

Review

A Survey on Open-Source Simulation Platforms for Multi-Copter UAV Swarms

Ziming Chen ¹, Jinjin Yan ^{1,*}, Bing Ma ¹, Kegong Shi ¹, Qiang Yu ¹ and Weijie Yuan ²

¹ Qingdao Innovation and Development Center, Harbin Engineering University, Qingdao 266400, China

² Department of Electrical and Electronic Engineering, Southern University of Science and Technology, Shenzhen 518055, China

* Correspondence: jinjin.yan@hrbeu.edu.cn; Tel.: +86-153-2115-5969

Abstract: Simulation platforms are critical and indispensable tools for application developments of unmanned aerial vehicles (UAVs) because the UAVs are generally costly, have certain requirements for the test environment, and need professional licensed operators. Thus, developers prefer (or have) to test their applications on simulation platforms before implementing them on real machines. In the past decades, a considerable number of simulation platforms for robots have been developed, which brings convenience to developers, but also makes them hard to choose a proper one as they are not always familiar with all the features of platforms. To alleviate this dilemma, this paper provides a survey of open-source simulation platforms and employs the simulation of a multi-copter UAV swarm as an example. The survey covers seven widely used simulators, including Webots, Gazebo, CoppeliaSim, ARGoS, MRDS, MORSE, and USARSim. The paper outlines the requirements for multi-copter UAV swarms and shows how to select an appropriate platform. Additionally, the paper presents a case study of a UAV swarm based on Webots. This research will be beneficial to researchers, developers, educators, and engineers who seek suitable simulation platforms for application development, (not only multi-copter UAV swarms but also other types of robots), which further helps them to save expenses for testing, and speed up development progress.

Keywords: open-source; simulation platform; multi-copter UAV; swarm; modeling



Citation: Chen, Z.; Yan, J.; Ma, B.; Shi, K.; Yu, Q.; Yuan, W. A Survey on Open-Source Simulation Platforms for Multi-Copter UAV Swarms. *Robotics* **2023**, *12*, 53. <https://doi.org/10.3390/robotics12020053>

Academic Editors: Goldie Nejat and Beno Benhabib

Received: 22 February 2023

Revised: 19 March 2023

Accepted: 26 March 2023

Published: 1 April 2023



Copyright: © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

The unmanned aerial vehicle (UAV), also known as a drone, was originally developed for military missions in the twentieth century because it has no human pilot, crew, or passengers on board and can be used to conduct dull, dirty, or dangerous assignments [1]. Nowadays, the UAV has been widely utilized in various fields, e.g., aerial photography [2], emergency rescue [3], item deliveries [4], policing and surveillance [5], science [6], inspections [7], and racing [8]. In recent years, applications of UAV are gradually shifting from single to swarm, because compared with single UAVs, a swarm has obvious advantages, such as flexibility, governance, control, scalability, extensibility, interoperability, integration, and high fault tolerance [9–11].

Before implementing applications or algorithms on real machines, testing them on simulators is a necessary step, because such a practice can aid in minimizing the potential risks and expenses involved in real-world UAV testing and improving safety and productivity. For instance, it is critical to consider the costs and risks in a real environment when hundreds or even thousands of UAVs will be involved. As we all know, a UAV is generally costly [12], have certain requirements for the test environment [13], and need professional licensed operators [14]. Due to the uncertainty of the applications or algorithms, UAV could collide with each other, and even crash, which will bring in big security threat to the experimenters once the test fails [15]. More importantly, it would be challenging to find a suitable area for testing since there are more and more no-fly zones for UAVs based on the air traffic control [16].

In the past decades, a considerable number of robot simulation platforms have been developed. They have many aspects, such as built-in model libraries, physics engines, and programming languages [17]. It makes it hard for developers to choose a proper simulation platform as it is tedious to fully understand all the functions of various platforms or make a choice by testing them one by one. For instance, some developers employ MATLAB or other tool-kits for verification of a UAV platform [18,19]. Such platforms have poor visualization performance; UAVs are only regarded as particles in the simulation process; no attitude information of a UAV, no environment interaction, and no physics engines are involved. Thus, they cannot simulate the friction, airflow, and other factors that should be considered in the real environments, which inevitably reduces the accuracy of simulation [20]. To put it differently, selecting a simulation platform that is suitable for the actual utilization of UAVs is crucial for users. This enables them to test and verify the performances of UAVs and adapt to various environments/conditions. However, for the time being, users are confused about choosing a simulation platform to ensure that it caters to their needs and delivers reliable solutions for their UAV applications, even though they could evaluate their objectives and requirements.

Currently, there are some academic surveys on simulation platforms [21–23], but these studies are focused on the introduction of the parameters of platforms, such as sensors. Such information can be easily collected from the websites. In other words, these studies are similar to a summary of the characteristics of the simulators, rather than a real survey. Because, on the one hand, no analysis and comparison are conducted between different simulation platforms; and on another hand, no investigations are presented on the requirements of UAV swarm simulation.

To alleviate the above-mentioned dilemma, this paper presents a survey on commonly used open-source simulation platforms for multi-copter UAV swarms, including Webots [24], Gazebo [25], CoppeliaSim [26], ARGoS [27], MRDS [28], MORSE [21], and USARSim [29]. It should be noted that we suppose the platforms that can be tried or used for free for non-commercial purposes (e.g., education use) are also open-source. The contribution of this survey is that it presents detailed comparisons of the simulation platforms in the field of UAVs swarm, rather than just listing parameters of simulators, which can directly help researchers, developers, educators, and engineers in seeking an appropriate simulation platform(s) for application developments, especially for the UAVs swarm. The remainder of this paper is organized as follows. The next section introduces and compares the seven open-source simulators. Section 2 describes the concerns for selecting proper simulation platforms for multi-copter UAVs swarm. Section 3 demonstrates a use case based on the selected platform—Webots, and finally presents the experimental results. Section 4 concludes the whole work.

2. Seven Widely Used Open-Source Platforms

The history of simulation platforms can be traced back to the 1980s. Since then, computer-aided design (CAD) software packages became capable of robot simulations [17], such as XAnimate [30], SAMMIE [31], and GraspIt! [32]. After a decade, emulation platforms are showing increasingly diverse capabilities. For instance, the Player Project [33], the modular robot simulation platform for humanoid robots [34], PhysX physics engine, and OpenGL-based simulation platform [35–37]. In the last few years, a growing number of open-source simulation platforms have been released, including Simbad [38], Open Dynamics Engine (ODE) [39], and Carnegie Mellon Robot Navigation Toolkit (Carmen) [40]. These open-source platforms offer several advantages over earlier platforms developed by research institutions, including built-in robotic libraries, sophisticated physical environment simulation, and user manual support. These features make open-source simulation platforms more accessible to researchers and developers worldwide.

In this paper, we have chosen to focus on seven widely used open-source simulation platforms, including Webots, Gazebo, CoppeliaSim, ARGoS, MRDS, MORSE, and USAR-

Sim. The motivation for this selection is that we found that these seven platforms have a large user base, and can satisfy the testing requirements of most researchers and developers.

2.1. Webots

Webots, developed by Cyberbotics Company [24], is an open-source simulation platform, which can simulate three-dimensional agents (models). This platform is highly compatible with the mainstream operation systems, such as Windows, Linux, MAC OS. It can support C, C++, Java, Python, MATLAB and other programming languages [41]. The simulation of a UAV in a jungle by Webots is shown in Figure 1.

