Article

An Approach to Simulation & Navigation of Autonomous Unmanned Aerial Vehicle in 3D

Mubeen Ahamed Kabir Ribayee [1,2], Ogbonnaya Anicho [1] and Emanuele Lindo Secco [2,*]

[1] AI Lab, School of Computer Science and the Environment, Liverpool Hope University, Liverpool L16 9JD, UK; 22011771@hope.ac.uk (M.A.K.R.); anichoo@hope.ac.uk (O.A.)

[2] Robotics Lab, School of Computer Science and the Environment, Liverpool Hope University, Liverpool L16 9JD, UK

* Corresponding author. E-mail: seccoe@hope.ac.uk (E.L.S.)

Received: 27 June 2025; Accepted: 8 September 2025; Available online: 17 September 2025

ABSTRACT: Drone simulation refers to the emulation of *Unmanned Aerial Vehicles* (UAVs) in a virtual environment, replicating real-world conditions to study and test the behavior, performance, and functionalities of drones. This paper explores the simulation of UAVs in the *Unreal Engine* environment using *MAVProxy* (Micro Air Vehicle Proxy) and the Python library *DroneKit*. By leveraging the computational capabilities of computers, this approach enables precise visualization and control of UAV flight dynamics in three dimensions. The use of Blueprints in Unreal Engine facilitates a cost-effective and accessible simulation process, allowing engineers and scientists to refine their UAV designs before real-world deployment. Results show the applicability of this approach *vs.* different environments, where an alternative approach also emerges as a viable option for visualizing textured buildings. This approach shows the power of open-source collaboration in advancing innovative solutions in the dynamic field of science and technology.

Keywords: UAV; Drones; Unreal Engine; Blueprints; MAVLink (Micro Air Vehicle Link); SITL (Software in The Loop); 3D-visualization

1. Introduction

An *Unmanned Aerial Vehicle* (UAV) or drone is a vehicle without a pilot. An *Unmanned Aerial System* (UAS) allows communication with the physical drone [1]. These aircraft, operated remotely or autonomously, provide widespread applications across diverse fields, from recreational use and aerial photography to industrial and military purposes [2]. Drones' versatility is showcased through tasks such as surveillance, agriculture, search and rescue operations, package delivery, and scientific research. Thanks to their sensors, cameras, and communication systems, these devices can carry out a wide array of tasks, making them invaluable tools in modern society [3,4].

From a historic viewpoint, the evolution of UAVs commenced in the 1940s, marking a pivotal period in aeronautical history. The significance of these developments gained further prominence during the Iraq and Afghan Wars in the 2000s [5]. A highly reliable UAV was developed in the 1990s and became of interest for some universities and other academic institutions [5]. Later, researchers and technologists directed their endeavors towards the enhancement of both hardware and software systems. In 1997, DARPA (Defense Advanced Research Project Agency) set up a project of *Micro Air Vehicle* (MAV). The primary objective of the MAV project was to engineer a drone with a diameter not exceeding 15 cm [5,6]. Meanwhile, a transformative milestone was achieved in 2009 thanks to a collaboration between 3D Robotics and the Swiss Federal Institute of Technology in Zurich [6]. This cooperation brought to the development of the *ArduPilot Mega* (APM), an open-source autopilot system that heralded a new era in UAV autonomy [5,6]. In 2014, a study was conducted by M. Muhammad et al., where an autonomous quadcopter was introduced for home delivery, employing an Android device as the onboard processing unit, and connecting to the APM flight controller. The use of the *Raspberry Pi* (RPI) as the onboard processing unit was highlighted, enabling the execution of resource-intensive applications such as image processing, object tracking, and path planning, which are unfeasible on mobile devices alone [7,8]. Lisanti et al., in 2015 presented a system for quadcopter-assisted drug shipments, featuring an Android application for clients and pharmacies to request medicines. This research extends its scope by accommodating multiple destination points from registered users, incorporating load calculations, and achieving full autonomy using the GUIDED mode [9]. In 2016, Cameron Roberts designed a GPS-guided quadcopter with the Raspberry Pi connected to the Navio Flight Controller. This implementation received waypoints directly from users through a *Secure Shell* (SSH) protocol, lacking path planning considerations. The absence of user remote access and a *Graphical User Interface* (GUI) for submitting requests were identified as limitations. Moreover, power estimation and path planning were omitted from the author's considerations [10]. In a study by Salih M. et al., in 2018, a monitoring system utilizing a quadcopter with an attached camera to the Raspberry Pi was introduced to evaluate video transfer using different protocols. The study lacked user interaction with the system, path planning, and a notification system [11].

In this context, drone's simulation has emerged as an essential aspect of training and development in the realm of UAVs. Simulations allow the implementation of virtual environments that mirror the characteristics and behavior of real-world drones, offering users a risk-free platform for practice. This is particularly crucial for operators—whether they are beginners or experienced—who need to refine their piloting skills and to develop effective strategies for navigating challenges in different scenarios. Simulation environments also play a significant role in software and hardware testing: developers can use these simulations to refine drone's technologies, ensuring their reliability and efficiency before deploying them in real-world situations. The ability to conduct scenario testing, including adverse weather conditions and emergency situations, contributes to the overall safety and preparedness of drone operations. Furthermore, drone simulation serves as a valuable tool for regulatory compliance training.

In practical applications, the integration of a companion computer such as the Raspberry Pi [3] or Nvidia Jetson Nano plays a pivotal role in executing control over a quadcopter. This control is achieved, for example, by interfacing with the drone's flight controller: typically, a Pixhawk can be combined with a Drone-Kit API in order to facilitate the transmission of the commands by means of MAVLink. This framework easily extends the capabilities of the autopilot software, enabling the execution of other tasks such as image processing, path planning, obstacle collision detection and more, surpassing the inherent capabilities of the flight controller on its own. However, the actual testing of these integrated systems using physical drones introduces inherent challenges, including significant costs associated with wear and tear, potential hardware failures, and unforeseen accidents. Researchers often turn to simulation environments for preliminary testing and development to mitigate these challenges and reduce potential damage.

A standard simulation setup encompasses the integration of SITL running concurrently with the *MAVLink* protocol facilitated by *MAVProxy* [4]. This configuration is integral for exchanging messages between components. *MAVProxy* acts as the intermediary responsible for transmitting and receiving messages from the simulated drone, which, in the case of this project, is an SITL copter. The implementation of such a simulation scenario enables developers to emulate and test various aspects of the drone's behavior in a controlled and safe virtual environment. A DroneKit-Python script plays an important role in this framework, serving as the conduit for sending commands to the drone. This script links with a *MAVProxy*, leveraging the *MAVLink* protocol to communicate instructions to the simulated drone through a UDP or TCP connection. Through this integration, the *DroneKit-Python* script serves as a commander, sending high-level prompts into the *MAVLink* messages that the SITL copter can then interpret. The *MAVProxy* middleware efficiently handles the translation and communication processes, providing a seamless interaction between the scripted commands and the simulated drone.

In the visualization aspect of the simulation, a *Ground Control Station* (GCS) is also used to supply a graphical representation of the drone's activities. The most famous GCSs are *Mission Planner*, *QGroundControl* and *APM Planner 2*. Unfortunately, this software does not offer spatial data since it visualize flight paths in 2D. To address this limitation, we want to explore whether the *Unreal Engine* could be implemented for the visualization, offering an immersive 3D environment for monitoring and analyzing the simulated drone's behavior. This graphical representation enhances the debugging and testing processes, allowing developers to observe and validate the impact of commands and scripts in a visually intuitive fashion. Figure 1 shows a flowchart representation of the proposed setup that we have just described.

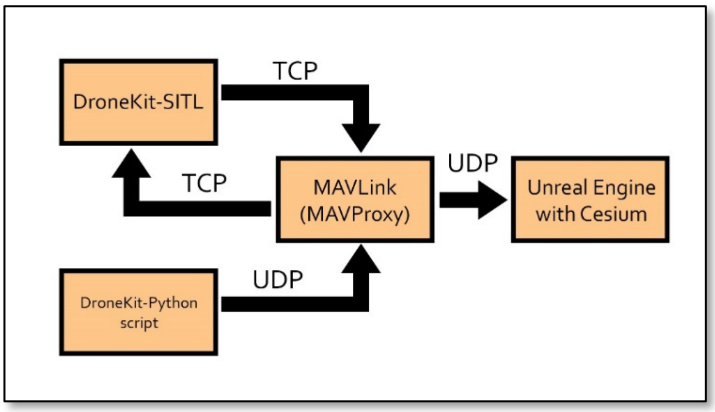


Figure 1. The main structure and setup of the proposed architecture.

Therefore, this work aims to design an Unreal Engine based simulation in 3D, which can be a visualization tool for an SITL copter drone sending and receiving messages through a *MAVLink* protocol. The proposed system integrates the Unreal Engine, MAVLink protocol and the DronKit library in a unique fashion in order to visualize and control UAV flight in 3 dimensions. The specific objectives include:

- Setting up an *SITL copter module* on a console window
- Designing a *MAVLink* protocol that connects to the drone via SITL or TCP, and sending and receiving messages
- Implementing a script that can automate and monitor a mission for the drone
- Integrating an Unreal Engine level that can simulate the globe with real global coordinates and geolocate the drone accurately according to the simulated drone setup
- Analyzing the flight path of the drone using plots.

2. Materials and Methods

The integration of Cesium with Unreal Engine to design a 3D visualization of drone simulations represents a transformative leap in the realm of UAV technology. This innovative system offers a multitude of advantages, revolutionizing training, research and development, urban planning, emergency response, precision agriculture, environmental monitoring, and various commercial applications. Table 1 shows some of the main advantages of such an approach.

