Development of a 3D visualization interface for virtualized UAVs

Chloé Riviere, Jamie Wubben, Carlos Calafate, Tahiry Razafindralambo

▶ To cite this version:

HAL Id: hal-04531959

https://hal.univ-reunion.fr/hal-04531959

Submitted on 11 Apr 2024

Development of a 3D visualization interface for virtualized UAVs

Chloé Rivière[1], Jamie Wubben[2][0000−0001−8121−995X], Carlos T. Calafate[2][0000−0001−5729−3041], and Tahiry Razafindralambo[1][0000−0002−5239−7498]

[1] Reunion Island and Indian Ocean Engineering School (ESIROI), University of Reunion Island, Reunion Island, France,
[2] Computer Engineering Department (DISCA), Universitat Politecnica de València, 46022 Valencia, Spain
chloe.riviere@esiroi.re, jwubben@disca.upv.es, calafate@disca.upv.es, tahiry.razafindralambo@univ-reunion.fr

Abstract. Nowadays, Unnamed Aerial Vehicles (UAVs) are used in many different fields, ranging from agriculture and entertainment to parcel delivery, among others. For several of these tasks, the UAVs are programmed to follow a specific path, as defined in their flight missions. In addition, as more sophisticated solutions begin to be adopted, new protocols should be developed to handle possible collisions among UAVs, as well as to create UAV swarms. However, directly testing new protocols on UAVs can be hazardous and time consuming. Therefore, many investigators first perform simulations. Although different UAV simulators exists, not all of them offer a 3D rendering of the UAVs in the target flight environment. Hence, in this work, we present a real-time 3D visualization interface that can be easily coupled to any simulator. In this way, we offer developers a powerful way to validate their solutions and make in-depth analysis.

Keywords: UAV simulator · 3D visualization · ArduSim · Unity.

1 Introduction

Unnamed Aerial Vehicles (UAVs), more commonly known as drones, have experienced a huge adoption in recent years by both particulars and professionals, for different reasons, and in different fields. According to the field and their intended use, the UAVs need specific algorithms to, e.g., avoid collisions, perform surveillance on a specific area, etc.

The process of testing these new algorithms is complex and prone to frequent failures. As such, attempting to perform tests on real UAVs is a time-consuming and costly endeavor. For this reason, especially in the initial stages, simulation is preferred. Different UAV simulation tools have been made available in recent years, such as AirSim [10], which allows controlling UAVs and many other vehicles in a 3D environment, or tools which are specific for UAVs like Gazebo [6]. In

particular, our focus in this work in on ArduSim [3], an UAV emulation tool developed to model interactions between UAVs based on wireless communications. However, ArduSim lacks a proper 3D visualization interface, complicating the proper validation of novel protocols under development. Hence, in this work, we address this challenge by developing a new component for ArduSim: a new 3D visualization interface. More specifically, the visualization interface allows seeing if, for example, a UAV swarm has the formation wanted, and if their coordinates on the map are correct.

We created this interface to extend the capabilities of our simulator ArduSim. Nevertheless, there are more simulators that do not provide a 3D interface. Hence, we created our interface in such a way that it is open, meaning that it can be easily connected to other simulators as well. Our new 3D tool is able to (i) render multiple UAVs at the same time, (ii) render buildings, and the terrain, (iii) and work in real-time.

The remainder of this paper is organized as follows: the following section will present the most relevant works in the field of UAV simulation, with greater emphasis on the OpenStreetMap project. Afterward, in section 3, we explain how our application is developed, we go over its features, and detail how it should be used. Then, in section 4, we perform various experiments in order to test the limits of our tool. Finally, Section section 5 concludes the paper and refers to future works.

2   Related Work

With all the interest around UAV topics in recent years, a plethora of protocols have been developed using UAV simulators. Currently, we can find several UAV simulators, and some of them have a 3D or 2D visualization built-in. For instance, InDrone [1] is a UAV simulator that has only visualizes the UAVs in 2D. This requires less computational power and, although it is useful in many cases, sometimes 3D rendering is necessary in order to fully analyze the behaviour of the UAV flight paths. For instance, when simulating collision avoidance, take-off algorithms, landing, etc.

