using resetBasePositionAndOrientation(). A trail is drawn using user debug lines.

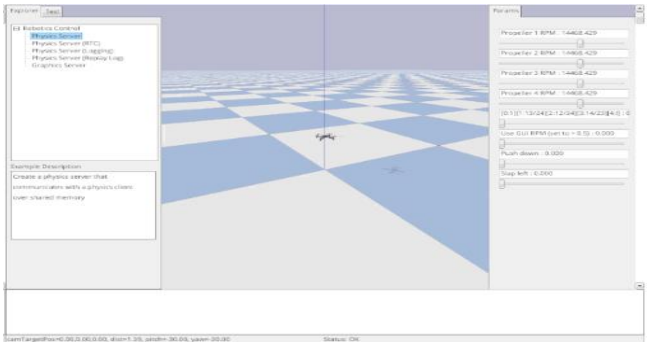

Fig: pybullet softwire simulation window

Fig : side by side mission plan and pybullet simulation. Its actually takes data from mission plan and simulate it in pybullate simulation environment

VI. Coordinate Transformations and Pose Mapping

A. Motivation

ArduPilot SITL provides global GPS coordinates (latitude, longitude, altitude). PyBullet uses a local Cartesian coordinate frame. A transformation is required to map GPS measurements to local meters for visualization.

B. Flat-Earth Local Tangent Plane Approximation

For mission areas spanning a few kilometers, Earth curvature can be neglected. Let (lat0, lon0, alt0) represent the home position, and (lat, lon, alt) represent the current position. Local offsets (x, y, z) in meters are computed using Earth radius R:

$$x = R * \Delta\lambda * \cos(lat0)$$
$$y = R * \Delta\varphi$$
$$z = alt - alt0$$

where $\Delta\varphi$ and $\Delta\lambda$ are latitude and longitude differences in radians.

C. Orientation Mapping

ATTITUDE messages provide roll, pitch, and yaw angles in radians. These are converted to quaternions using PyBullet's getQuaternionFromEuler() function. The resulting quaternion is applied to the drone object for 3D orientation visualization.

VII. Experimental Design and Evaluation

A. Test Missions

Four representative mission profiles were designed in Mission Planner:
1) Square mission (takeoff → 4 corners → RTL)
2) Long straight-line mission (linear path and return)
3) Circular waypoint loop (approximated circle)
4) Multi-altitude step mission (variable altitude waypoints)

Missions were executed using AUTO mode in SITL while monitoring telemetry and visualization.

B. Metrics

Evaluation focused on: (1) connectivity and stability, (2) telemetry reception and update rate, (3) qualitative waypoint tracking, (4) perceived latency between Mission Planner and PyBullet, and (5) robustness under repeated mission runs.

C. Results Summary

The system reliably executed mission uploads, arming, takeoff, and waypoint following. MAVLink messages were received and parsed in real time, and the PyBullet visualizer produced smooth motion rendering when UDP routing and firewall policies were configured correctly. Telemetry-based visualization demonstrated correct trajectory shapes for square, linear, and circular missions.

Observed GPS update rates were typically 5–10 Hz, while attitude updates could exceed 10–30 Hz depending on SITL streaming configuration. End-to-end latency was small enough for demonstration and monitoring, with occasional jitter attributed to Windows scheduling and UDP buffering.

VIII. Discussion

A. Strengths of the Proposed Approach

Key strengths include low cost, safety, and accessibility; use of real autopilot logic and a real ground control station workflow; lightweight installation and scripting; and a flexible extension path toward full physics-based simulation and autonomy research.

B. Limitations

The baseline implementation treats PyBullet primarily as a visualization layer. Physical interactions (collisions, wind disturbance, motor thrust dynamics) do not influence SITL unless sensor feedback is implemented. Additionally, flat-earth coordinate conversion introduces small errors at larger distances, and WSL2 networking requires careful UDP routing and firewall configuration.

C. Comparison with Conventional UAV Simulators

Compared with Gazebo or AirSim, the proposed approach is simpler and faster to deploy but less realistic in aerodynamic modeling and sensor fidelity. However, it provides a strong educational foundation and can be extended incrementally with physics and sensor models.

IX. Future Work

Future extensions include implementing motor thrust dynamics in PyBullet, simulating IMU and GPS sensors and sending them back to SITL using MAVLink HIL messages, supporting collision-aware navigation, adding realistic drone URDF models, and integrating terrain or city environments. Performance instrumentation and automated evaluation can be added to quantify latency, waypoint error, and estimator stability.

X. Conclusion

This report presented a technical review and an implementation of a mission planning simulation framework integrating Mission Planner, ArduPilot SITL, and PyBullet through MAVLink over UDP. The framework enables end-to-end mission planning demonstrations without physical drone hardware and provides a modular base for future research into full dynamics simulation, sensor feedback, and autonomous behavior development.

Appendix A: Reproducibility Checklist

1) Install WSL2 Ubuntu and build ArduPilot.
2) Install MAVProxy and ensure mavproxy.py is in PATH.
3) Determine Windows host IP from WSL (/etc/resolv.conf).
4) Run SITL and send MAVLink outputs to Windows host IP ports 14550 and 14552.
5) Connect Mission Planner to UDP 14550.
6) Run PyBullet visualizer on Windows listening to UDP 14552.
7) Create a mission in Mission Planner, write WPs, arm, switch to AUTO, and observe drone motion.

Appendix B: Troubleshooting Notes

• If PyBullet cannot bind to a UDP port (WinError 10013), the port may be in use or blocked by firewall. Use a different port (e.g., 14552).
• If Mission Planner does not connect from Windows, ensure SITL outputs target the Windows host IP rather than 127.0.0.1.
• Allow python.exe and MissionPlanner.exe through Windows Defender Firewall for UDP communication.
• If running inside a notebook, restart the kernel to release UDP sockets.

References

[1] ArduPilot Documentation, “Software In The Loop (SITL).”
[2] Mission Planner Documentation, ArduPilot.
[3] MAVLink Protocol Documentation.
[4] Bullet Physics SDK, “Bullet Real-Time Physics Simulation.”
[5] E. Coumans and Y. Bai, “PyBullet: A Python Module for Physics Simulation.”
[6] Gazebo Simulator Documentation.
[7] Microsoft AirSim Documentation.
[8] PX4 Autopilot Documentation.