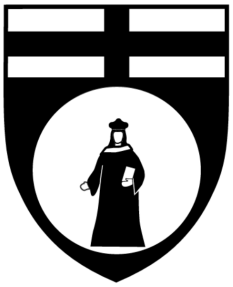

Dibris Dipartimento di Informatica, Bioingegneria,
Robotica e Ingegneria dei Sistemi

Autonomous Drone Navigation

Virtual Reality for Robotics

Advisor: Prof. Gianni Viardo Vercelli
Co-advisors: Luca Martini
Academic Year: 2025–2026

Team

Student Name	Student ID
Sayna Arghideh	5934809
Mahnaz Mohammad Karimi	6212087
Alireza Tajabadi Farahani	6212483
Amir Mahdi Matin	5884715

Contents

1 | Introduction

This project simulates an autonomous drone mission in a realistic city environment. The drone must move through an urban area and reach specific building positions defined by the user. The mission is executed inside **Unreal Engine 5.2.1**, using **Cesium for Unreal** to load a 3D city map, **Cosys-AirSim** to simulate the drone, and **Python** to control the drone and implement navigation logic.

The drone's task is:

- Take off and climb to a **safe height** above the ground and obstacles.
- Navigate between buildings **waypoint-by-waypoint** (waypoints are placed in Unreal Engine).
- Stop at target buildings and generate logs such as **"Color Delivered"**.
- Perform a "painting" behavior on a building by moving vertically along the **Z axis** (up/down) while avoiding collisions.
- Detect a **Forbidden Zone** in the city and stop before entering it.
- End the mission in a safe position while remaining stable (hover).

This project demonstrates a complete pipeline: map creation, simulation integration, waypoint generation, autonomous movement, collision-safe altitude management, and forbidden-zone stopping logic.

2 | State of the Art

Autonomous drone navigation in urban areas is difficult due to dense obstacles and safety constraints. Typical challenges include collision avoidance near buildings, planning safe paths between targets, and respecting restricted zones (no-fly areas). For early-stage autonomous missions, **waypoint navigation** is widely used because it is reliable, easy to debug, and can be extended later using more advanced planning.

Simulation frameworks based on Unreal Engine are commonly used for UAV research and development because they provide high realism, physics, collision meshes, and flexible environment design. AirSim-based simulators are especially useful since they offer a direct API for drone control and sensor access, enabling rapid testing before real-world deployment.

3 | Tools and Technologies

3.1. Unreal Engine 5.2.1

Unreal Engine was used to build the simulation level and manage all scene elements, including:

- waypoints as actors,
- building collision meshes,
- forbidden zone trigger volumes,
- blueprint logic for waypoint export and debugging.

3.2. Cesium for Unreal (3D City Map)

A realistic city map was created using Cesium for Unreal by streaming 3D tiles. The city location is selected using geographic coordinates (example: Genova **44.4071° N, 8.9347° E**). This approach allows the simulation to resemble real-world urban structure.

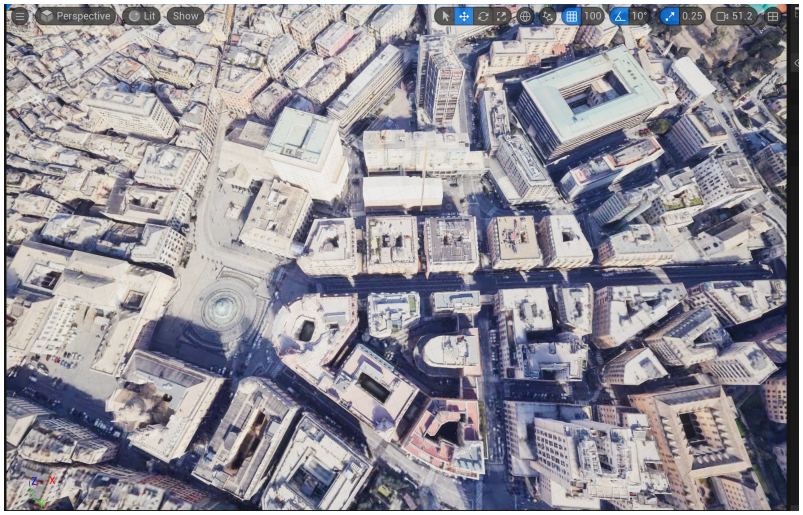

Figure 3.1: Cesium city map loaded in Unreal Engine.