Other well-known UAV simulators do include a 3D visualization. For instance, FlightGear [8], and Gazebo [6]. However, in their simulators, the 3D visualization is completely built-in. This is a problem for two reasons. First, one must always use the 3D user interface, which makes performing many tests more-time consuming. Secondly, it does not allow users to easily integrate different simulators at the same time. This might become important in the future when we want to simulate UAVs, rovers, and boats at the same time. Hence, we propose a 3D visualization interface that can be easily connected to any simulator.

A common feature in many simulators is the choice and generation of the map/environment. Often the map is build using real building information (e.g. OpenstreetMap) and the ground is artificially created by a script [4].

There are different libraries that allow for a easy creation of the map. As seen in the table 1, the most convenient choice for the OpenStreetMap was the

Table 1: Map Information sources.

Name	License	Library for Unity	References
OpenIndoor	Free	No	[7]
Streets GL	Free	No	[2]
Esri OpenStreetMap 3D Scene Layers	Proprietary	No	[9]
Map SDK Mapbox	Free	Yes	[5]

Map SDK for Unity from MapBox [5], which enables a direct implementation, and provides various examples of using the map for the Unity game engine.

3 Application development and design

In this section, we will explain how we developed our 3D interface. We will start by the architecture, then go over the different features our 3D visualization tool offers, and finally explain how a user can interact with the tool.

3.1 Architecture

Our entire tool was created using the Unity game engine. This allowed us to develop the tool faster, and also to make use of many libraries that are compatible with all major platforms (Windows, Linux, and macOS). As stated before, the intention of this visualization interface is to only visualize the UAVs and their environment. The actual simulation of the drones is performed by ArduSim [3]. Hence, this visualization tool requires some data from ArduSim in order to accurately display the UAVs. It is worth reiterating that our goal is that this tool can be used for other simulators as well. In order to facilitate this, the data that this visualization tool requires is received via web sockets. Notice that communication through web sockets is both straightforward and universally used.

In order for our tool to work, messages must be sent (using JSON) to this web socket. The messages only have to contain the x, y, and z coordinates (in UTM format), and a unique number in order to identify the drone. From the moment our visualization tool receives the first message, it will create a drone instance and place it on the desired location (based on the UTM coordinates). If new data is received, the drone's location will be updated. It is obvious that, in order for the drone to move smoothly, the simulator should send messages frequently.
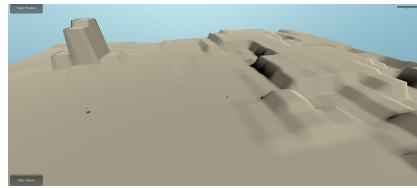
Furthermore, our visualization tool supports two different types of maps (and the option to disable the map completely). The first type of map that we include in our tool uses information from OpenStreetMap in order to accurately display buildings, streets, etc. In order to do so, we relied on the map SDK (software development kit) from MapBox. In order to initialize the map, we use the first UTM position received as the center of the virtual world. Then, we render the world around this first drone. The specific area that will be rendered depends on

the position of the camera and the field of view of the camera. When the UAV moves, the map will automatically update and render new parts of the virtual world when needed. In Figure 1a, we show what this virtual world looks like.

Besides this type of map, we also have the option to use information from Digital Elevation Maps (DEMs). These maps consist of information about the terrain level (sometimes they also include buildings). The advantage of these types of files is that (i) they are readily available, and (ii) we can accurately display the terrain level. This might be important when researchers are simulating a drone flight in the mountains. Since these files only provide the elevation level (above sea level) for specific coordinates, some processing was needed. First of all, at the beginning of the experiment, this file needs to be read, and the elevation data needs to be retrieved and stored in local memory. This processing takes some time, but it only has to be done once. Afterward, we create our virtual world by rendering multiple triangles. Since we have the elevation information of many points, we are able to create a smooth surface. In Figure 1b, we provide a visual representation.
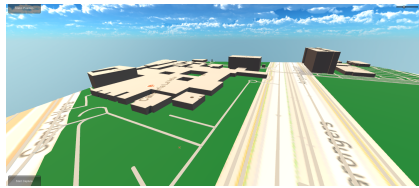
(a) Map view with MapBox.   (b) Map view with DEM.