Table 1. Benefits of the proposed approach.

Benefits	Description
Realistic Visualization	One of the primary advantages of this system lies in its ability to deliver a realistic and immersive visualization experience. Leveraging the power of the Unreal Engine, renowned for its unmatched graphical fidelity, and Cesium, a platform synonymous with geospatial visualization, this integrated system crafts a high-fidelity 3D environment that faithfully replicates the complexities of real-world scenarios.
Spatial Context	Adding a layer of spatial context to drone simulations is a pivotal aspect which is facilitated by the integration of Cesium. The geospatial capabilities of Cesium enable the incorporation of real-world data, allowing users to simulate missions in specific geographic locations. This spatial context proves invaluable in applications such as urban planning. Moreover, the system allows for a detailed assessment of the environmental factors and of the variations in the terrain, providing a holistic view that is indispensable in diverse fields (*i.e.*, emergency interventions, agricultural applications).
Dynamic Environments	Dynamic environments are a hallmark of real-world scenarios, and this system excels in simulating such dynamics. This functionality is instrumental in testing and refining drone responses *vs.* unpredictable situations. For instance, emergency response teams can benefit from this feature to simulate disaster scenarios with changing conditions and support the decision's process.

Interaction & Exploration	Interaction is another key strength of the Unreal Engine, especially when coupled with the geospatial skills of Cesium. Users, including drone operators and mission planners, can interact with the simulation in real-time. This interactivity allows for adjustments to flight paths, exploration of different perspectives, and on-the-fly decision-making. In essence, users are not just passive observers but contribute to more engaging and effective training and planning.
Multi-User Collaboration	Multi-user collaboration is another distinctive advantage of the proposed architecture. In scenarios where teamwork and coordination are paramount, multiple users can simultaneously be involved within the simulation. This feature facilitates collaborative training scenarios, joint mission planning exercises, and real-time decision-making, replicating the team dynamics of the actual UAV missions.
Versatility	Whether used for drone pilot training, research and development of autonomous systems, urban planning simulations, or precision agriculture scenarios, the system is a versatile tool. Its adaptability is particularly noteworthy, making it applicable in different industrial scenarios with a set of use cases and requirements.
Risk-Free Training	One of the significant advantages lies in the provision of a risk-free training environment for drone pilots and operators. With the ability to practice maneuvers, emergency responses, and complex missions in the simulation, pilots can enhance their skills without the risk of damaging physical UAVs.
Efficient Algorithm Testing	Efficiency in algorithm testing is another dimension where this system shines. Researchers and developers can test different algorithms optimizing the performance of autonomous systems.
Data-driven Decision Making	Users can analyze simulated data, assess mission outcomes, and extract insights for the optimization of further tasks. Such a data-driven approach contributes to the development of better informed decision-making approaches in real-world drone missions, ensuring continuous improvement as well.
Cross-Industry Applicability	The cross-industry applicability of this system is a key factor that broadens its impact. Its capabilities cater to diverse industries, including but not limited to agriculture, infrastructure, defense, and research, making the system an asset for a different set of applications.
Headspace for Future Technologies	As we look toward the future, this integrated system has the potential to unlock even more possibilities. The incorporation of Augmented and Virtual Reality technologies could elevate the simulation experience, allowing users to immerse themselves further for even more realistic training and planning scenarios. Additionally, the integration of blockchain technology could enhance data security and transparency, crucial for applications in sensitive areas such as defense and surveillance.

A notable and inspiring initiative in this domain is the *Royal Air Force's* (RAF) collaboration with VRAI [12], culminating in the groundbreaking Project Sphinx. This collaborative venture not only underscores the importance of cutting-edge technologies but also mirrors the approach embraced by the project at hand, leveraging Unreal Engine 5, Cesium, and similar tools for creating a *Virtual All-Domain Environment* (VADE) [13]. The goal was to develop immersive XR (eXtended Reality) simulations tailored to train troops in aerial missions, emphasizing both aircrew and ground operators. Now, with Cesium being open source [14], much of its potential is now unlocked and available to users with ideas. The integration of Cesium for Unreal Engine to simulate drone missions in 3D holds promising prospects across various domains. Some potential opportunities this project could offer are reported in Table 2.

Table 2. Potential impacts of the proposed approach.

Impact	**Description**
Training and Education	Virtual Flight Training: the 3D simulation platform holds significant promise as a comprehensive training tool for drone pilots. Educational Tool: fields such as aeronautics, geography, and engineering can harness the power of 3D simulation as an invaluable educational tool to offer students immersive and hands-on experiences.
Research and Development	Autonomous Systems Testing: Researchers and developers can use the 3D simulations to test and refine algorithms for autonomous drone systems. This contributes to advancements in artificial intelligence and machine learning. Mission Planning: The platform provides a versatile environment for experimenting and optimizing mission planning algorithms.
Urban Planning and Infrastructure Management	Drone Integration: Urban planners could use the simulation tool to study seamless integration of drones into urban environments. This may include tasks such as traffic monitoring, emergency response, and infrastructure inspection. Risk Assessment: 3D simulated drone's missions could support the assessment of potential risks associated with deploying drones in complex urban landscapes. Urban planners and decision-makers may also use these data to inform regulatory frameworks and risk mitigation strategies.

Emergency Response and Disaster Management	Preparedness Training: emergency response teams can conduct preparedness training, refining strategies for using drones in search and rescue missions. Real-time Decision Support: the platform's evolution could include features for providing real-time data during disasters, offering critical insights to coordinate an efficient response.
Precision Agriculture	Crop Monitoring: expanding the simulation tool to include precision agriculture scenarios should enable farmers to visualize and plan drone missions for crop monitoring, pest control, and yield estimation [15]. This should in turn contribute to the optimization of agricultural practices and resource management.
Environmental Monitoring	Wildlife Conservation: Conservationists may leverage simulations to study the impact of drone missions vs wildlife monitoring and conservation. Ecosystem Analysis: Environmental researchers can apply simulations to study the ecosystems and to monitor the environmental changes over time, looking at the impact of human activities on nature.
Commercial Applications	Industry-specific simulations: tailoring the simulation to specific industries, such as oil and gas, construction, automobiles [2,16], or mining, provides companies with a powerful planning and monitoring tool. Simulating drone missions within complex industrial environments can also aid in the optimization of daily life activities while ensuring safety. Regulatory Compliance: the platform's evolution could assist companies in adhering to regulatory requirements within current legal frameworks. This feature is crucial for industries where compliance is a key requirement.
Integration with Emerging Technologies	Extended Reality (XR) integration: future advancements may see the integration of *Extended Reality* technologies, enhancing user immersion in the simulation, pushing the boundaries of simulation realism [17] Blockchain Integration: Incorporating blockchain technology into the simulation enhances data security and transparency. This is particularly crucial in applications related to defense and surveillance, where secure and tamper-proof data are paramount.
Global Collaboration	Cloud-based collaboration: transitioning the simulation platform to a cloud-based model enables global collaboration and data sharing. Researchers, industries, and governmental agencies can collaborate seamlessly, sharing insights and data for a more comprehensive understanding of drone missions [14] Standardization: efforts to standardize simulation protocols and data formats foster a collaborative ecosystem for drone mission simulation across borders. Standardization facilitates interoperability and promotes a unified approach to drone mission simulation on a global scale.

The diverse applications and potential advancements outlined underscore the transformative impact that simulating drone missions in 3D using Cesium for Unreal Engine can have across various domains. As technology evolves, the project stands at the forefront of innovation, opening new possibilities for the effective deployment and management of autonomous UAVs.

2.1. DroneKit-SITL

Like the broader SITL concept, Drone Kit-SITL facilitates this simulation without the need for physical hardware, enabling developers to test and validate their code in a controlled virtual environment before deployment on actual drones. The fundamental principle underlying Drone Kit-SITL involves interfacing with the ArduPilot autopilot software through the Drone Kit API. This interface empowers developers to command the simulated drone's controller, typically Pixhawk, using MAVLink messages. This level of abstraction enables the emulation of real-world scenarios and interactions, offering a comprehensive testing ground for applications and scripts that interact with the drone's autopilot. It is worth noting the versatility of Drone Kit-SITL, allowing the simulation of various types of vehicles, including planes, copters, and rovers. This adaptability aligns with the diverse landscape of UAV applications, enabling developers to tailor their simulations to the specific characteristics and requirements of different drone configurations. Drone Kit-SITL is designed to run on multiple operating systems, including Linux, Windows, and Mac OS, enhancing its accessibility for developers across different platforms. The convenience of this cross-platform compatibility facilitates widespread adoption and collaboration in the drone development community. In this project, however, an SITL is implemented for the Copter module running purely in a Windows environment, although common practice for such a simulation would be to use a virtual Linux system for the Drone Kit API environment, and the MAVLink setup in Windows, running on the same machine.

In practical terms, the utilization of Drone Kit-SITL in a simulated environment mirrors the behavior of actual drones, providing a reliable basis for testing and debugging code. This simulation-driven approach not only mitigates the challenges and costs associated with physical testing but also accelerates the development cycle by offering rapid iteration and debugging capabilities. As such, Drone Kit-SITL becomes an invaluable resource for developers seeking to ensure the robustness, reliability, and efficiency of companion computer systems, such as Raspberry Pi or Nvidia Jetson Nano, before deploying them onto physical UAVs. The integration of Drone Kit-SITL into the development workflow exemplifies a strategic and pragmatic approach to advancing the state of autonomous drone systems, leveraging the power of simulation to refine and optimize software components in a controlled and reproducible environment.