3.3. Cosys-AirSim

Cosys-AirSim is an open-source simulation framework (based on Microsoft AirSim) that runs as an Unreal Engine plugin and provides high-fidelity simulation for drones and other vehicles, together with extensive API support. It can be integrated into any Unreal environment by adding the plugin to the project, allowing the user to control simulated vehicles through programmatic interfaces (e.g., Python). [**cosysAirSimRepo**]

In this project, Cosys-AirSim is used to provide:

- a multirotor drone model and physics-based flight simulation,
- an API for autonomous control (takeoff, move commands, hover, state queries),
- logging and debugging support from the simulator,
- camera and sensor interfaces (e.g., RGB, depth) that can be extended for perception-based methods.

Cosys-AirSim is a maintained research-focused fork that extends the original AirSim with additional features and Unreal Engine version support (including long-term support branches such as UE 5.2.1).

3.4. Blueprints (Waypoint Export + Debug)

A dedicated Blueprint **Actor class** (`BP_WaypointExporter`) is used at runtime to gather waypoint actors placed in the level. On `BeginPlay`, the Blueprint retrieves all waypoint actors, filters them using an actor tag (e.g., `Wp`), and prints each waypoint's **actor name** and **world location (X,Y,Z)** for debugging and verification.

Waypoints are **manually named** in the editor (e.g., `wp_01` ... `wp_08`) and tagged to ensure they can be found programmatically. The printed output is used to verify correct waypoint placement and ordering.

Note: Unreal Engine locations are in **centimeters** and follow the Unreal coordinate system (Z-up). For drone navigation, these waypoint coordinates are converted into **AirSim NED coordinates** (meters, Z-down) before being used by the Python controller.

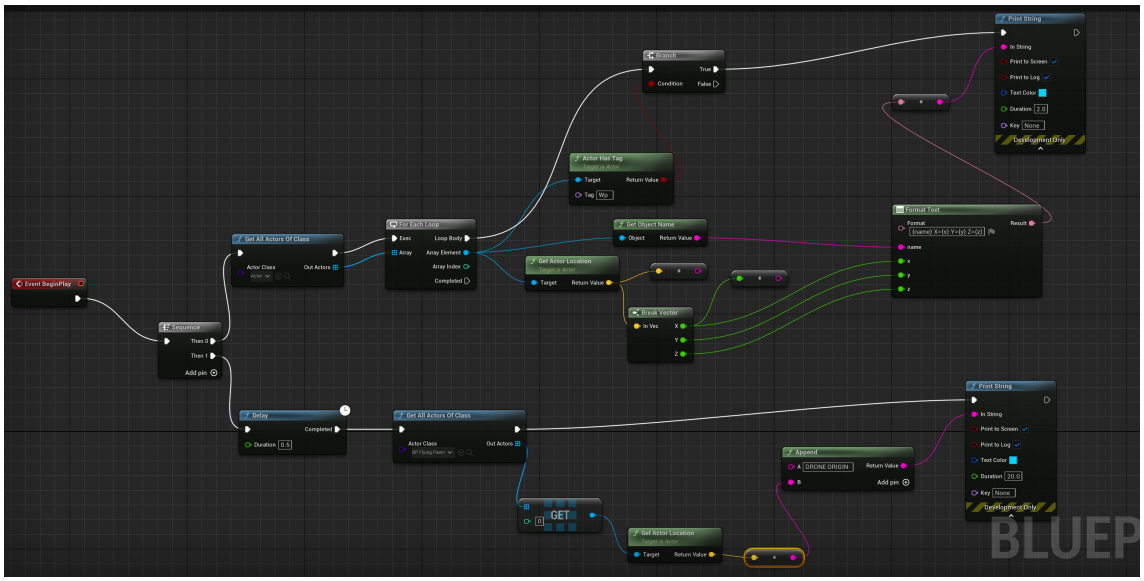

Figure 3.2: Blueprint event graph (`BP_WaypointExporter`) used to print waypoint names and positions for debugging/export.

3.5. Python (AirSim API Controller)

Python is used to control the drone mission. The script:

- connects to AirSim,
- takes off,

- moves to safe altitude,
- navigates through waypoints automatically,
- executes painting at a selected waypoint,
- checks forbidden zone (target and path),
- stops safely and logs events.

4 | Project Description and Control Logic

4.1. Scenario Overview

The simulation models an urban drone mission inside a streamed city environment. The drone starts from an initial spawn/origin position and visits multiple buildings. Each building is represented by a manually placed waypoint actor, and the mission follows the waypoint order defined by the waypoint naming convention.