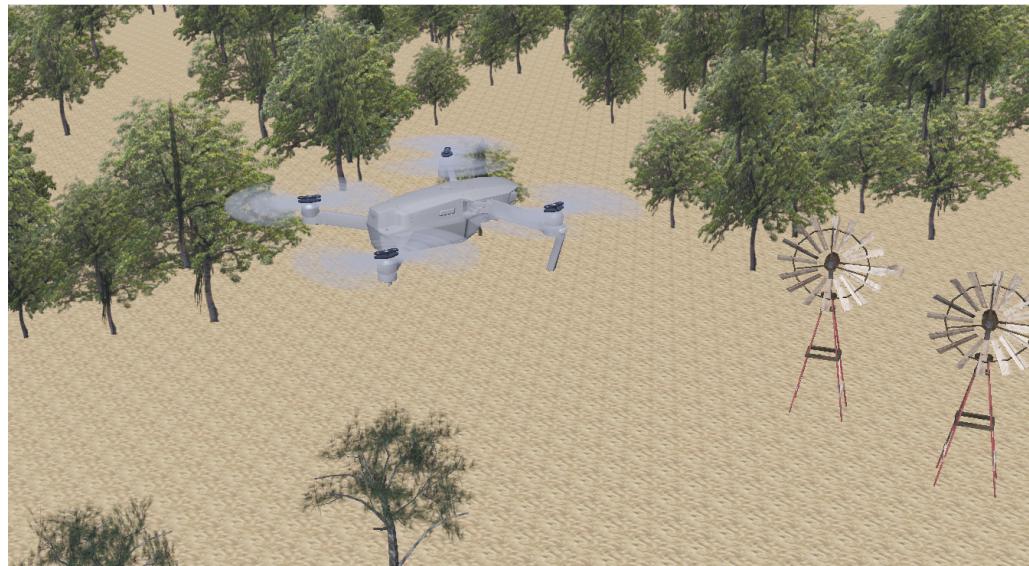


Figure 1. Simulation of a UAV by Webots.

Webots includes a variety of built-in agents (models) such as wheeled robots, underwater robots, UAVs, pedestrians, humanoid robots, and bionic robots. These agents can be easily customized by developers with enriched attributes such as shape, material, and color. Additionally, developers can create their own models by using other tools. For instance, Webots can import files in VRML97 format, which can be produced using SolidWork [42]. Developers also can easily obtain the parameters of the agents, such as motor speed, accelerometer, camera screen, and other such parameters of the robot, via its interface. Webots also includes a range of sensors that are commonly used in robots, such as distance sensors, cameras, gyroscopes, and LIDARs. Moreover, Webots is capable of supporting the Robot Operating System (ROS) [22], an open-source framework that assists researchers and developers in creating and reusing code across various robotics applications. Additionally, Webots allows for the simulation of physical properties such as gravity, illumination, density, and friction coefficients. To simulate collision and dynamics, Webots uses an open-source physical simulation engine known as the Open Dynamics Engine (ODE). This engine facilitates the simulation of rigid body dynamics and collision systems, bringing the simulations closer to realism and greatly enhancing their reliability [43].

In addition to its reliable updates and maintenance, Webots has a thriving community and official technical support. The platform is widely used for simulation and control of UAV, offering numerous examples and tutorials for UAV simulation, as well as support for popular open-source flight control systems such as ArduPilot [44]. There is also an abundance of open-source tutorials available to developers who wish to learn from them. Because of its rich functionality and diverse development environments, more than two hundred universities and research centers use this platform to test applications and algorithms [45]. For example, it can be utilized to evaluate the performance and reliability of UAVs swarm algorithms, such as obstacle avoidance algorithms [46].

2.2. Gazebo

Gazebo, developed by Dr. Andrew Howard from the University of Southern California in 2002 [25], is a widely used open-source simulation platform in scientific research and engineering, particularly due to its close integration with ROS. Typically, it is used in conjunction with ROS, as Gazebo-ROS [47], to simulate complex robot systems, especially robot swarms [48].

Gazebo is compatible with Windows, Mac OS, Ubuntu, and other Linux distributions. It supports applications and algorithms developed in C, C++, and Python. Gazebo also offers a range of physical simulation engines, including ODE, Bullet [49], Simbody [21], and Dynamic Animation and Robotics Toolkit (DART) [50]. This enables the platform to simulate physical properties such as friction, gravity, and lighting [51]. Additionally, Gazebo can provide high-quality lighting and texture simulations by using its use of the Object-Oriented Graphics Rendering Engine (OGRE). Figure 2 illustrates the main interface and a simulation example of Gazebo.

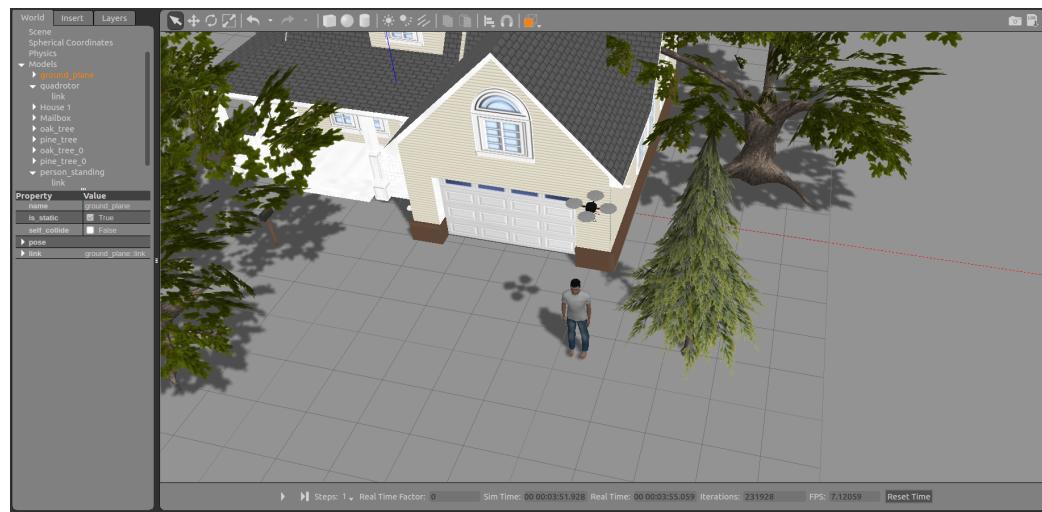


Figure 2. The main interface and a simulation example of Gazebo.

Gazebo comes with various built-in robot models, mainly wheeled and flying robots such as TurtleBot, PR2, and iRobot Create. The default model is relatively simple but suitable for complex validation calculations [23]. With ROS integration, this platform supports motion path planning. Sensors such as laser radar, monocular, stereo, RGB-D, and infrared sensors can also be simulated. Developers can customize plug-ins for robots and sensors using plug-in functions, including GUI, system, sensor, world, model, and visual. The sensor plug-in is used to model laser radar and infrared sensors. Additionally, the internal data of Gazebo can be directly accessed by its application program interface (API). Gazebo also offers cloud simulation, enabling developers to run projects on an online server [52].

The user operability of Gazebo is relatively poor. It can be difficult to install, and has a high threshold for users [17]. Built-in models are accessed online instead of locally, resulting in longer loading times or failure to load when the internet is unstable. Moreover, the built-in models are not well classified, making it difficult for users to locate the desired models quickly. Developers cannot edit models using this platform; they need to use 3D modeling software (such as Blender, SketchUp) and learn an XML-based SDF specification [52]. However, Gazebo can support for mainstream open-source flight control systems, such as PX4 and ArduPilot, which are widely used on real UAVs, has attracted researchers to evaluate the performance and reliability of UAV control algorithms, such as UAVs swarm validation and visual target tracking [53,54].

In short, the advantage of Gazebo is its close connection with the ROS, which maximizes the use of advantages of ROS. Additionally, its built-in models are accessible online,

providing developers with unrestricted access. However, the downside is that poor network connectivity can make it difficult for developers to connect to the server and load the models.