Fig. 1: The two different maps that are available.

3.2   Features

Besides choosing between the two maps, our visualization tool also includes a few extra features. First of all we have different cameras so that the user analyzes the UAVs from different perspectives. We include four different types of views: a static view, a "following view", a focused view, and a view from a camera placed on a UAV. In Figure 2 we show the different type of views. As one can image, the static view remains at one place, the following view follows the UAVs from a distance, the focused view is a close up view for one specific UAV, and finally the drone view is a first person view taken from a camera attached to the UAV. This drone view gives a more realistic aspect to the interface by allowing the user to have the view of the UAV camera as in a real life use.

To further explain where the cameras are placed for each specific view, we include Figure 3. In this figure, one can clearly see the different camera views.

Since there are different types of views, we can easily lose sight of one UAV. For this reason, we included small flags on top of the UAVs (including their ID).

(a) Static view.

(b) Following view.

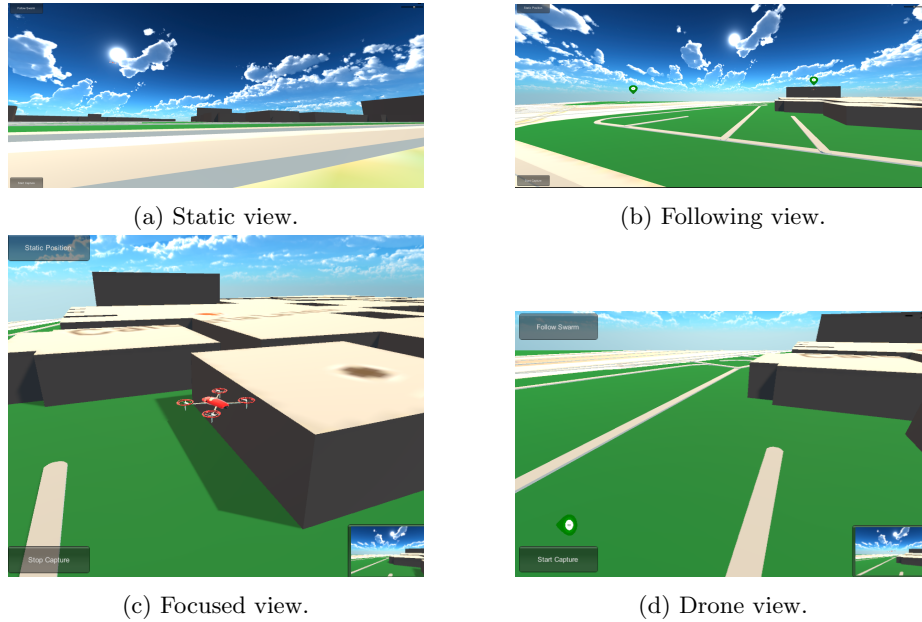(c) Focused view.

(d) Drone view.

Fig. 2: The different views available in our visualization tool.

This flag is placed differently depending on one criterion: whether the UAVs are in the camera's field of view or not. For UAVs in the camera's field of view, the flag is displayed when the distance between the camera and the UAV in each axis (x, y, z) exceeds a fixed distance. To place the flag correctly on the screen (above the UAV), the UAV's position in the 3D world is converted to a 2D position on the screen. This conversion is made possible by a function of the camera in Unity. For UAVs that remain outside the camera's field of view, the corresponding flags are placed on the right or left side of the screen. This would tell users in which direction they need to turn the camera to see the UAVs. Such direction in determined by first calculating the direction between the camera and the UAV; based on this direction, we can calculate the angle between the camera and the UAV. To this angle we add the angle of rotation of the camera; thus, if this angle is below or above a certain threshold, we know in which direction the camera must be turned, and on which side of the screen we need to place the flag.