Implementing simulations using SITL on a virtual drone replicates the functionalities and outcomes observed in real-world scenarios. The use of the ArduPilot autopilot software ensures consistency between simulated and actual drone behavior. This simulation-driven approach not only circumvents the challenges associated with physical testing but also offers a controlled and repeatable environment for refining and validating the companion computer's functionalities before deploying onto physical drones. The symbiotic relationship between companion computers and simulation environments thus becomes indispensable in advancing the development and reliability of autonomous drone systems. More about the Drone Kit-SITL can be found in its documentation provided in [18].

2.2. MAVLink and MAVProxy

The *Micro Air Vehicle Link* (MAVLink) protocol assumes a pivotal role, serving as a lightweight communication protocol designed specifically for exchanging information between systems onboard the vehicle, such as the autopilot, and external ground control stations or companion computers. Details of the MAVLink protocol are reported in Appendix A.

MAVLink plays a crucial role in enabling communication between the autopilot software, such as ArduPilot, and external entities like ground control stations or companion computers (Figure 2). This facilitates the implementation of sophisticated tasks and functionalities on companion computers, such as image processing, path planning, and other intelligent operations that complement the core capabilities of the autopilot. In the context of simulation environments like *Software in The Loop* (SITL), MAVLink ensures that the communication dynamics between simulated components accurately mirror those of real-world scenarios. This enables developers to test and validate their companion computer scripts or applications in a simulated environment before deploying them on physical UAVs.

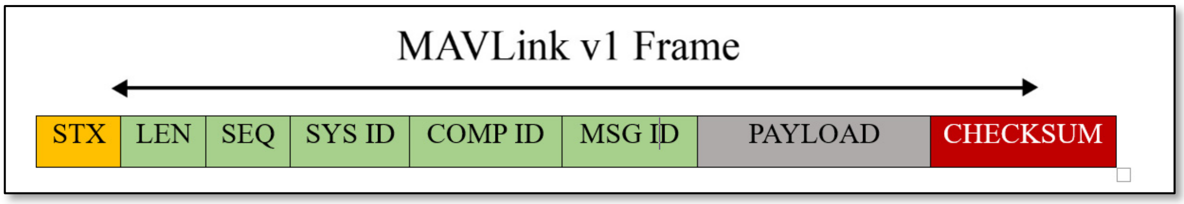


Figure 2. The *MAVLink* Message Structure.

All these steps are carried out by MAVProxy, short for Micro Air Vehicle Proxy, a versatile and lightweight command-line GCS tool widely used in the realm of UAVs. Developed to work seamlessly with the MAVLink protocol, MAVProxy serves as a communication proxy between a ground control station, companion computers, and the autopilot system on the UAV. One of MAVProxy's primary functions is to facilitate the transmission of MAVLink messages, enabling bidirectional communication between ground control stations and UAV systems. This intermediary role makes it an essential component for relaying commands, telemetry data, and mission information between various elements in a UAV ecosystem. Another notable feature of MAVProxy is its support for multiple transport layers, including serial, UDP, TCP, and more. This flexibility allows MAVProxy to adapt to various communication mediums and network configurations, enhancing its utility in diverse UAV applications. More implementations for MAVProxy and MAVLink can be found in [19] and [20].

2.3. DroneKit-Python

In the field of UAV development, DroneKit-Python emerges as a robust and versatile framework, facilitating the seamless integration of companion computers with UAV systems. Developed by 3D Robotics, DroneKit-Python serves as a Python API that empowers developers to communicate with autopilots using MAVLink. This open-source toolkit is particularly designed to interact with ArduPilot, a popular open-source autopilot software.

The primary advantage of *DroneKit-Python* lies in its simplicity and ease of use. Python, a widely adopted and user-friendly programming language, ensures developers can quickly grasp and implement UAV-related functionalities using DroneKit-Python. This accessibility is further augmented by a comprehensive set of documentation and examples provided by the DroneKit community, fostering a supportive environment for novice and experienced developers. Although the downside of this is that it does not support *Python 3.1*, and can only be run using *Python 2.1* [21], which is considered as deprecated on January 2020.

DroneKit-Python enables the creation of scripts that run on companion computers, such as Raspberry Pi [3,6] or Nvidia Jetson Nano, allowing these devices to communicate with the autopilot. This communication occurs through the MAVLink protocol, facilitating the exchange of commands, telemetry data, and mission information between the companion computer and the autopilot system. One of the key features of DroneKit-Python is its support for high-level commands. Developers can use pre-defined functions and methods to instruct the UAV to perform specific actions without delving into the intricacies of low-level MAVLink message creation. This abstraction simplifies the scripting process and accelerates the development cycle, making it more accessible for a broader range of developers [22].

The framework also supports the creation and execution of mission plans, allowing developers to define a sequence of waypoints, commands, and actions that the UAV should follow autonomously. This capability is essential for applications such as aerial surveying, surveillance, and precision agriculture, where pre-defined mission plans enhance the efficiency and autonomy of UAV operations. Moreover, DroneKit-Python facilitates real-time monitoring and visualization of telemetry data, enabling developers to assess the status of the UAV, receive live updates, and make informed decisions during mission execution. This aspect is crucial for debugging, testing, and refining UAV applications, ensuring optimal performance and reliability.

2.4. Unreal Engine

Unreal Engine (by Epic Games) has evolved into a premier and versatile game development engine that extends its influence far beyond its gaming roots. Its transformative impact is felt across diverse industries, establishing itself as a formidable force in virtual production, architecture, automotive design, and simulation [23]. Since we must simulate an entire planet (*i.e.*, rendering millions of polygons with real-world lighting and shadows calculations while still maintaining real-time rendering), Unreal Engine integrates features like *Nanite*, *World Partition* and *Lumen*.

At the forefront of Unreal Engine 5's arsenal is Nanite, a revolutionary technology that fundamentally transforms the handling of geometry in real-time rendering. Nanite employs a virtualized micro-polygon approach, allowing artists unprecedented freedom to create intricate and highly detailed 3D models without constraints imposed by traditional polygon budgets [24]. On the other side, World Partition addresses the challenges posed by large and open-world environments, providing an efficient spatial partitioning system within Unreal Engine. By dynamically streaming in relevant portions of the world based on the player's location, World Partition optimizes memory usage and enhances overall performance [25]. Finally, Lumen is a game-changing global illumination solution that redefines the dynamics of real-time lighting. Lumen is a fully dynamic, global illumination system that eliminates the need for pre-baked lighting scenarios. By realistically simulating indirect lighting, reflections, and the interplay of light in the virtual world, Lumen enhances visual fidelity and realism [26].

2.5. Cesium

Cesium is a JavaScript library that has emerged as a pivotal tool in the realm of web-based geospatial visualization. Renowned for its open-source nature, this library empowers developers to craft dynamic and interactive 3D globes and maps directly within web browsers. Leveraging the capabilities of WebGL, Cesium enables high-performance rendering of geospatial data, eliminating the need for plugins and facilitating smooth, hardware-accelerated graphics directly within the browser environment [27].

One of the standout features of Cesium lies in its support for dynamic data visualization. The library allows developers to animate and update visualizations in response to changing spatial and temporal data, providing an asset for applications requiring real-time tracking, weather pattern visualization, and time-dependent information. Cesium has extended its reach by offering integration with Unreal Engine. Unreal Engine and Cesium together contribute towards the visualization of the drone simulation flying around the virtual globe with reference to real-world coordinates [28].

2.6. Anaconda

Anaconda, a widely used open-source distribution of *Python* and *R programming languages*, offers a versatile and personalized experience for data scientists and developers. Anaconda provides a comprehensive platform designed to meet the specific needs of users in the realm of data science and analytics. The package manager, *conda*, allows for the easy installation and management of libraries, dependencies, and tools. Anaconda provides diverse pre-built packages and libraries for data analysis, machine learning, and scientific computing. Anaconda's user interface and integrated development environments (IDEs) enable users to interactively explore and manipulate data as well [29].

2.7. Mission Planner

The *Mission Planner* is a comprehensive software application used in the field of UAVs and drones. Developed to facilitate mission planning, monitoring, and control of UAVs, Mission Planner offers a versatile platform for users to customize and optimize their drone operations [30]. In particular *U Mission Planner* involves a personalized approach to planning and executing drone missions: users can plan flights, set waypoints, and customize various parameters based on their specific needs. The software's user-friendly interface enables real-time adjustments, providing a high level of flexibility during mission planning. The real-time monitoring capabilities of Mission Planner allow users to track their drone's status, telemetry data, and camera feeds during the mission. This level of detail contributes to efficient decision-making and enhances the overall control users have over their UAVs.

3. Implementation and Software Development

To implement the system, the first stage relies on the installation of the different packages and libraries. The process refers to the execution of software on a Windows 11 Machine and Python 2 [31].

3.1. Framework

The framework requires the installation of 4 main packages, namely the *Phython 2*, the *Mission Planner*, the *Unreal Engine* and the *Design of a Simulation Level*. The process allows the setting of a BP_UDP connection as shown in Figure 3. Details of the installation have been reported in Appendix B.