2.3. CoppeliaSim

CoppeliaSim is a powerful simulation platform for robots, formerly known as Vrep [26]. It supports ODE, Bullet, Vortex, and Newton physics engines, allowing for the simulation of dynamic scenarios such as collisions, rolling, gravity, and other physical factors [55,56]. Other than that, this platform is able to simulate dynamic particles such as jet engines, water jets, and propellers [57]. Its distributed architecture allows for scene object association or an unlimited number of scripts [58], and it supports applications or algorithms developed using C, C++, Python, Java, LUA, Matlab, or Octave. The conventional API is written in C or LUA. CoppeliaSim is highly portable, performs well on Windows, Mac OS, and Linux systems, and has kinematics solver and data visualization functions [56,59].

CoppeliaSim offers a variety of built-in agent models, such as biped, wheeled, bionic, flying, hexapod, and others. The user-friendly interface (Figure 3) allows developers to easily select appropriate models. Additionally, the platform supports importing models in mainstream formats such as URDF, COLLADA, DXF, OBJ, STL, and glTF. There are six options for developing applications or algorithms, including embedded script, plugin, add-on, ROS node, remote API, and node talking TCP/IP. CoppeliaSim also provides various sensors for detecting distances, visuals, and pressures. Each type of sensor has different variants; for instance, proximity sensor includes ray-type, randomized ray-type, pyramid-type, cylinder-type, disk-type, and cone-type; vision sensors are divided into orthogonal projection-type and perspective projection-type [60,61]. These sensors can meet the needs of most developers. Furthermore, this platform has rich development resources such as a large number of tutorials, API documents for developers, and an official forum. It is still frequently updated and released, making it a great tool for developers working with robotics simulations.

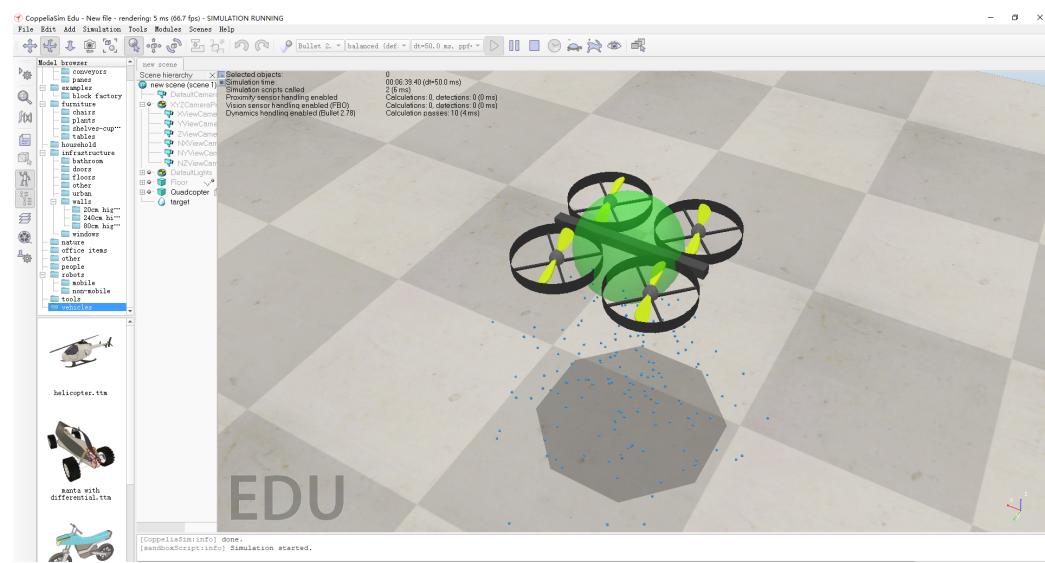


Figure 3. The main interface and a simulation example of CoppeliaSim.

To put it simply, this platform offers a rich selection of built-in models that can fulfill most simulation requirements. It also provides various UAV models, including multi-copter UAVs, with high realism and flexibility, enabling users to simulate different UAV behaviors and motion modes. To enhance UAV behavior simulation, CoppeliaSim allows users to incorporate different sensors such as cameras, LIDAR, GPS, and inertial navigation systems that provide perception and navigation capabilities to help UAVs perform tasks better. With

the powerful physics engine, users can simulate various physical effects and model UAV behavior more realistically. In simulations of UAVs swarm, communication and coordination between UAVs are crucial. This platform enables users to set communication protocols and frequencies between UAVs, simulating information exchange and collaborative actions. On the basis of network simulation, researchers can better understand and study UAVs swarm behavior. Many researchers currently use this platform to simulate and verify UAVs swarm algorithms such as entrapping multiple targets [62], formation control [63], and collaborative transportation [64]. However, due to the high accuracy and precision of the built-in models, they may not be suitable for simulating large numbers of swarms [23].

2.4. ARGoS

ARGoS is a simulation platform developed by the European Union as part of the Swarmanoid project. It is designed to support large-scale robot simulations, and provides a flexible and modular architecture that enables researchers to experiment with a wide range of algorithms and scenarios. ARGoS integrates several engines, including ODE-based 3D dynamics engine, 3D particle engine, Chipmunk-based 2D dynamics engine, and 2D dynamics engine [27]. It also comes with a built-in LUA script editor that allows researchers to easily create and modify simulation scenarios. While ARGoS lacks a scene editor, its modular architecture and built-in scripting capabilities enable researchers to customize and manipulate the simulation environment through custom scripts. The platform supports Ubuntu/Kubuntu, OpenSuse, and Mac OS, but does not currently support Windows [65]. Figure 4 illustrates the main interface of this platform and a simulation example based on it.

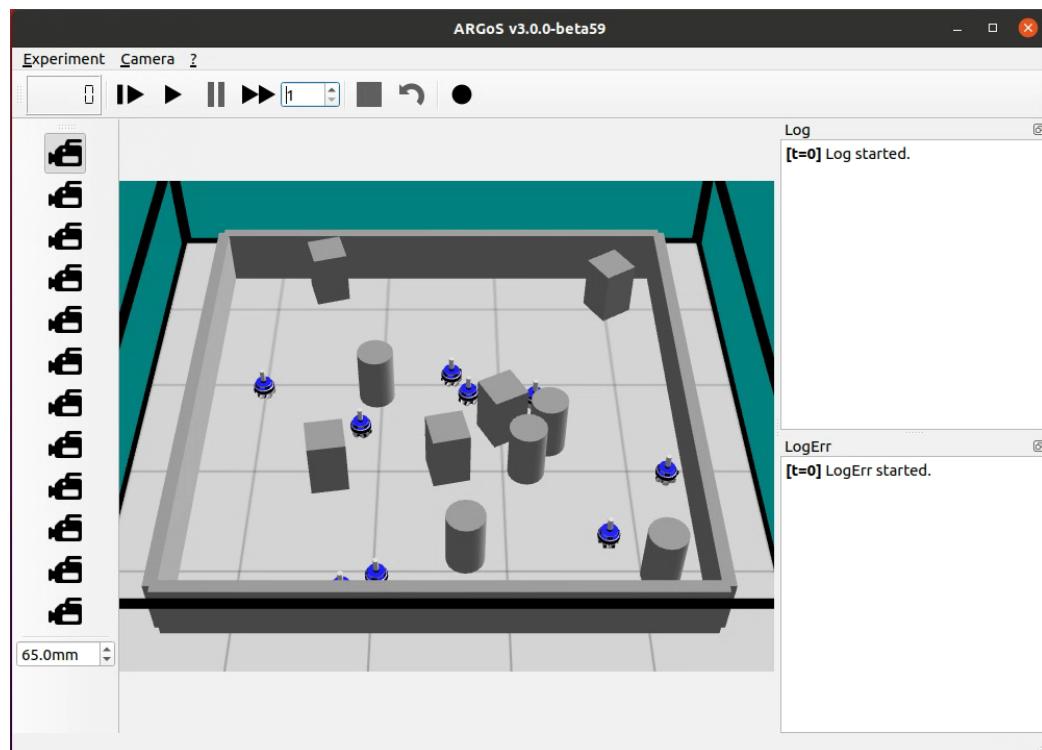


Figure 4. The main interface and a simulation example of by ARGoS.