Finally, we also include an option to record a video. This will provide the user the ability to review the simulation later on. The user has 4 states regarding the recording: not recording, currently recording, paused recording, and finished recording. At the end of the recording, if the option to send video has not been disabled, the recorded video will be sent to a dedicated server via a web socket. The recording of the simulation is done using a library (Rock VR) created for Unity. This library allows recording the field of view of any designated camera.
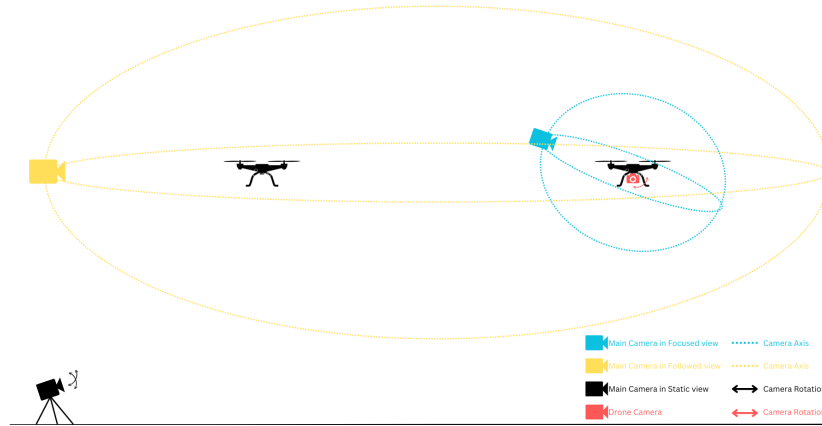
Fig. 3: Static view and camera rotation features.

Despite the fact that the library only records the flow of a specific camera and the UI elements, this library has been chosen due to the fact that it is free, and works out of the Unity environment, in contrast to the Unity editor recording which only works in the Unity environment. This in an important element, especially when users merely intend to run the executable, and not work within the Unity environment.

3.3   User interface

The simulator provides many opportunities for user interaction. These interactions allow the user to tailor the interface to their needs. Many of these interactions are optional and intuitive to use (click on a button, move the mouse). We can find some of these interactions in Figure 4.

Before starting the simulation, the user has the opportunity to make some configurations. These configurations will not be editable during the simulation. In these pre-configurations we will find the choice of map type, the scale of the map, the choice of the camera path for saving the recording, the possibility of sending the video to a server or not, the rendering quality of the simulation, and the possibility of hiding or showing the UAV flags. The user can leave the focused view with a double click/tap, or by pressing the space bar, which will set the main camera to the selected view type and "destroy" the UAV camera.

4   Evaluation

In this section, we proceed to validate the developed application by performing different types of tests to determine how different loads/features affect the CPU and RAM use. Notice that both CPU and RAM are critical resources, being intimately related to an adequate execution of the application. Hence, these
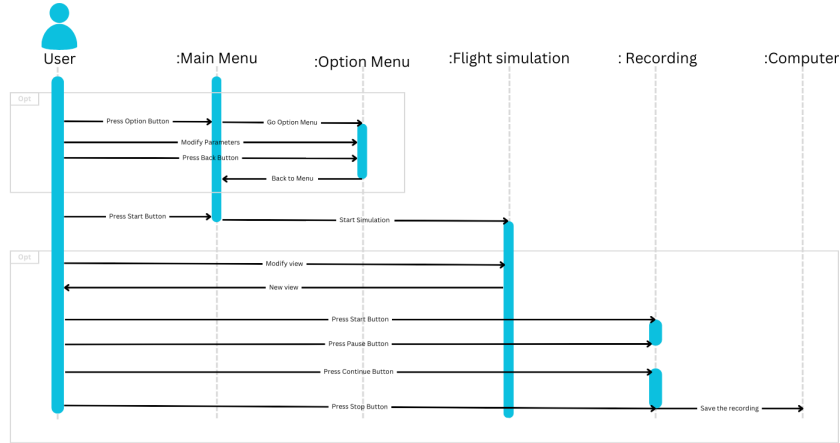
Fig. 4: User interaction options.