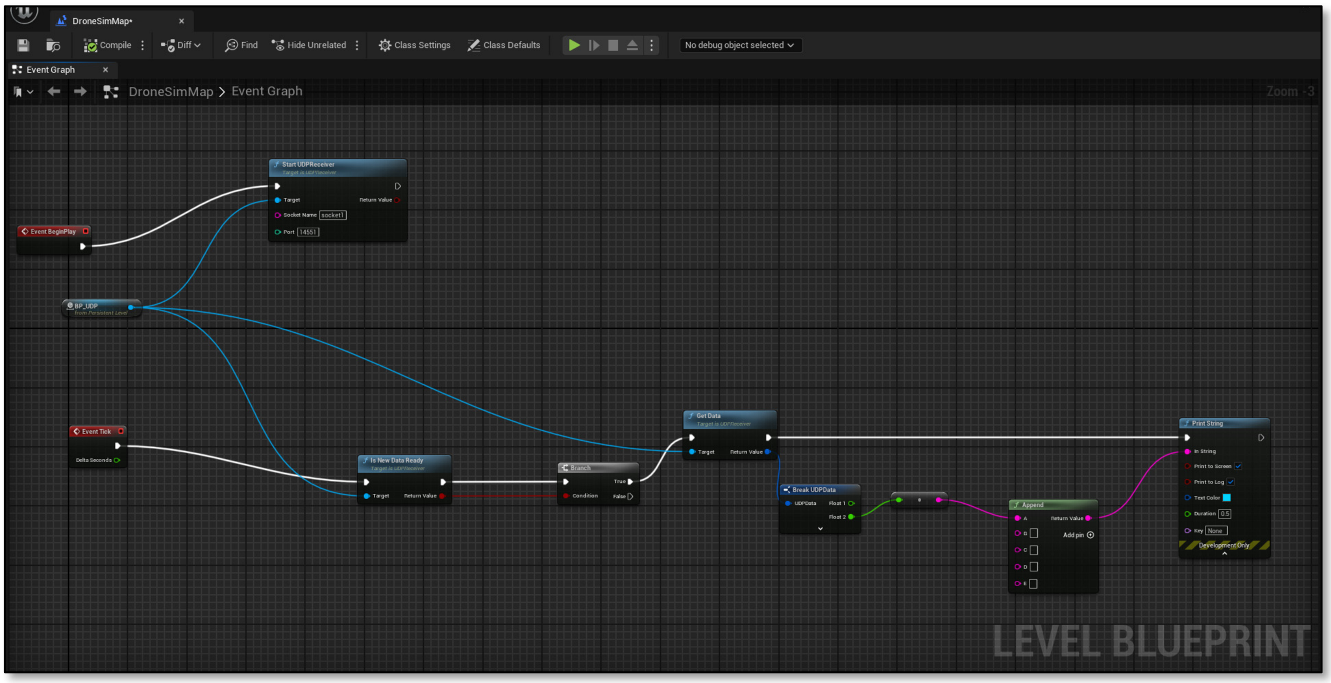
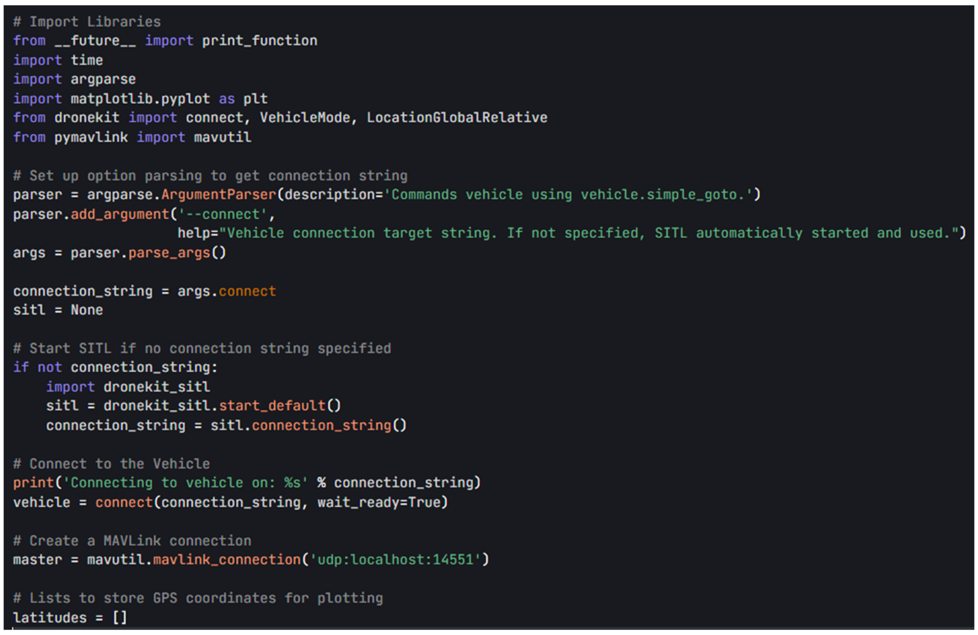


Figure 3. The BP_UDP Blueprint Visual Script.

3.2. Implementation

The correct path for the Anaconda installation and the location of 'simple_goto.py' script should be set in the batch file, along with the correct environment name, according to the specific user installation procedures and system file paths. Figure 4 shows a snippet of the code. An instance of *Mission Planner* is kept open for a top-down graphical visualization of the drone's path. While establishing the *MAVLink* protocol, *Mission Planner* automatically connects to the UDP through port 14550.



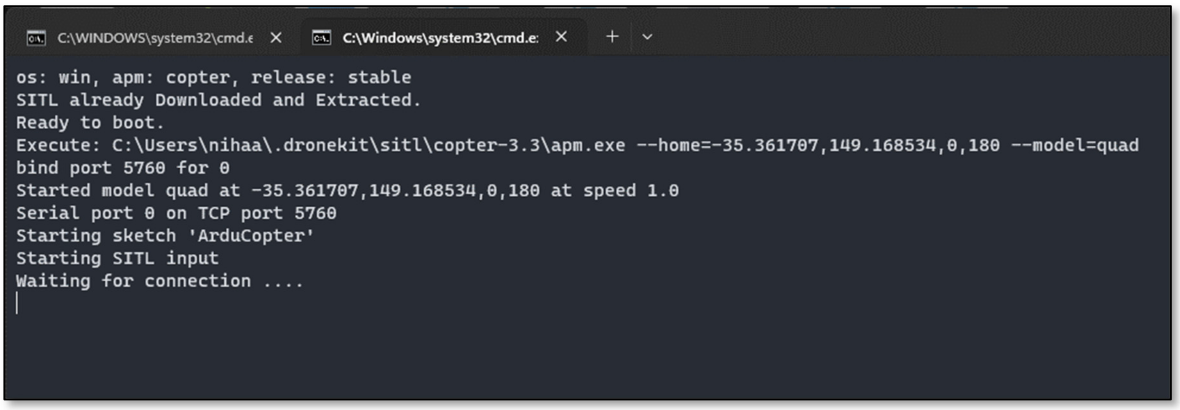
```
# Import Libraries
from __future__ import print_function
import time
import argparse
import matplotlib.pyplot as plt
from dronekit import connect, VehicleMode, LocationGlobalRelative
from pymavlink import mavutil

# Set up option parsing to get connection string
parser = argparse.ArgumentParser(description='Commands vehicle using vehicle.simple_goto.')
parser.add_argument('--connect',
                    help="Vehicle connection target string. If not specified, SITL automatically started and used.")
args = parser.parse_args()

connection_string = args.connect
sitl = None

# Start SITL if no connection string specified
if not connection_string:
    import dronekit_sitl
    sitl = dronekit_sitl.start_default()
    connection_string = sitl.connection_string()

# Connect to the Vehicle
print('Connecting to vehicle on: %s' % connection_string)
vehicle = connect(connection_string, wait_ready=True)

# Create a MAVLink connection
master = mavutil.mavlink_connection('udp:localhost:14551')

# Lists to store GPS coordinates for plotting
latitudes = []
```

Figure 4. Snippet of the *simple_goto.py*.

A *dronekit-sitl* instance for a copter is also initialized by running the following command in a terminal window, in which the pydrone2 environment is activated.

dronekit-sitl copter --home=−35.361707,149.168534,0,180

The command initializes a copter drone in the geolocation (−35.361707, 149.1685234). Once the command is run, Figure 5 can be seen, which indicates successful activation, and is ready to listen for *MAVLink* connection requests.



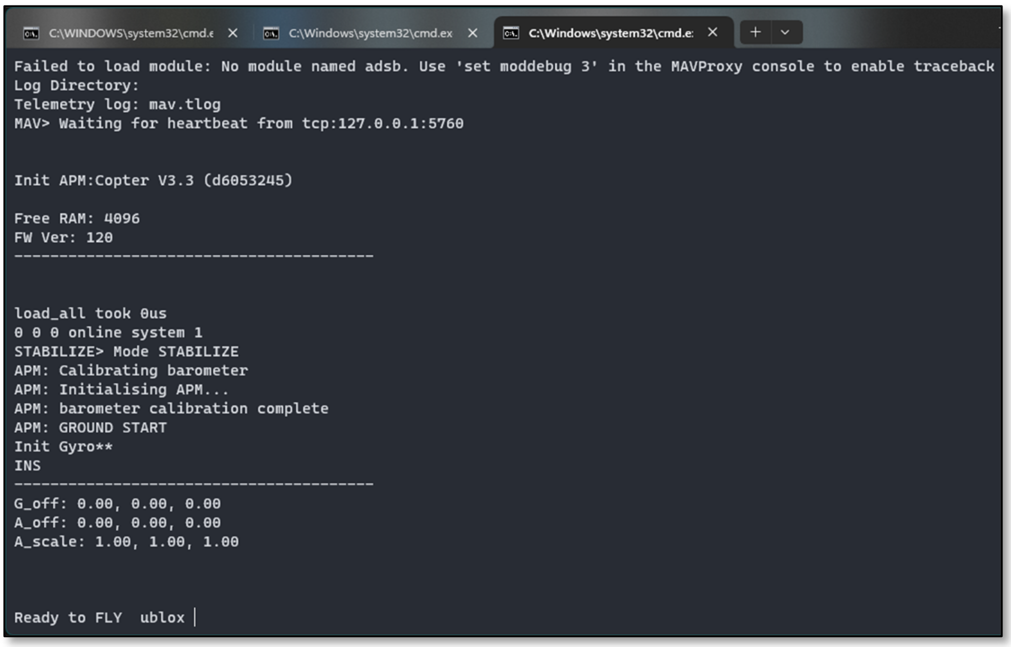
```
os: win, apm: copter, release: stable
SITL already Downloaded and Extracted.
Ready to boot.
Execute: C:\Users\nihaa\.dronekit\sitl\copter-3.3\apm.exe --home=-35.361707,149.168534,0,180 --model=quad
bind port 5760 for 0
Started model quad at -35.361707,149.168534,0,180 at speed 1.0
Serial port 0 on TCP port 5760
Starting sketch 'ArduCopter'
Starting SITL input
Waiting for connection ....
```

Figure 5. The DroneKit-SITL Initialization.