The most significant feature of ARGoS is its ability to divide the simulation space into independent subspaces, and each is controlled by a separate physics engine. This enables concurrent execution of multiple simulations, resulting in significant reductions in processing time [27]. Additionally, the platform has a multi-threaded structure that includes master and slave threads. The master thread assigns the task of updating a single plug-in, such as a sensor, actuator, or physics engine, to the slave thread, which effectively employs multi-core processors to significantly improve computing performance, particularly for

multi-robot or swarm simulations. ARGoS can support the simulation of thousands of robots simultaneously, saving 40% of time compared to the two-dimensional dynamic verification of 10,000 robots [27].

The robot library comprised ARGoS relatively limited and includes only a few simple models such as e-puck, eye-bot, Kilobot, marXbot, and spiri robots. These robots can be programmed using LUA script or C++, but mesh importing is not supported, and the object representation is encoded with OpenGL [66]. While ARGoS does provide an official technical support forum and manuals for developers, the update frequency is not very high, and the development tutorial resources are not extensive.

In summary, ARGoS is a powerful simulation platform that is particularly well-suited for simulating robot swarms, especially when large numbers of robot models need to be generated and involved. It has been widely adopted for simulating swarm foraging robots due to its ability to simulate large-scale swarms of robots quickly [67]. This feature is also applicable to simulating UAVs swarms, where ARGoS is better suited for simulating large-scale swarms of UAVs compared to high-precision simulation of individual UAVs. However, ARGoS still has room for improvement. Its robot library needs to be enriched, and its poor cross-platform performance cannot be ignored. Additionally, 3D mesh models are currently not supported, which limits its use in certain applications. Despite these limitations, nonetheless, ARGoS remains a valuable tool for researchers in swarm robotics and related fields.

2.5. MRDS

The full name of MRDS is Microsoft Robotics Developer Studio, which is a 3D robot simulation platform designed and developed by Microsoft [28]. Figure 5 shows an outdoor scene simulated by MRDS. In addition to CPU, MRDS can also leverage other independent floating-point processors such as GPU and PPU for computation, making it capable of completing intensive computing-based simulations [68]. The platform is based on Physx Engine, which is commonly used on various simulation platforms for modeling rigid bodies and liquids [35,69,70].

MRDS can only be used on the Windows operating system, and the applications or algorithms for MRDS can be developed by VPL, C#, Visual Basic, Jscript, or Python [41]. Although MRDS is not an open-source platform, it can be used free of charge for non-commercial purposes. It uses decentralized software services (DSS) and concurrency and coordination runtime (CRR) to improve the performance of multi-line programming capabilities and to reuse the multi-core capabilities of processors [71]. This means that each service component can operate independently without synchronizing, allowing visual and acoustic services to be delivered independently without interruption. Even if one of the sensors fails, other functions will continue to operate normally. Unfortunately, this platform only provides limited robot models, such as iRobot, Lego NXT, and MobileRobotics, and it cannot support path planning [72]. The tutorials and development resources for MRDS are also very limited. Furthermore, the platform has not been updated for a long time, and its maintenance is becoming more and more difficult.

To be brief, the Physx Engine integrated into MRDS provides significant advantages to this platform, particularly in simulating harsh environments, such as sand. Additionally, the decentralized framework enhances its robustness and fault tolerance; even if one sensor fails, other robot functions can continue to operate normally. Users can leverage the Visual Simulation Environment (VSE) tool in MRDS to create and modify UAV models to suit their needs. Within VSE, users can design and adjust the physical entities, sensors, behaviors, and more of the UAV before controlling them with MRDS services. However, the platform has shortcomings such as limited built-in models, variable cross-platform performance, and maintenance challenges.

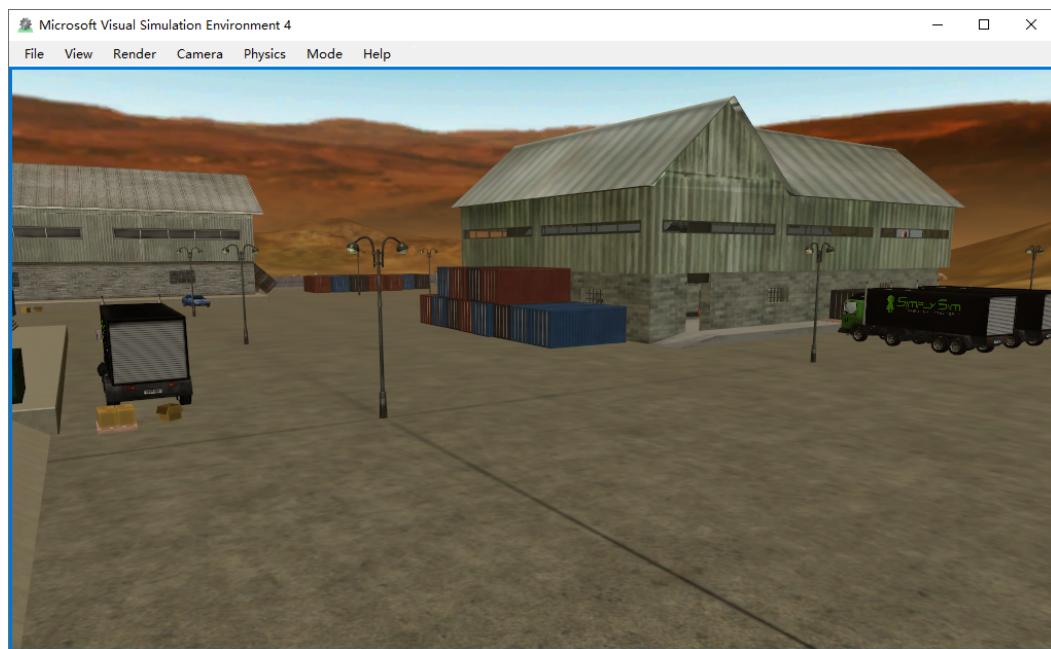


Figure 5. An outdoor scene simulated by MRDS.

2.6. MORSE

MORSE (Modular Open Robots Simulation Engine) is a simulation platform developed based on Blender engine [21], which offers a realistic graphics environment that can be used for testing and evaluating robots without any further modifications in real-life scenarios [73]. The simulation engine utilizes the Bullet library to define physical properties such as object shapes, mass, friction, boundary collision, and other physical factors. This platform provides three types of robot components, namely, sensors, actuators, and robots. Robots carry two components: sensors and actuators. The former is used for data collection, while the latter is responsible for executing instructions. In addition, there are three components related to environment interaction, including scenes, middlewares, and modifiers. Scenes are simulated environments such as buildings, indoor environments, and jungles. Middlewares are responsible for binding sensors and actuators. As the data generated by sensors in simulation do not contain any noise, the modifier is used to simulate the noise data to achieve an effect that is much closer to the real environment.

MORSE can be deployed on Linux, Windows, and Mac OS, but with a better support for Linux. This platform generates simulation scenes using a set of Python classes called Builder API, which provides a special internal language called domain-specific language (DSL) [74]. This allows developers to use Python to configure MORSE, even if they are not familiar with Blender. The built-in models contain agents such as iRobot ATRV unmanned ground vehicle (UGV) and Yamaha RMAX UAV, which can be enriched by importing Collada, DXF, 3DS Max, VRML, and other formats. MORSE also has built-in sensors such as cameras, gyroscopes, GPS, accelerometers, thermometers, and lasers [73], making it suitable for simulating realistic graphics. MORSE has unique advantages in human-computer interaction and UAVs attitude simulation. It provides UAV models such as multi-copter UAV in the model library. Users can define parameters such as the number of UAVs, their positions, sensors, controllers, etc. by editing Python scripts. The platform also provides a variety of simulation scenarios, such as the outdoor (environment), which meets the needs of UAV simulation, and users can also model the environment based on it. Researchers use MORSE to simulate UAVs, such as multi-UAV communication [75] and path planning [76,77]. The simulation of an UAVs swarm using MORSE is shown in Figure 6.