tests will have a series of experiments with different numbers of UAVs, while other tests will vary the different map options, the rendering quality, and the recording option. Each test was repeated on a laptop (see specifications, 2). In order to have accurate results, each experiment was repeated three times with a simulation of 2 UAVs for 30 seconds (except when stated otherwise). The CPU and RAM usage was collected every 5 seconds.

Table 2: Hardware used for experiments.

Feature	Value	Feature	Value
Processor	Intel(R) Core(TM) i5-8265U	Speed	1.60GHz (1.80 GHz max)
Cores	4	Hyper-Threading	Yes
RAM	8 GB	HHDD	237 GB
GPU	Intel UHD Graphics 620	Screen resolution	1920 x 1080
OS	Windows 10 Professional		

4.1 Impact of the rendering quality

For all the different qualities tested, the same configuration was used: Mapbox map, following view, without recording, and with the same UAV scheme. This test shows whether the quality of the video selected affects the CPU or the RAM during the simulation. Figure 5 illustrates the results achieved with 6 different rendering qualities:

– VL (Very Low)	– H (High)
– L (Low)	– VH (Very High)
– M (Medium)	– U (Ultra)

These different qualities affect the frame rate, anti-aliasing (which allows for smooth edges), shadow rendering, and more.

From figure 5a we find that the rendering quality of the interface has a major impact on the use of CPU; we can specifically see the gap for the first quality level (VL) to the fourth (H). Concerning the last three, High, Very High and Ultra, they present a similar CPU usage. From figure 5b we can also see that the different qualities do not affect RAM usage.
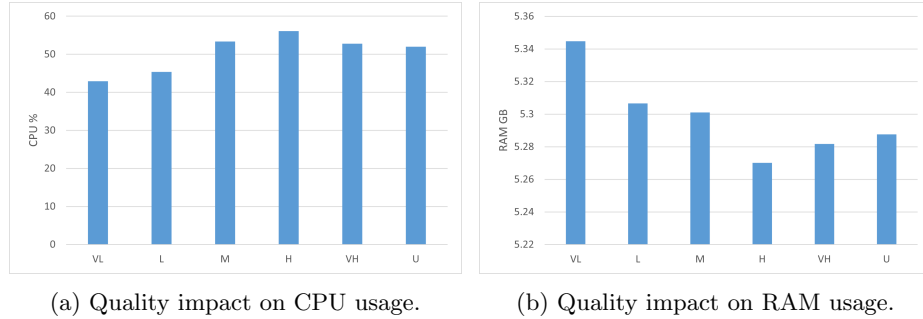

(a) Quality impact on CPU usage.   (b) Quality impact on RAM usage.

Fig. 5: Impact of video quality on CPU and RAM.

4.2   Impact of the map

The tests detailed below were carried out by fixing the configuration as follows: same rendering quality, following view, and no recording. This way, only the map itself is changed. Our purpose is to determine if the map choice has an effect on the CPU or the RAM usage during the simulation.

Figure 6a shows that the different maps do not have a significant impact on the use of CPU; in contrast, in Figure 6b we can see an increase of the RAM usage for the DEM map due to the reading of the DEM file (needed for the ground elevation data).

4.3   Impact of recording

The tests that follow were carried out by fixing the configuration as follows: same rendering quality, and following view. This way, only the impact of the recording and the sending would be measured. Our purpose is to determine if the recording and the sending of the video have an effect on the CPU or the RAM usage during the simulation.