In a second terminal with the same environment activated, MAVLink protocol is initiated using the following command.

python C:\Users\nihaa\anaconda3\envs\pydrone2\Scripts\mavproxy.py --master tcp:127.0.0.1:5760 --out udp:127.0.0.1:14550 --out udp:127.0.0.1:14551 --out udp:127.0.0.1:14552

This opens *MAVLink* protocol to connect to the SITL drone module using TCP at port 5760, and send or receive commands through UDP ports 14550, 14551 and 14552. Mission Planner uses port 14550 to receive messages from the drone, 14551 by Unreal Engine and 14552 by the 'simple_goto.py' script to send location data to *Unreal Engine*. The result can be seen in Figure 6.


C:\WINDOWS\system32\cmd.e
C:\Windows\system32\cmd.ex
C:\Windows\system32\cmd.e

```
Failed to load module: No module named adsb. Use 'set moddebug 3' in the MAVProxy console to enable traceback
Log Directory:
Telemetry log: mav.tlog
MAV> Waiting for heartbeat from tcp:127.0.0.1:5760


Init APM:Copter V3.3 (d6053245)

Free RAM: 4096
FW Ver: 120
----------------------------------------


load_all took 0us
0 0 0 online system 1
STABILIZE> Mode STABILIZE
APM: Calibrating barometer
APM: Initialising APM...
APM: barometer calibration complete
APM: GROUND START
Init Gyro**
INS
----------------------------------------
G_off: 0.00, 0.00, 0.00
A_off: 0.00, 0.00, 0.00
A_scale: 1.00, 1.00, 1.00


Ready to FLY  ublox
```

Figure 6. The MAVProxy Terminal is connecting to SITL with TCP at port 5760.

The Python script *'simple_goto.py'* controls the simulated drone through *MAVLink* commands. This script essentially performs a simple mission where the drone takes off, moves to specified waypoints, and returns to the launch location. The *'__future__'* module allows you to use features from future versions of Python in older versions and solves most of the compatibility issues that occur with using Python 2.1. The script uses *argparse* to parse command-line arguments. It expects a connection string (e.g., the UDP address) to connect to the drone. If no connection string is provided, the script starts a *Software in The Loop* (SITL) simulator using *dronekit_sitl*. The connection string is then set according to the SITL instance. The script establishes a connection to the drone using the provided connection string and waits until the vehicle is ready. Waypoints (*point1* and *point2*) are then set, and the drone is commanded to move to these points using *'vehicle.simple_goto()'*. A 30 s delay is added after each destination to allow for the update to be detected. The script also commands the drone to return to the launch (home) location in *Return to Launch (RTL)* mode and then to wait for 5 s. Finally, the script closes the connection to the drone using *'vehicle.close()'* and, if SITL was started, stops the simulator using *'sitl.stop()'*.

To run the script, the following command has to be typed within a terminal where the environment has been activated:

python simple_goto.py --connect udp:127.0.0.1:14552

To make the whole terminal process as automatic, a batch script *'batch_script.bat'* is implemented, which runs all the terminal commands one by one. The script operates as it follows:

- It opens a terminal, initializes *conda* and activates the *pydrone2* environment, and then initializes the *dronekit-sitl* module.
- A 5 s delay is executed before opening the second *conda* terminal and running the *mavproxy.py* script. The delay ensures that the *SITL* is setup successfully, such as the *MAVProxy* connecting to the drone. Then, a further 15 s delay is executed. This ensures that the *MAVLink* is established and that the *Ready to Fly* message is displayed, such that the *'simple_goto.py'* script can be run as well
- After this delay, a 3rd terminal is opened and the *'simple_goto.py'* script is initialized with the connection string

4. Results

Following the setup and initial testing, the virtual drone flies to an initial location, which was set with the following coordinates (−35.361354, 149.165218) and then to a second location set at (−35.363244, 149.168801). The visualized positions of the drone were real-time updated within the *Unreal Engine,* as shown in Figure 7. The same positions can also be reported and visualized within the *Mission Planner* window, which also shows the path traced by the drone (Figure 8).

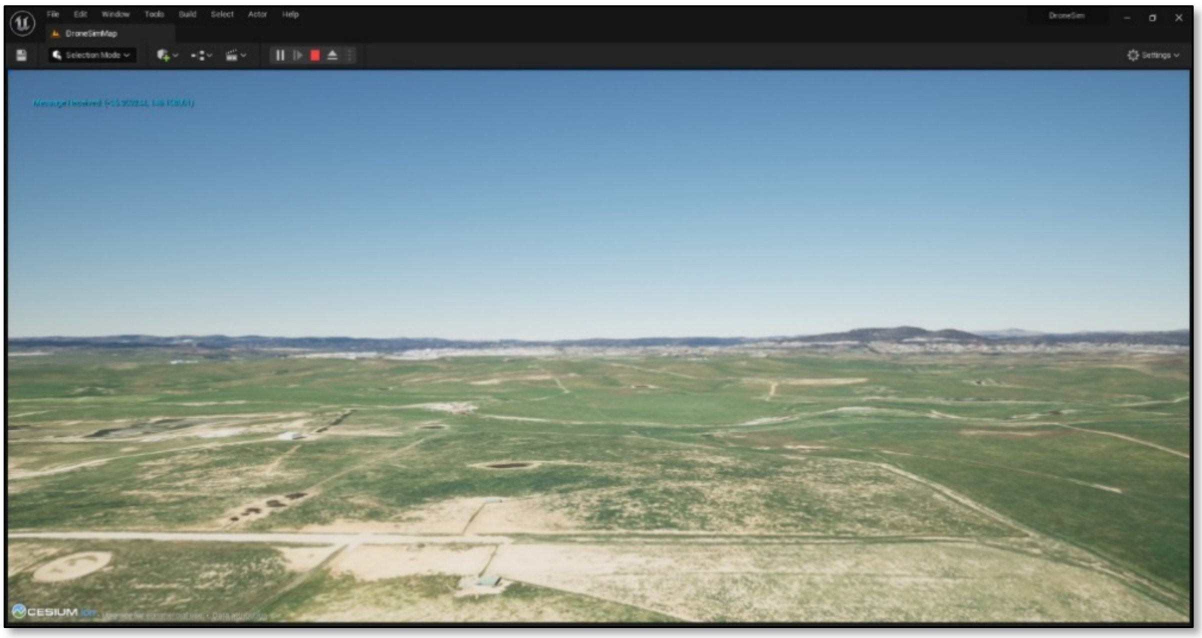


Figure 7. The Unreal Engine Visual.

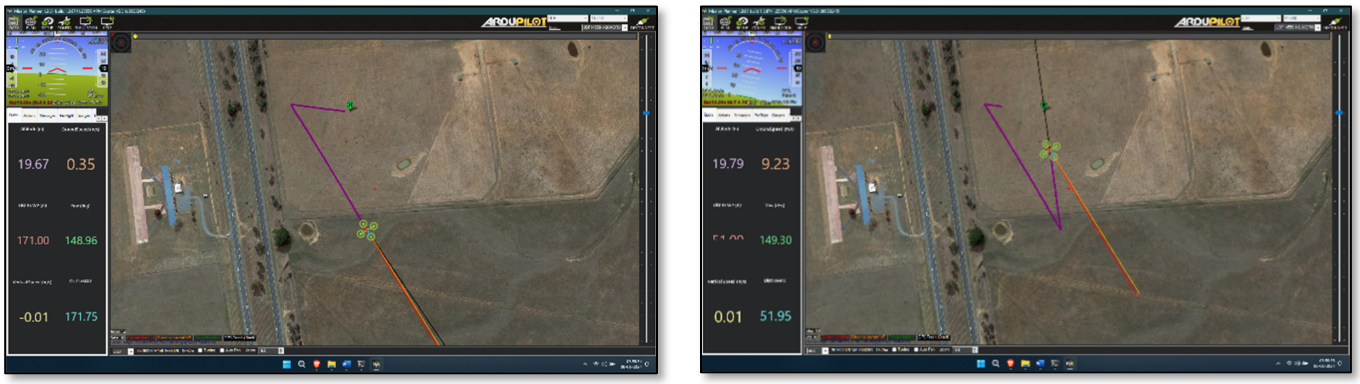


Figure 8. The Flight Path Traced by the Drone: flying towards Waypoint 2 and Returning to Launch on the left and right panels, respectively.