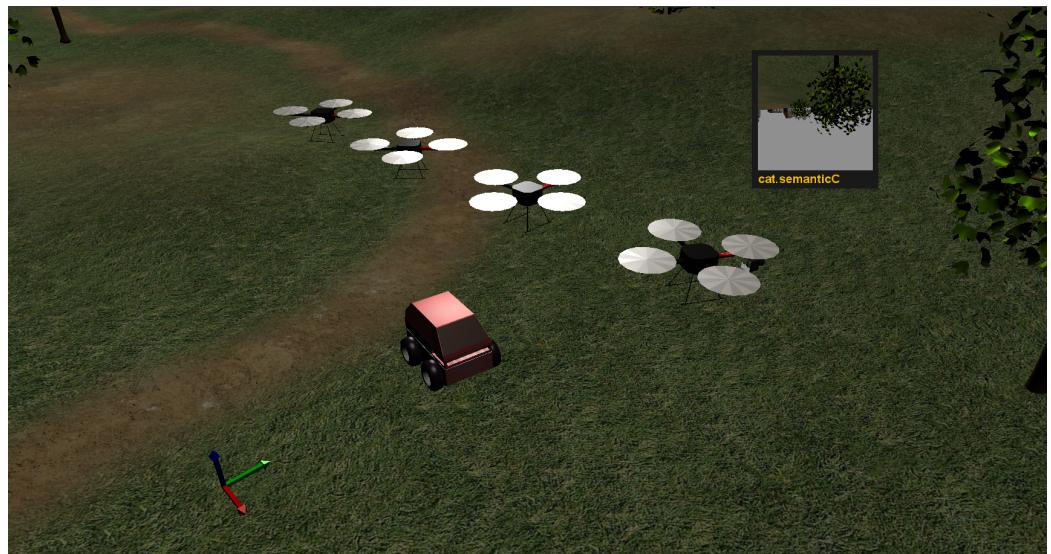


Figure 6. An UAVs swarm simulated by MORSE.

Unfortunately, the lack of updates since 2016 and limited community resources and development tutorials make it challenging for developers to use MORSE effectively. Furthermore, the absence of a graphical user interface and the requirement for developers to be familiar with command line tools and the Blender engine can be a barrier to entry for beginners. Additionally, MORSE lacks built-in algorithms such as path planning, which can make it challenging for developers to use it for more complex projects. Overall, while MORSE is not a good choice for entry-level developers due to its technical requirements and limited support resources.

2.7. USARSim

USARSim is a simulation platform developed by the National Institute of Standards and Technology in the USA for researching rescue robots and intelligence agents [29]. The platform is based on the Unreal Engine 2.0 developed by Epic Games [78], which provides high-quality 3D rendering capabilities using the PhysX physics engine. Applications and algorithms can be developed using C, C++, Java [41], or the built-in Unreal Script language. Control codes are developed using the GameBot interface, which is a communication interface exclusive to the phantom engine. USARSim is compatible with Windows, Linux, and Mac OS. An example of USARSim simulating UAVs can be seen in Figure 7.

The Mobility Open Architecture Simulation and Tools (MOAST) framework of USARSim offers a range of hierarchical and modular controllers, interfaces, and tools that provide motion generation, world modeling, and sensor information processing [79]. In addition, USARSim includes two controllers, Pyro and Player [80]. Pyro is a controller for AI exploration and robot development, which includes Python libraries, environments, a GUI, and drivers. Player is a robot service that can control the robot and sensors, such as touch sensors, lasers, and acoustics, in the simulation environment. Users can easily and fully control the controller and actuator carried by a robot using Player.

USARSim supports eight types of ground robots, including P2AT, P2DX, ATRV-Jr, Zerg, Tarantula, Talon, Telemax, and Soryu; four types of vehicles, namely, Hummer, Sedan, SnowStorm, and Cooper; two bipedal robots, QRIO and ERS, a helicopter, and a submarine. As USARSim is developed by using Unreal Engine 2.0, it has high-quality 3D capabilities. Furthermore, it supports multiple types of UAVs, including multi-copter UAV, as well as different sensors and controllers. It provides predefined scenarios such as buildings, earthquake zones, and tunnels, while also allowing users to create custom scenarios. Users can write their algorithms to control the UAV. Although USARSim was primarily designed for search and rescue missions, it has also been used by many researchers for UAV simulation, such as obstacle avoidance and navigation [81,82]. However, the robot

library is limited, and similar to MRDS, this platform has not been updated or maintained, resulting in poor support for new types of robots and sensors. While there are some official communities and manuals, community activity is weak, and tutorials are inadequate [21].

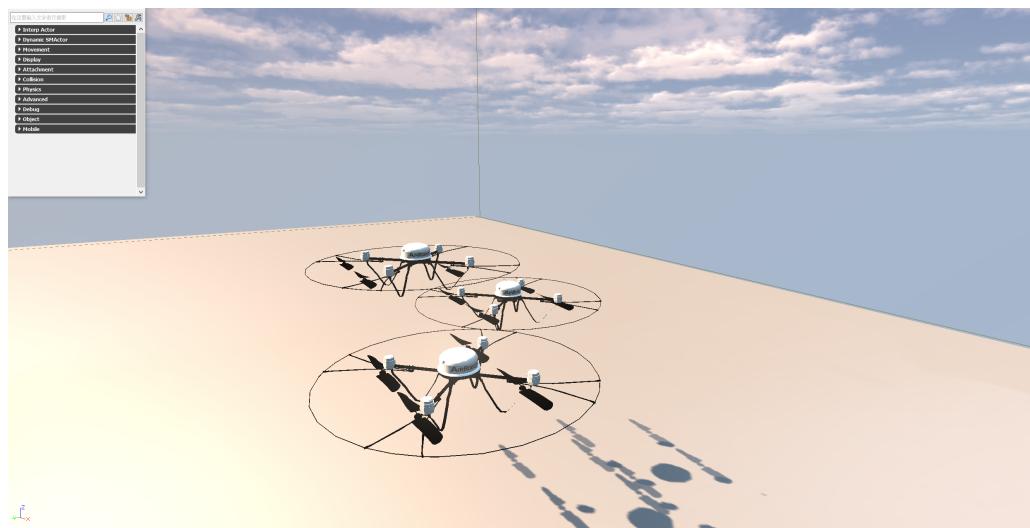


Figure 7. Simulation of UAVs by USARSim.

3. Selection of Simulation Platforms for Multi-Copter UAVs Swarm

This section provides an example of selecting a suitable simulation platform for the development of multi-copter UAVs swarm. Firstly, a general list of seven platforms is presented from the aspects of programming language, operating system (OS), physical engine, and sensors (Table 1). Then, the requirements for multi-copter UAVs swarm simulation are further refined into eight items. After comparing the eight items of the seven platforms, it is concluded that Webots is the comprehensive optimal choice for the simulation of multi-copter UAVs swarm.

3.1. The Requirements of Multi-Copter UAVs Swarm Simulation

Nowadays, UAVs swarm technology has been investigated in various applications, such as capture [83], target searching [84], cooperative operation [85], path planning [86], formation control [87], and artificial intelligence algorithms [88]. Having them in mind, and considering UAVs are not always work in outdoors, a simulation platform should be able to simulate other environments/scenarios, such as indoor, pipeline, forest, jungle, and canyon. Moreover, a simulation platform should have a good performance on computation when conducting simulations of multi-robot or swarm. Other than that, a platform should support applications developed by multiple programming languages, have rich built-in models, be compatible with multiple OS, have rich learning resources or technical support, and be user-friendly. To specify the requirements, we refer to previous surveys and summarize the following concerns [89,90].