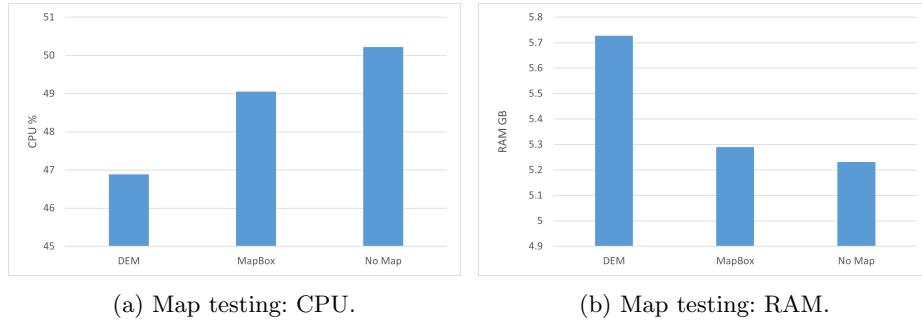(a) Map testing: CPU.   (b) Map testing: RAM.

Fig. 6: Impact of maps' usage on CPU and RAM.

Figure 7a shows that the recording and sending are tasks that significantly increase the CPU usage. Similarly, in figure 7b we can notice the increase of the RAM usage according to the space needed during the recording for the data of the video and for the sending of the video, which required the reading of the video data for the conversion of the video while sending.
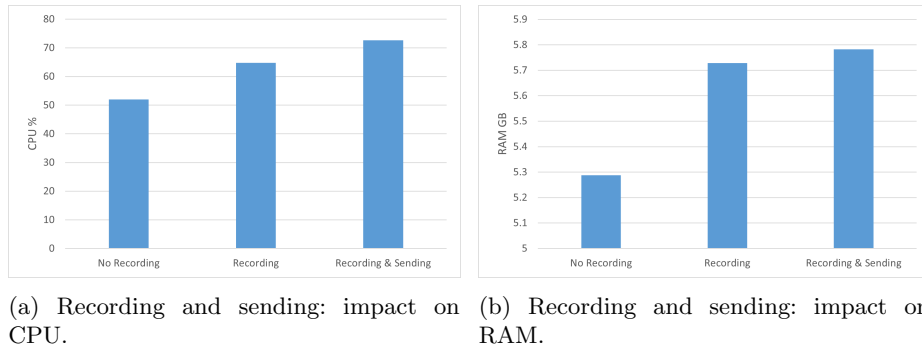

(a) Recording and sending: impact on   (b) Recording and sending: impact on
CPU.   RAM.

Fig. 7: Impact of recording and sending on CPU and RAM.

4.4   Impact of the number of UAVs

The tests that follow were carried out to determine how many UAVs we can visualize in real time using our interface. Hence, we have tested increasing the number of UAVs in the experiment, while testing with different rendering qualities, and again measuring both CPU and RAM usage. The *following* view was chosen, and we also made one test with recording on.

Figure 8a shows that the CPU usage reaches its maximum with about 15 UAVs, independently of the rendering quality; yet, the movement of the camera for each view still remains smooth with up to 25 UAVs. Above 25 UAVs, the

simulation is not smooth anymore, failing to represent mobility realistically. In terms of RAM usage, Figure 8b demonstrates that the number of UAVs is not a factor having a significant impact on simulation. Hence, in most cases, only CPU limits should be accounted for.

(a) CPU use for different UAVs.
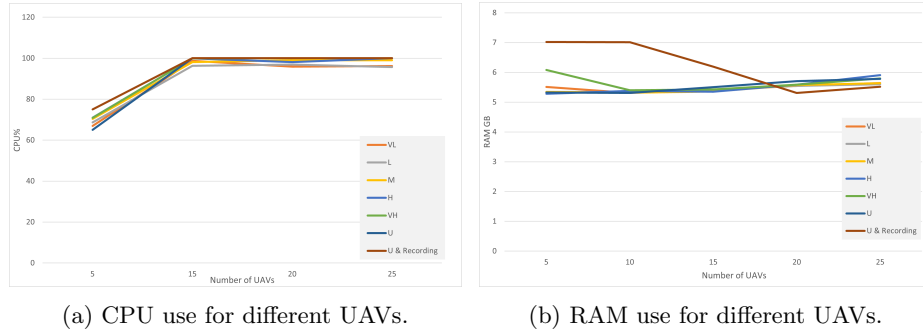

(b) RAM use for different UAVs.