The mission is divided into phases:

1. Connect to the simulator, arm the vehicle, take off, and stabilize.
2. Climb to a global safe cruise altitude before any long horizontal motion.
3. Navigate through the ordered list of building waypoints.
4. At each waypoint, pause and log a simulated delivery event (e.g., "Color Delivered").
5. At one selected building, perform a "painting" action using a vertical sweep while maintaining safe clearance.
6. Evaluate forbidden-zone constraints for the next navigation step; if a violation is detected, stop before entry and terminate safely.
7. End the mission by hovering at a safe altitude.

4.2. Waypoints and Coordinate Frame

Waypoints are authored inside Unreal Engine and verified/exported using the Blueprint tool described in Chapter 3. The exported Unreal world locations (cm, Z-up) are

converted to AirSim NED coordinates (m, Z-down) before being used by the Python controller.

4.3. Drone Control Logic (Python)

The mission logic is executed using a Python script that controls the vehicle through the AirSim API. The controller implements a simple finite sequence of actions: initialization, climb to safe altitude, waypoint traversal with delivery logging, an optional painting sweep, and forbidden-zone stopping.

4.3.1. Initialization

The controller connects to AirSim and prepares the drone for flight:

- Connect to the AirSim simulator instance.
- Enable API control and arm the multirotor.
- Execute takeoff and wait until the vehicle is stable.

4.3.2. Safe Altitude Policy

Before long horizontal navigation, the drone moves to a global safe cruise altitude `SAFE_Z` in the **NED** frame. Because NED uses **z-down**, a higher altitude corresponds to a **more negative** z value. The safe altitude is a predefined cruise value chosen to maintain clearance above typical obstacles (e.g., rooftops) plus an additional safety margin.

4.3.3. Waypoint Navigation

For each waypoint in the ordered list:

1. Read the waypoint position (after conversion to AirSim NED coordinates).
2. Compute the commanded cruise altitude z_{cmd} (typically based on `SAFE_Z` unless a special behavior is active).
3. Command the vehicle to move to $(x_{\text{ned}}, y_{\text{ned}}, z_{\text{cmd}})$.
4. Hover briefly to stabilize before logging and moving to the next waypoint.

4.3.4. Delivery Logging

At each building waypoint, the drone pauses and prints a log message (e.g., `Color Delivered`) to confirm that the waypoint was reached and the simulated delivery action was completed.

4.4. Painting Behavior (Vertical NED Movement)

To simulate painting at one selected building, the controller executes a vertical sweep near the target waypoint. This behavior is performed along the **NED z-axis (z-down)**:

- Moving **downward** corresponds to increasing z (less negative / more positive).
- Moving **upward** corresponds to decreasing z (more negative).

During the sweep, the controller maintains safe clearance from building geometry by limiting the vertical range and applying conservative margins (e.g., by keeping the sweep within a predefined safe band around the cruise altitude and/or using collision-aware constraints if available).

4.5. Forbidden Zone Definition and Stopping

A forbidden zone is defined in Unreal Engine using a box trigger volume to represent a restricted region of airspace near a building or sensitive area. In the Python controller, the forbidden zone is modeled as a rotated 3D box defined by:

- center position,
- extents (half-dimensions),
- yaw rotation,
- additional safety margin.

All forbidden-zone checks are performed in the same AirSim NED coordinate frame used for navigation, with yaw applied consistently around the vertical axis.

Before executing motion toward a waypoint, the controller evaluates forbidden-zone constraints using two checks:

1. **Target check:** determines whether the destination point lies inside the forbidden volume.
2. **Path check:** determines whether the planned segment between the current position and the destination intersects the forbidden volume.

If either check indicates a violation, the controller prints a warning log (e.g., `Forbidden Zone`), cancels forward progress, and transitions the vehicle into a safe state by stopping and hovering at a safe altitude, thereby ending the mission without entering the restricted region.

5 | Results and Discussion

The system achieved the expected behavior in simulation:

- The drone successfully connected to AirSim, took off, and navigated inside the Cesium-streamed city environment.
- Waypoints authored in Unreal were used as mission targets without manually typing coordinates in Python (waypoints were exported and converted to AirSim NED coordinates before navigation).
- The safe altitude policy reduced risk during horizontal travel, especially in dense building areas.
- The drone performed a painting-like vertical sweep along the NED z-axis at a selected building without collisions under the tested conditions (with collision meshes enabled).
- The forbidden zone trigger volume was visible in the level and the Python target/path checks correctly caused the drone to stop before entering, with log output confirming the event.

List of Figures