- Multi-sensors. It can simulate the general sensors, such as GPS, radar, camera, sonar, and inertial measurement unit (IMU).
- Built-in models. It should have commonly used built-in agents/robots, e.g., UAVs, pedestrians, humanoid robots, unmanned underwater vehicles (UUV).
- Realism of the simulated environment. It can simulated both outdoor and non-outdoor environments/scenarios, such as indoor, jungle, and pipeline. Moreover, the simulation results should be as close as possible to the real environment. Meanwhile, the simulated data can be directly applied to real UAVs.
- Computation performance and accuracy. It can make full use of the existing computing resources to support a large number of UAVs swarm and a wide range of scenarios. The simulation accuracy meets certain requirements.

- Multiple programming languages and OS. It can support a variety of mainstream programming languages. Further, the developments can be directly transplanted to real machines.
- Usability. It has rich documents, technical support, and friendly interfaces. More importantly, it is easy to be reused (e.g., reuse the developed codes).
- Stability and maintenance. It is still being updated and maintained by developers or communities.
- Have commonly used functions or applications. It can support or has built-in commonly used applications, such as path planning, artificial intelligence algorithm, etc.

Table 1. Main characteristics of the seven simulation platforms.

	Programming Languages	OS	Physics Engines	Main Sensors	Access Addresses (accessed on 28 March 2023)
Webots [91]	C, C++, Java, Python, MATLAB	Windows, Linux, MAC OS	ODE	Accelerometer, altimeter, compass, GPS, gyroscope, distance sensor, inertial unit, position sensor, receiver, touch sensor, camera, LIDAR, radar, rangefinder	https://cyberbotics.com
Gazebo [92]	C, C++, Python	Mac OS, Linux, Windows	ODE, Bullet, Simbody, DART	Laser sensor, camera, inertial measurement, contact sensor, force sensor, torque sensor	https://gazebosim.org/home
CoppeliaSim [93]	C, C++, Python, Java, LUA, Matlab, Octave	Windows, Mac OS, Linux	ODE, Bullet, Vortex, Newton	Proximity sensors, Force sensors, Vision sensors, Cameras	https://www.coppeliarobotics.com
ARGoS [23]	Lua, C++	Ubuntu, KUbuntu, OpenSuse, Mac OS	ODE, 3D particle engine, 2D-dynamics open-source physics engine library Chipmunk, 2D-kinematics engine		https://www.argos-sim.info
MRDS [94]	VPL, C#, Visual Basic, JavaScript, Iron-Python	Windows	PhysX engine	Analog sensor, Analog sensor, Array, GPS, Contact sensors, Depth Camera, Encoder, Sonar, WebCam, SICK Laser Range Finder	https://learn.microsoft.com/en-us/previous-versions/microsoft-robotics/bb648760(v=msdn.10)
MORSE [95]	Python	Linux, Windows, Mac OS	Blender engine, Bullet	Accelerometer, Airspeed, Armature Pose sensor, Barometer, Attitude sensor, Collision, Battery sensor, Depth camera, Generic Camera, GPS, Gyroscope, Infrared Proximity sensor, Laser Scanner Sensors, Magnetometer, Odometry, Inertial measurement unit, Radar Altimeter, Thermometer sensor, Proximity Sensor, Velocity	https://www.openrobots.org/morse/doc/stable/morse.html
USARSim [96]	C, C++, Java	Windows, Linux, Mac OS	Unreal engine	State Sensor, Range sensor, Range Scanner sensor, Odometry sensor, GPS, INS, Encoder sensor, Touch Sensor, RFID, Sound sensor, Human-motion sensor, Robot Camera, Omnidirectional Camera	https://sourceforge.net/projects/usarsim

3.2. Comparisons of the Seven Platforms

3.2.1. Multi-Sensors

A UAV typically consists of three key components: the main flight controller, IMU, and navigation unit. The main flight controller is responsible for receiving and processing signals from the ground control station, then outputting the corresponding PWM signals to control the motors to execute specific actions. The IMU, which includes sensors such as barometers, accelerometers, gyroscopes, and compasses, receives and measures the attitude data of UAV. Finally, a navigation unit such as the global navigation satellite system (GNSS) is typically carried to provide accurate positioning and navigation information for the UAV.

Some previous studies have investigated the types of sensors required for UAVs [22]. Based on the sensors commonly equipped on UAVs and previous surveys, we consider the following five types of sensors to be important for UAVs swarm: GNSS, radar, camera, sonar, and IMU (Table 2). In the table, 'Yes' or 'No' indicates whether the sensor is supported

or not by a particular platform, while ‘NaN’ indicates that there is no information available regarding support for that sensor. We also propose a ‘Score’ column to evaluate the level of support for multiple sensors, with a higher score being awarded for platforms that support more sensors. In this scoring system, a ‘+’ symbol adds one point to the score, while a ‘–’ symbol represents a score of zero. The meaning of the symbols in the following sections is the same.

Table 2. Multi-sensors in the seven simulation platforms.

Platforms	GNSS	Radar	Camera	Sonar	IMU	Score
Webots	Yes	Yes	Yes	Yes	Yes	++
Gazebo	Yes	Yes	Yes	Yes	Yes	++
CoppeliaSim	Yes	Yes	Yes	Yes	Yes	++
ARGoS	NaN	NaN	NaN	NaN	NaN	–
MRDS	Yes	NaN	NaN	Yes	NaN	+
MORSE	Yes	Yes	Yes	No	No	+
USARSim	Yes	No	NaN	NaN	NaN	–

3.2.2. Various Types of Built-in Models

In general, UAVs are designed to fly in outdoor environments, but they may also need to operate in non-outdoor environments, such as indoor areas [97], jungles [98], and pipelines [99]. One notable feature of non-outdoor environments is that GNSS signals may be weak or even unavailable, which means that UAVs cannot rely on GNSS for positioning and navigation. In simulations, the main difference between outdoor and non-outdoor environments is whether the UAV carries GNSS sensors. If a GNSS sensor is loaded, the simulated environment can be regarded as outdoor; otherwise, it is considered non-outdoor. In addition to the environment type, the shapes of terrain and obstacles are also different between outdoor and non-outdoor scenarios. For instance, in indoor scenes, developers can create different shapes of obstacles without considering their appearance, color, material, and other information. However, if cameras on UAVs are used to detect the surroundings, developers need to add textures to the obstacles to make the model closer to the real world. Thus, simulation platforms need to support textured scenarios. Other than the environments, a simulation platform should also have general built-in agents/robots, such as UAVs, pedestrians, and humanoid robots.

In short, the ability of a simulation platform to simulate different environments depends on its built-in models. Moreover, the simplicity of the platform’s modeling method determines the threshold of development and the efficiency of modeling. One of the easiest ways is to provide a user-friendly interface that allows developers to directly drag and drop models into the simulation environment [52]. The comparison of different platforms on the built-in models and ease of the modeling process is shown in Table 3.

Table 3. Richness of built-in models and ease of modeling process.

Platforms	Richness of Built-in Models	Ease of Modeling Process	Score
Webots	Good	Good	++
Gazebo	Medium	Medium	+
CoppeliaSim	Good	Good	++
ARGoS	Poor	Medium	–
MRDS	Poor	Medium	–
MORSE	Good	Poor	+
USARSim	Poor	Medium	–

3.2.3. Realism of Simulated Environment

The realism of the simulated environment is another critical aspect of simulation platforms because real-world environments involve physical factors such as air density,

wind speed, gravity, friction, and more [100]. Thus, physical engines are developed, and their richness determines the realism of simulated environments.