Different locations were tested, including building areas, to see how the Cesium OSM buildings were adding up to the virtualization of the project. Figures 9 and 10 show two other locations where the drone's fly was simulated, namely Denver (USA) and Sydney (Australia), respectively.

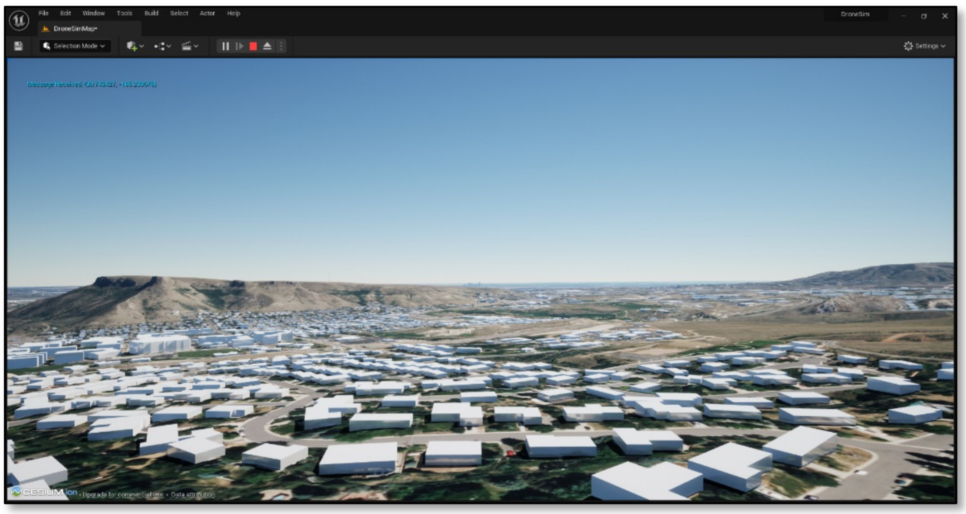


Figure 9. The 3D Visual of Denver (USA).

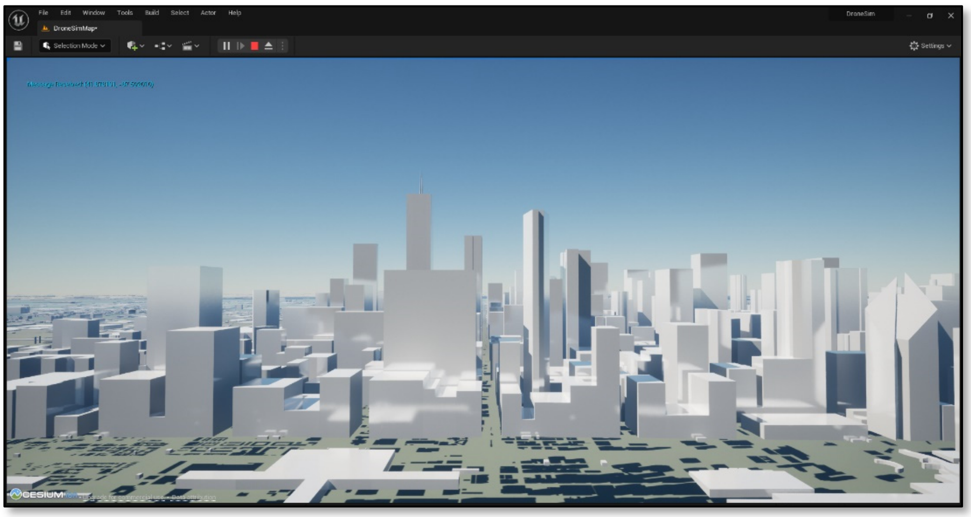


Figure 10. The 3D Visual of Sydney (Australia).

Since the *DynamicPawn* and *OSM buildings* integrate a pre-defined collision logics, a drone can fly through a building, even if it crashes when impacting the walls. Another issue occurring within the current setup is that sometimes the world shows a bright flash. To avoid that, enabling the *Auto Exposure* and *Extend default Luminance range* options in the *Auto Exposure settings* is recommended. These adjustments can be accessed through the *Edit menu*, followed by *Project Settings*, and then navigating to *Engine -> Rendering*. It is also recommended to use the *CesiumSunSky object*, a primary light source for the *Unreal Engine* world, which employs real-world brightness levels surpassing the standard *Unreal Engine Directional Light*. This adjustment ensures a more stable and visually consistent rendering of the environment.

As an alternative approach to the combination of *Cesium* and *Bing data*, *Google's Photogrammetry Data* emerges as a viable option for visualizing textured buildings. Cesium for Unreal offers users a pre-defined photogrammetry *tileset*, and the option of using a locally stored *tileset* in *.json* format provides a flexible choice for developers seeking to enhance the visual fidelity of their scenes. Figure 11 shows an example of using Cesium with Google's photogrammetry data instead of Bing's Aerial imagery, in location (53.390826, −2.892320), which places the virtual drone above Our Institution in the UK, namely *Liverpool Hope University*.

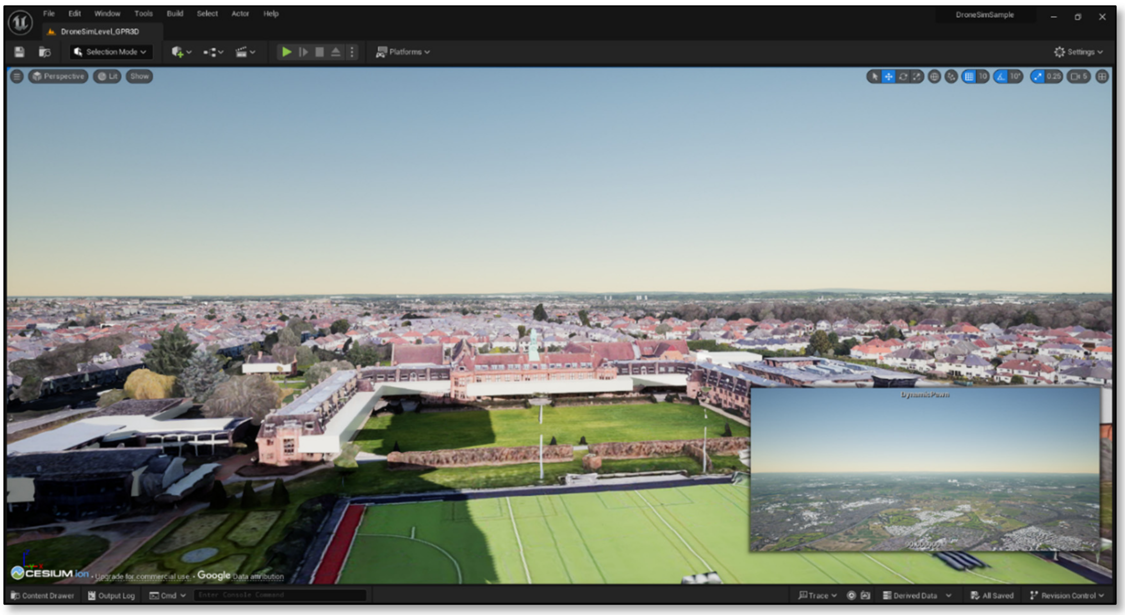
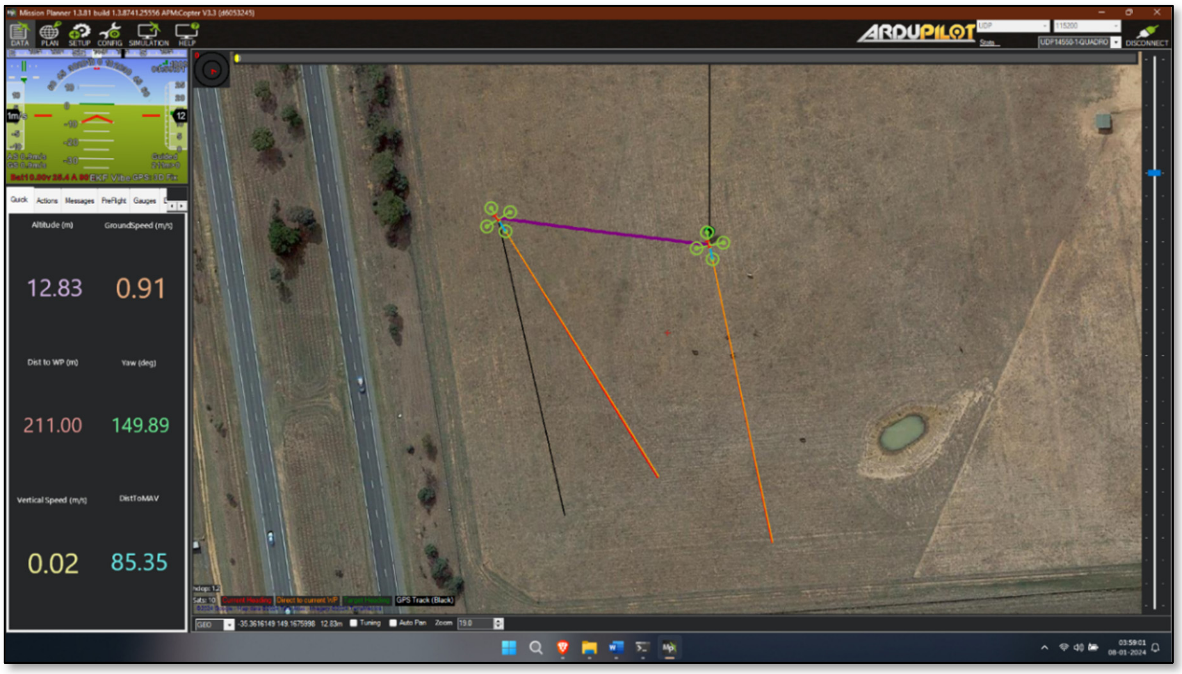


Figure 11. The 3D Visual of Liverpool Hope University, using Google's Photogrammetry Tileset.