Fig. 8: Impact of increasing the number of UAVs on CPU and RAM.

In this test we have also noticed that, if we enable recording, the limit of UAVs to be used is just under 15 UAVs. This lower limit is associated to the high needs in terms of CPU usage associated to the recording process. Above 15 UAVs, the CPU can no longer handle adequately both the real-time UAV simulation and the recording process itself. This can also be seen in figure 8b: above 15 UAVs, the use of RAM decreases when recording is enabled because the interface can no longer record adequately.

5   Conclusion and future work

In this paper, we have presented a new 3D UAV visualization interface that can be easily used by different UAV simulators. In this way, good UAV simulators that do not have the capability to render UAVs in 3D can now use our tool to do so. Additionally, this allows for the rendering of the UAV to be executed on a different machine. With the use of this 3D UAV visualization interface, investigators will now be able to analyze their simulations in a better way. With our visualization tool, the surrounding (buildings, streets, etc.) of the UAVs is automatically built. Furthermore, we have different cameras in order to inspect the UAV from different angles. One of these cameras is a first person view camera, which provides a realistic view of what the UAV sees. Furthermore, videos can be recorded from every camera.

We performed various tests in order to validate our visualization tool. Results show that our 3D interface allows visualizing in real-time up to 25 UAVs with the best rendering quality; tests also highlighted the huge consumption of the CPU and the RAM during the recording, which limited the overall number of UAVs to 15.

To improve the current interface, it would be interesting to test other libraries for the recording to find a way to reduce the use of CPU and RAM associated to this process. Also, to have a more realistic simulation, it will also be interesting to have different types of UAVs with different weights and characteristics.

Acknowledgement

References

1. Eiris, R., Albeaino, G., Gheisari, M., Benda, W., Faris, R.: Indrone: a 2d-based drone flight behavior visualization platform for indoor building inspection. Smart and Sustainable Built Environment **10**(3), 438–456 (2021)
2. Esri: Streetgl map online, https://streets.gl
3. Fabra, F., Calafate, C.T., Cano, J.C., Manzoni, P.: Ardusim: Accurate and real-time multicopter simulation. Simulation Modelling Practice and Theory **87**, 170–190 (2018). https://doi.org/https://doi.org/10.1016/j.simpat.2018.06.009, https://www.sciencedirect.com/science/article/pii/S1569190X18300893
4. Hong, W.T., Lee, P.S.: Mesh based construction of flat-top partition of unity functions. Applied Mathematics and Computation **219**(16), 8687–8704 (2013)
5. Laksono, D., Aditya, T.: Utilizing a game engine for interactive 3d topographic data visualization. ISPRS International Journal of Geo-Information **8**(8), 361 (2019)
6. Meyer, J., Sendobry, A., Kohlbrecher, S., Klingauf, U., Von Stryk, O.: Comprehensive simulation of quadrotor uavs using ros and gazebo. In: Simulation, Modeling, and Programming for Autonomous Robots: Third International Conference, SIMPAR 2012, Tsukuba, Japan, November 5-8, 2012. Proceedings 3. pp. 400–411. Springer (2012)
7. OpenIndoor: Openindoor github (2022), https://github.com/open-indoor
8. Perry, A.R.: The flightgear flight simulator. In: Proceedings of the USENIX Annual Technical Conference. vol. 686, pp. 1–12 (2004)
9. Piccione, M., Fuhrmann, S.: Using esri cityengine (2016)
10. Shah, S., Dey, D., Lovett, C., Kapoor, A.: Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In: Field and Service Robotics: Results of the 11th International Conference. pp. 621–635. Springer (2018)