To evaluate the ability of platforms to create realistic simulations, we investigate the physical engines used by each platform. The most widely used physical engines in simulation platforms are ODE, Bullet, PhysX, Unreal, and Blender. While ODE may not be as good as Bullet and PhysX in simulating collisions, it has better constraint and stacking stability. ODE also does not simulate friction as well as Bullet, but it is slightly better than PhysX in this aspect [101]. PhysX engine is excellent at collision detection and stacking stability, but it falls short in simulating friction. Although the Blender engine is not designed for simulation platforms, it can achieve advanced graphical details [73]. ODE has good documentation support and a stable API, but Bullet does not, and its API often changes [102]. Because there is no clear distinction between the advantages and disadvantages of each engine, and all seven platforms have their own physical engines, we consider all platforms to have the same score.

3.2.4. Computation Performance and Accuracy

The computational performance of simulation platforms is one of the major concerns for developers, especially when it comes to swarm simulations. Among the seven platforms, ARGoS is more suitable for simulating swarms compared to Gazebo and CoppeliaSim. CoppeliaSim is known to consume the most computing resources [23], while MORSE is more efficient than Gazebo for large-scale robot environment simulations [103]. In the comparison of the average CPU efficiency load among Webots, Gazebo, CoppeliaSim, and MORSE, Webots was found to be the best performing simulation platform. On the other hand, Gazebo has the lowest CPU efficiency, while the performance of MORSE and CoppeliaSim is similar [104]. However, it is worth noting that if Webots, USARSim, or Gazebo is used to simulate more than a dozen robots, it may become slower than on a physical robot [27].

When it comes to accuracy, taking the IMU simulation as an example, CoppeliaSim has the smallest error, followed by Gazebo, and MORSE has the worst performance. As stated on their official website, the authors do not consider MORSE as a physically accurate simulator [105]. Webots is similar to CoppeliaSim in terms of IMU angular velocity accuracy, but it lags behind in IMU linear acceleration accuracy [104]. While the Unreal engine of USARSim offers better image quality, its physical simulation performance is not good, making it unsuitable for accurate physical simulation [79,106]. The accuracy of MRDS in purely longitudinal motion and trajectory tracking simulations is not as good as the ODE engine [107]. For accurate physical simulation, it is recommended to use the ODE engine simulator rather than MRDS. While ARGoS offers the best computational performance, its simple model makes it difficult to use for high-precision simulation experiments. Table 4 shows the scores for the computational performance and accuracy of the seven platforms.

Table 4. The performance and accuracy of the seven platforms.

Platforms	Computation Performance	Accuracy	Score
Webots	Good	Good	+++
Gazebo	Poor	Good	+
CoppeliaSim	Medium	Good	++
ARGoS	Good	Poor	+
MRDS	Poor	Medium	-
MORSE	Medium	Medium	+
USARSim	Medium	Medium	+

3.2.5. Multiple Programming Languages and OS

One of the most significant considerations for developers when choosing a simulation platform is its ability to support multiple programming languages. If programming lan-

guage in the simulation platform does not match that of real robots, developers will have to translate it, which can increase their workload. For instance, Python is widely used in deep learning (DL) and machine learning (ML) applications [108]. Therefore, developers proficient in Python who want to simulate DL and ML algorithms or applications are likely to avoid choosing a platform that does not support Python. Additionally, a simulation platform that supports multiple mainstream programming languages can facilitate the process of transplanting simulated results to real machines, making it an attractive option for developers.

In addition, different developers prefer to or are familiar with different OS, and transferring knowledge from one type of OS to another can be time-consuming. Thus, the type of OS supported by a simulation platform also affects the preferences of developers, since programming languages are usually matched or associated with OS. We make scores on the two aspects together (Table 5).

Table 5. Comparison of multiple programming languages and cross-platform of the seven platforms.

Platforms	Multiple Programming Languages	Cross-Platform	Score
Webots	Good	Good	++
Gazebo	Good	Good	++
CoppeliaSim	Good	Good	++
ARGoS	Poor	Poor	–
MRDS	Good	Poor	+
MORSE	Poor	Good	+
USARSim	Medium	Good	+

3.2.6. Usability

The usability of simulation platforms is determined not only by their features but also by technical support factors such as official manuals, user guides, open-source projects, maintenance, and update frequency. Developers need to become familiar with the interface and framework of platforms before using, making these resources critical. Simulation platforms with active technical communities and development projects tend to be more popular among developers [90]. When facing complex technical issues, developers require adequate technical support to quickly solve problems. Table 6 displays the scores of technical supports and communities.

Table 6. Scores of technical supports and communities.

Platforms	Official Manual	Technical Supports/ Communities	Score
Webots	Yes	Good	+
Gazebo	Yes	Good	+
CoppeliaSim	Yes	Good	+
ARGoS	Yes	Medium	–
MRDS	Yes	Medium	–
MORSE	Yes	Medium	–
USARSim	Yes	Medium	–

3.2.7. Stability and Maintenance

The ongoing updating and maintenance of a simulation platform is crucial because new versions are typically developed to address bugs and stability issues in older versions. In the update log of CoppeliaSim, for example, each new version fixes previous errors and instabilities [109]. Continuous maintenance can improve the stability of simulation platforms, which further helps to ensure reliable simulation processes.

In addition, robots are constantly being updated and replaced with newer models, which makes it crucial for simulation platforms to keep pace with these changes. If a simulation platform is rarely updated or has stopped receiving maintenance, it will eventually become unstable and fall behind in terms of compatibility with new robot models. This means that its simulation results cannot be used to test and evaluate new robots effectively. Table 7 lists the maintenance and update status of seven simulation platforms. It should be added that although the three platforms (MRDS, MORSE, and USARSim) are no longer being updated, many researchers continue to use them [110–112], which is why we still consider them in this survey.

Table 7. The stability and maintenance of the seven platforms.

Platforms	Still Being Updated	Year of the Last Updating	Score
Webots	Yes	2022 [113]	+
Gazebo	Yes	2020 [114]	+
CoppeliaSim	Yes	2022 [109]	+
ARGoS	Yes	2022 [115]	+
MRDS	No	2014 [108]	–
MORSE	No	2016 [116]	–
USARSim	No	2013	–

3.2.8. Have Commonly Used Functions and Applications

Last but not least, the availability of commonly used functions and applications in a simulation platform is another crucial factor for developers to consider, because they want to focus their efforts on the specific issues they need to address rather than on a platform itself. For example, ROS is widely used in robot development, as it provides developers with access to many basic robot functions. Specifically, ROS can improve code utilization and enable developers to make the best use of their code [117]. This simplifies the development process by allowing developers to focus on the application layer. Therefore, the ability of a simulation platform to support ROS is an important criterion to consider when selecting a platform.

In addition, in recent years, as artificial intelligence (AI) algorithms such as reinforcement learning and deep learning have advanced, developers have begun to consider whether a simulation platform can support these algorithms as a critical factor in their selection criteria. Therefore, the ability of a simulation platform to support commonly used functions and applications is reflected in its support for ROS and AI algorithms (see Table 8).

Table 8. Support of ROS and AI algorithms.

Platforms	ROS and AI Algorithms	Reference	Score
Webots	Yes	[22]	+
Gazebo	Yes	[23]	+
CoppeliaSim	Yes	[23]	+
ARGoS	Yes	[118]	+
MRDS	No	–	–
MORSE	Yes	[73]	+
USARSim	Yes	[119]	+

3.3. A Case Study of Multi-Copter UAVs Swarm Based on Webots

Table 9 provides an overall comparison of the seven simulation platforms. Based on the scores, Webots is highly recommended for simulating multi-copter UAVs swarm. This platform offers a range of built-in sensors for UAVs, including accelerometers, compasses, GPS, gyroscopes, inertial sensors, cameras, LIDARs, and altimeters. Since it has built-in DJI Mavic Pro model, developers can easily obtain data on motor speed, gyroscope readings,

GPS coordinates, and other relevant information via the interface. The parameters of the built-in models are fully aligned with the physical models, making it easy to transfer simulated data to a physical UAV and ensuring the reliability of the simulation results.

Table 9. Overall comparisons of the seven simulation platforms.