Another common issue we encountered in this study is that sometimes the simulated drone would not connect to the Mission Planner, or sometimes two simultaneous drones would be connected to the *Mission Planner*, instead of one to the *Unreal Engine* and one *to the Mission Planner*. This issue can be solved by opening *Mission Planner* and *Unreal*

Engine before running the *Python Script*. This issue is rare, however, it disables the connection with the *Unreal Engine*, freezing the simulation and displaying 2 drones within the Mission Planner, as shown in Figure 12.



Altitude (m) 12.83
GroundSpeed (m/s) 0.91
Dist to WP (m) 211.00
Yaw (deg) 149.89
Vertical Speed (m/s) 0.02
DistToMAV 85.35

Figure 12. The dual drone issue, as it appears in the Mission Planner.

5. Conclusions

The 3D simulation of an SITL copter drone with *Unreal Engine* connected through *MAVLink* and controlled through a *DroneKit-Python script* has been implemented, established, and analyzed. The study utilized various versions of *DroneKit-Python* and its dependencies. Various results were seen from various location experiments and found to have simulated the flight path of the missions. This study can be a prospect for future studies on drone imagery and drone missions. A set of limitations and future developments has been identified. The main limitation is that using Unreal Engine is a heavy resource. A minimum of 4 core CPUs with 2 GB of VRAM, 8 GB of RAM, and 50 GB of ROM storage is required to run the software, let alone the project setup [32]. In this context, the hardware requirement also makes it a less feasible project for researchers.

A realistic simulation of a drone flying over a city can be performed thanks to the proposed system. The Unreal Engine module allows a user to connect a controller to the engine, thereby enabling free-roam possibilities within the engine itself, without the need of a DroneKit-Python script. Path planning, obstacle detection, collision, and environmental influence simulation, such as wind and turbulence, can be established and researched. Moreover, the system could be integrated with other technologies and human-device interfaces, enhancing the user experience and navigation planning, combined with intuitive interactions between the user and the UAV [33–37].

Appendix A

MAVLink facilitates seamless and efficient communication, transmitting critical data, commands, and telemetry information in a standardized format. MAVLink operates on a publish-subscribe architecture, enabling multiple components to exchange information without direct point-to-point connections. This architecture enhances the modularity and extensibility of UAV systems, as new components can be added or removed without necessitating significant changes to the overall communication infrastructure.

The MAVLink protocol presents a distinct advantage owing to its inherent flexibility and support for various transport layers and mediums. Its lightweight structure allows seamless transmission through diverse channels, including Wi-Fi, Ethernet (TCP/IP Networks), and serial telemetry operating at sub-GHz frequencies such as 433 MHz, 868 MHz, or 915 MHz. This adaptability in frequency bands enables extensive communication ranges, which is crucial for remotely controlling unmanned systems. With a maximum data rate reaching 250 kbps, MAVLink proves versatile, though the range is subject to environmental conditions, noise levels, and antenna configurations, typically extending up to 500 m.

A complementary alternative involves leveraging network interfaces, predominantly Wi-Fi or Ethernet, to transmit MAVLink messages through IP Networks. Autopilots supporting the MAVLink protocol often accommodate both UDP and TCP connections at the transport layer. The choice between UDP and TCP hinges on the application's reliability requirements. UDP, being a datagram protocol, prioritizes the need for a connection and lacks a mechanism for ensuring reliable message delivery. However, it serves as a lightweight alternative suitable for real-time and loss-tolerant message streaming. In contrast, being a reliable connection-oriented protocol, TCP incorporates an acknowledgement mechanism, enhancing transfer reliability at the expense of potential congestion and heavier connection management.

A typical MAVLink message is represented in Figure 2 (please find the Figure within the text of the paper). As shown in the Figure, there are 8 important fields, which take between 8 and 263 bytes in length, according to the following legend:

- *STX*—this is the first byte and represents the start of a MAVLink frame. In MAVLink 1.0, STX is equal to 0xFE.
- *LEN*—this is the second byte, representing the length of the message in bytes, and is encoded into 1 byte.
- *SEQ*—the third byte denotes the sequence number of the message, encoded in 1 byte, and takes values from 0 to 255.
- *SYS*—it is the fourth byte representing the system ID, denoting which system is sending the message. In MAVLink 1.0, the system ID 255 is typically allocated for ground control stations.
- *COMP*—this points to the component ID, denoting which component of the system is sending the message. In MAVLink, there are 27 components, but if there is no component in the mentioned system, this ID is not used.
- *MSGID*—the sixth byte refers to the type of message embedded in the payload. For example, MSGID 0 refers to a message of type HEARTBEAT.
- *PAYLOAD*—this is the part of the MAVLink message that contains the information needed to be parsed. It can take a maximum of 255 bytes.
- *CHECKSUM*—the last two bytes of the message, the checksum is mainly used to check the integrity of the message during transmission.

Appendix B

To implement the proposed system, the first stage relies on the installation of the different packages and libraries whose details have been reported in the Appendix. The framework preparation and process refer to the execution of software on a Windows 11 Machine and Python 2. The framework requires the installation of 4 main packages, namely (1) *Python 2*, (2) the *Mission Planner*, (3) the *Unreal Engine* and finally (4) the *Design of a Simulation Level*.

Appendix B.1. Installing Python 2

It is important to know that *Python 2* has been deprecated since January 2020 [31]. Anaconda is recommended for easy installation in this project. From the Anaconda website (https://www.anaconda.com/download, accessed on 15 September 2025), the Windows version of the setup is downloaded. It is noted that the setup is available for Mac OS and Linux as well. Once downloaded, the setup is downloaded and installed. The following command is run next in a terminal to add the Anaconda directory to PATH so that the terminal commands running in the later processes can be automated. This can be done using the command

setx PATH "%PATH%;C:\path\to\your\conda\Scripts"

Once the installation is complete, an Anaconda Prompt is opened from the Windows Start menu. To set up a Python 2.7 environment, the following command is run. In this command, the *env_name* is chosen to be pydrone2.

conda create --name <env_name> python=2.7

When asked for confirmation, press Y, and the installation will commence. Once the environment is created, it can be activated using the '*conda activate <env_name>*' command. The installation can be verified by running 'python --version' after activating the environment. It should return the Python version that is being installed. An alternative to this would be to download Python 2.7 from Python's official website and find the appropriate version package. Once installed, pip should be installed after that by downloading the *get_pip.py* script (reference and source (https://bootstrap.pypa.io/pip/2.7/get-pip.py, accessed on 15 September 2025)) and running this script from the installed *Python 2.x* version. While this process is tiresome and overcomplicated, it is less resource consuming and lightweight than Anaconda installation.

Appendix B.2. Installing the Mission Planner

A package should be installed in order to install the Mission Planner. The planner can be downloaded from the official website (https://ardupilot.org/planner/docs/mission-planner-installation.html, accessed on 15 September 2025) and installed on the Windows machine by following the instructions.

Appendix B.3. Installing Unreal Engine

Epic Games Launcher can be downloaded from the official Epic Games website (https://www.epicgames.com/site/en-US/home accessed on 15 September 2025) and installed. An Epic Games account is setup to be used to sign into the Epic Games Launcher, the Unreal Engine and the Cesium Ion, and to set up the API key later. The launcher is opened, navigating to Epic Marketplace, and downloading Cesium for Unreal, which is available for free. Navigating to the Library tab, a new engine version (in this case, 5.1.1) is added. Once the download location is confirmed, the app starts downloading UE5.1. While this is downloading, Cesium for Unreal plugin can be added to the engine version 5.1. This queues the plugin to be added to UE5.1 after its installation. It takes between 30 min to a few hours to download and install them, depending on the hardware specifications and the network speed.

A Cesium Ion account is set (https://ion.cesium.com/ accessed on 15 September 2025) by connecting the Epic Games account created earlier. A new blueprint-based project named ‘DroneSim’ is created after the installation, with starter content disabled. It takes a while to compile the shaders for the first time. The project should be closed once it opens successfully to install a third-party plugin for UDP connections. An appropriate version of the UDPConnection plugin is downloaded from GitHub (https://github.com/is-centre/udp-ue4-plugin-win64/tree/master accessed on 15 September 2025) and is extracted to a new folder called ‘Plugins’ inside of the Unreal Engine’s project folder. This will add the plugin to the project and automatically enable it. After the migration, the project is opened again. The Cesium for Unreal plugin is activated from the Plugins window, found in *Edit -> Plugins*. An engine restart will be requested to compile shaders once more. The setup takes a few minutes to complete, depending on the hardware.

Appendix B.4. Design of a Simulation Level

A new level is created by navigating to the “File” menu and selecting “New Level” , with the “Empty Level” option chosen to ensure the absence of objects. World Bounds Checks are disabled by accessing World Settings through the “Window” menu, specifically in the “World” -> “Advanced” category. The potential hindrance posed by enabling World Bounds Checks is explained in relation to Pawn movement away from the origin in Unreal Engine, which may conflict with the requirements of Cesium applications. Cesium for Unreal actors are added to the level via the Cesium panel, with the “CesiumSunSky” located in the “Quick Add Basic Actors” section and added to the level, resulting in the appearance of a sky-like gradient in the viewport. At this point, it is recommended to save the current level by clicking the “Save the current level to disk” button or using the shortcut CTRL + S. A name is provided for the level, and consideration is given to setting the new level as the project's default map to ensure automatic opening upon restarting the project. This involves navigating to the “Edit” menu, selecting “Project Settings”, then choosing “Maps & Modes” from the left sidebar. In the “Default Maps” section, both the “Editor Startup Map” and the “Game Default Map” are modified to reflect the newly created level. Returning to the Cesium panel, the user is instructed to click on the “Connect to Cesium ion” button, initiating a pop-up browser window. If not already logged in, the user is prompted to log in using Cesium ion, Epic Games, GitHub, or Google credentials. Following successful login, permission is granted for Cesium for Unreal to access assets. Continuing the setup process, the user is guided to create a default Access Token for the project by clicking on the “Token” button at the top of the Cesium panel. The option to “Create a new token” is selected within the subsequent window, and the token is renamed if desired. The “Create New Project Default Token” button is then pressed. Moving forward, the user is prompted to add “Cesium World Terrain + Bing Maps Aerial Imagery” through the Cesium panel. This results in terrain appearing in the level, and in the Outliner on the right, various Cesium actors are visible, including CesiumSunSky and Cesium World Terrain. The CesiumGeoreference actor in the Outliner is selected, emphasizing its role in determining the level's geographic location, including latitude, longitude, and height. To address the perceived flatness of the city, the addition of detail is recommended through the inclusion of Cesium OSM Buildings. This can be achieved by accessing the Cesium Quick Add panel and adding Cesium OSM Buildings to the level. A Blueprint class with the parent actor as UDPReceiver is created by right-clicking on the content browser, and choosing Blueprint Class. Renaming it to ‘BP_UDP’, it is then dragged into the viewer window to place it in the world. The location of this blueprint does not matter, as blueprints in

Unreal Engine are by default boundless. Then, the level blueprint is opened, where BP_UDP can be referenced. The node connections are shown in Figure 3.

Author Contributions

Conceptualization, M.A.K.R. and O.A.; methodology, M.A.K.R. and O.A.; software, M.A.K.R.; validation, M.A.K.R.; supervision, O.A.; writing—original draft preparation, M.A.K.R. and E.L.S.; final draft preparation, E.L.S.

Ethics Statement

Not applicable.

Informed Consent Statement

Not applicable.

Funding

This research received no external funding.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

1. Faruk TSO. Things to know before you build a drone. In *Building Smart Drones with esp8266 and Arduino: Build Exciting Drones by Leveraging the Capabilities of Arduino and ESP8266*; Packt Publishing Ltd.: Birmingham, UK, 2018; Volume 7.
2. Digital Showrooms Enrich Pagani Hypercar Configuration Experiences. Available online: https://www.unrealengine.com/en-US/spotlights/digital-showrooms-enrich-pagani-hypercar-configuration-experiences (accessed on 4 October 2023).
3. Alkadhim SA. Communicating with Raspberry Pi via MAVLink. *SSRN Electron. J.* **2019**. doi:10.2139/ssrn.3318130.
4. Figueiredo MC, Rossetti RJF, Braga RAM, Reis LP. An Approach to Simulate Autonomous Vehicles in Urban Traffic Scenarios. In Proceedings of the 12th International IEEE Conference on Intelligent Transportation Systems, St. Louis, MO, USA, 4–7 October 2009.
5. Nonami K. Research and Development of Drone and Roadmap to Evolution. *J. Robot. Mechatron.* **2018**, *30*, 322–336.
6. History of ArduPilot. Available online: https://ardupilot.org/planner/docs/common-history-of-ardupilot.html (accessed on 22 May 2023).
7. Qays HM, Jumaa BA, Salman AD. Design and implementation of autonomous quadcopter using SITL Simulator. *Iraqi J. Comput. Commun. Control Syst. Eng.* **2020**, *20*, 1–16. doi:10.33103/uot.ijccce.20.1.1.
8. Haque MR, Muhammad M, Swarnaker D, Arifuzzaman M. Autonomous Quadcopter for Product Home Delivery. In Proceedings of the ICEEICT 2014: International Conference on Electrical Engineering and Information & Communication Technology, Dhaka, Bangladesh, 10–12 April 2014.
9. Gatteschi V, Lamberti F, Paravati G, Sanna A, Demartini C, Lisanti A, et al. New Frontiers of Delivery Services Using Drones: A Prototype System Exploiting a Quadcopter for Autonomous Drug Shipments. In Proceedings of the 2015 IEEE 39th Annual Computer Software and Applications Conference, Taichung, Taiwan, China, 1–5 July 2015.
10. GPS Guided Autonomous Drone. Available online: https://www.scribd.com/document/421107398/Cameron-Roberts-Report (accessed on 15 September 2025).
11. Al-Qaraawi SM, Salman AD, Ibrahim MS. Performance Evaluation of Ad-Hoc Based Aerial Monitoring System. *Period. Eng. Nat. Sci.* **2019**, *7*, 1794.
12. Airmanship Data Capture Technology Flies High with a Defence Innovation Loan. Available online: https://www.gov.uk/government/case-studies/airmanship-data-capture-technology-flies-high-with-a-defence-innovation-loan (accessed on 4 October 2023).
13. Royal Air Force plans operations in XR with Cesium for Unreal. Available online: https://cesium.com/blog/2023/11/21/royal-air-force-plans-operations-in-xr-with-cesium/ (accessed on 4 October 2023).
14. Why Cesium. Available online: https://cesium.com/why-cesium/ (accessed on 28 November 2023).
15. Khan N, Ray RL, Sargani GR, Ihtisham M, Khayyam M, Ismail S. Current progress and future prospects of Agriculture Technology: Gateway to sustainable agriculture. *Sustainability* **2021**, *13*, 4883.
16. Antoinette Project: tools to create the next generation of flight simulators. Available online: https://www.unrealengine.com/en-

US/blog/antoinette-project-tools-to-create-the-next-generation-of-flight-simulators (accessed on 8 September 2023).

17. Anicho O, Charlesworth PB, Baicher GS, Nagar AK. Reinforcement learning versus swarm intelligence for autonomous multi-haps coordination. *SN App. Sci.* **2021**, *3*. doi:10.1007/s42452-021-04658-6.
18. Setting Up a Simulated Vehicle (SITL). Available online: https://dronekit-python.readthedocs.io/en/latest/develop/sitl_setup.html (accessed on 1 September 2023).
19. Message Signing. Available online: https://mavlink.io/en/guide/message_signing.html (accessed on 16 October 2023)
20. Telemetry Forwarding. Available online: https://ardupilot.org/mavproxy/docs/getting_started/forwarding.html#mavproxy-forwarding (accessed on 20 November 2023).
21. Installing DroneKit. Available online: https://dronekit-python.readthedocs.io/en/latest/develop/installation.html (accessed on 20 November 2023).
22. Welcome to DroneKit-Python's Documentation! Available online: https://dronekit-python.readthedocs.io/en/latest/ (accessed on 20 November 2023).
23. The Most Powerful Real-Time 3D Creation Tool. Available online: https://www.unrealengine.com/en-US (accessed on 20 November 2023).
24. Nanite Virtualized Geometry. Available online: https://docs.unrealengine.com/5.0/en-US/nanite-virtualized-geometry-in-unreal-engine/ (accessed on 29 November 2023).
25. World Partition. Available online: https://docs.unrealengine.com/5.0/en-US/world-partition-in-unreal-engine/ (accessed on 29 November 2023).
26. Lumen Global Illumination and Reflections. Available online: https://docs.unrealengine.com/5.0/en-US/lumen-global-illumination-and-reflections-in-unreal-engine/ (accessed on 29 November 2023).
27. The Platform for 3D geospatial. Available online: https://cesium.com/ (accessed on 28 November 2023).
28. Cesium for Unreal. Available online: https://cesium.com/platform/cesium-for-unreal/ (accessed on 28 November 2023).
29. About Anaconda. Available online: https://www.anaconda.com/about-us (accessed on 8 January 2024).
30. Mission Planner Overview. Available online: https://ardupilot.org/planner/docs/mission-planner-overview.html (accessed on 8 January 2024).
31. Sunsetting Python 2. Available online: https://www.python.org/doc/sunset-python-2/ (accessed on 1 July 2023).
32. Hardware and Software Specifications for Unreal Engine. Available online: https://docs.unrealengine.com/5.3/en-US/hardware-and-software-specifications-for-unreal-engine/ (accessed on 8 November 2023).
33. De Guzman CJP, Chua A, Chu T, Secco EL. Evolutionary Algorithm-Based Energy-Aware Path Planning with a Quadrotor for Warehouse Inventory Management. *High Tech. Innov. J.* **2023**, *4*. doi:10.28991/HIJ-2023-04-04-01.
34. Latif B, Buckley N, Secco EL. Hand Gesture & Human-Drone Interaction. *Intell. Syst. Conf.* **2022**, *3*, 299–308.
35. Chu T, Chua A, Secco EL. A Study on Neuro Fuzzy Algorithm Implementation on BCI-UAV Control Systems, *ASEAN Eng. J.* **2022**, *12*, 75–81.
36. Manolescu VD, AlZu'bi H, Secco EL. Interactive Conversational AI with IoT Devices for Enhanced Human-Robot Interaction. *J. Intell. Commun.* **2025**. doi: 10.54963/jic.v3i1.317.
37. Chu TSC, Chua A, Secco EL. Performance Analysis of a Neuro Fuzzy Algorithm in Human Centered & Non-Invasive BCI. In *Proceedings of Sixth International Congress on Information and Communication Technology*; Springer: Singapore, 2021.