Platforms	Multi-Sensors	Built-in Model Library	Physics Engine	Computation Performances and Accuracy	Programming Language	Cross-Platform	Usability	Being Updated	ROS and AI Algorithms	Overall Scores
Webots	++	++	+	+++	+	+	+	+	+	13
Gazebo	++	+	+	+	+	+	+	+	+	10
CoppeliaSim	++	++	+	++	+	+	+	+	+	12
ARGoS	—	—	+	+	—	—	—	+	+	4
MRDS	+	—	+	—	+	—	—	—	—	3
MORSE	+	+	+	+	—	+	—	—	+	6
USARSim	—	—	+	+	+	—	—	—	+	5

The case study here is to simulate a swarm with three UAVs (DJI Mavic 2 Pro) in an urban environment. The whole process includes six steps:

1. Create project and world files. To create a new simulation environment, start by clicking on “Wizards” in the top menu bar and selecting ‘New Project Directory’. Then, enter the name of the project and choose a storage location. Once done, click on ‘File’ and select ‘New World’ to begin setting up the simulation.
2. Designing the simulation environment. We firstly created an urban environment. To do this, navigate to the ‘Scene Tree’ and click on ‘Add New’. From there, select ‘PROTO nodes (Webots Projects)’ > ‘floors’ > ‘Floor(solid)’ to import the ground node. Adjust the ‘size’ parameter in the node to modify the size of the ground, and reset the material by clicking on ‘appearance’. Delete the original material and select the ground material from ‘PROTO nodes (Webots Projects)’ > ‘appearances’ > ‘Soil(PBRAppearance)’. Then, add obstacles including buildings, roads, and traffic by selecting ‘PROTO nodes (Webots Projects)’ > ‘objects’. To add a UAV, navigate to ‘rbots’ > ‘dji’ > ‘mavic’ > ‘Mavic2Pro(Robot)’ and add it to the simulation. Give it a name like ‘leader’. We also can add other vehicles and humans by selecting ‘vehicles’ and ‘humans’, respectively. The designed urban environment can be seen in Figure 8.



Figure 8. The urban environment designed by Webots.

3. Set UAV models. Continuing from the previous step, we created two additional ‘Mavic2Pro’ models identical to the first one, and named them as ‘follower’. After that,

we selected the ‘leader’ UAV node and right-click on ‘bodySlot’ > ‘Add New’ > ‘Base node’ > ‘Emitter’ to add a communication transmitter. Then, we selected ‘Receiver’ as the communication receiver on the ‘follower’ UAVs. We also set the communication distance, noise, channels, and any other parameters required for communication between the UAVs.

4. Set up controllers. Click on the ‘Wizards’ option in the top menu bar and select ‘New Robots Controller’. Choose the Python programming language as the controller language. Name the first controller as ‘leader control’, and similarly create the second controller called ‘follower control’. Write the control algorithm code, which includes UAV motion control, communication, sensor definition, and other necessary functions.
5. Set simulation parameters and start the simulation. In the ‘Scene Tree’, navigate to ‘WorldInfo’ and set the simulation parameters such as ‘basicTimeStep’ and ‘FPS’. We set the ‘basicTimeStep’ to 64 and ‘FPS’ to 60. After saving the project, click on the ‘Run the simulation in real-time’ button located at the top of the interface to start the simulation. Once the simulation starts, we can see the UAVs swarm moving in the environment Figure 9. These UAVs can take off, land, and move forward and backward simultaneously.
6. Debug the simulation and show results. Based on the simulation results, we can modify the controller code and the simulation environment. This may involve tweaking the parameters or algorithms used in the controller code, changing the obstacles or adding new elements to the environment. Use the ‘Show Robot Window’ feature to observe the sensor parameters of the UAV during the simulation. This allows us to gather data on the performance of the UAV in the environment. We can observe sensor parameters such as GPS, Gyro, InertialUnit, etc. Analyze the sensor data obtained during the simulation to refine the control algorithm and improve the performance of UAVs. Use the GPS data obtained during the simulation to analyze the UAV’s movement and improve its navigation capabilities. We visualized the GPS data and plotted it to gain insights into the UAV’s movement patterns (Figure 10).



Figure 9. Multi-copter UAVs swarm simulated by Webots.

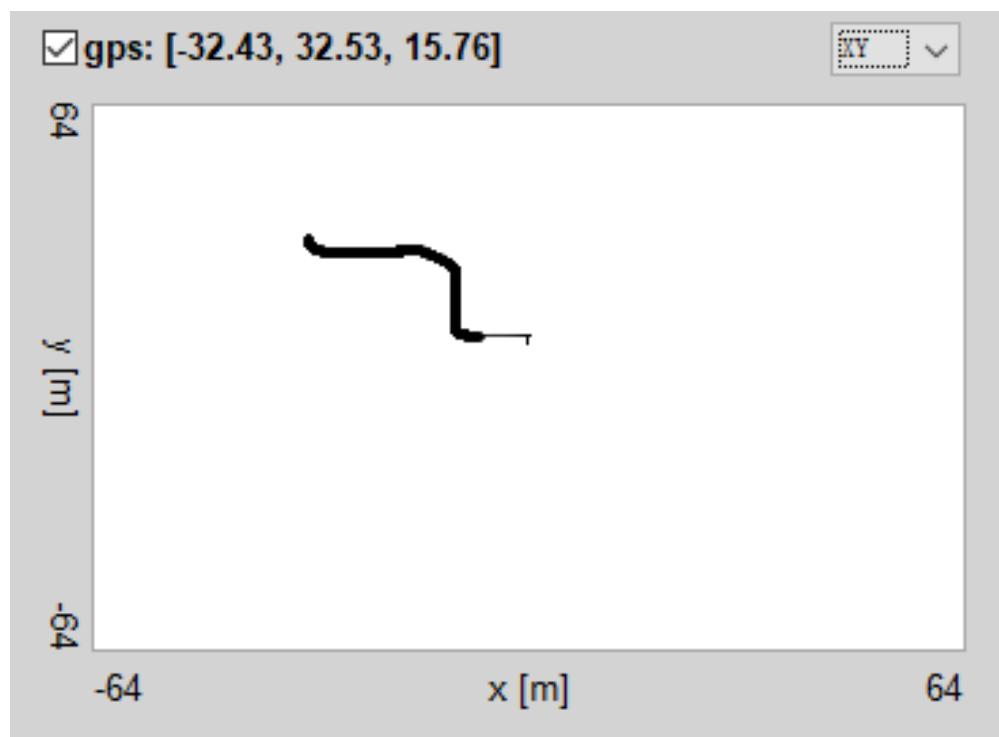


Figure 10. GPS simulation data in Webots.

3.4. Discussion

The above example shows how to make UAVs swarm simulations based on Webots. In addition to the Mavic2Pro UAV models, developers can import new (custom) models using VRML files. Developers can also add other sensors to their UAV models and develop their own controllers to achieve specific goals.

Other than the Webots, there are two suboptimal platforms for this case: ARGoS and MORSE. As mentioned in Section 2.4, ARGoS has significant advantages in simulating swarms with a large number of robots. If primary concern of a developer is simulating a large swarm, then ARGoS is a suitable choice. However, it should be noted that ARGoS has some drawbacks, including poor built-in models and sensors, limited technical support, and fewer available models compared to Webots. If a user does not require a large-scale simulation but needs high-precision simulation, it is not recommended to use the ARGoS. In such cases, Webots or CoppeliaSim will be a better choice as they offer more robust modeling and simulation capabilities.

MORSE provides many built-in sensors for UAVs, including accelerometer, flight speed sensor, an attitude sensor, barometer, magnetometer, IMU, odometer, and radar. It also contains a cloud platform unit, quad-rotor dynamic controller, rotator attitude motion controller, and rotator speed motion controller. In the simulation, UAVs have detailed representations instead of particle-like models used in some other platforms. However, one of the main disadvantages of MORSE is that it only supports Python and has limited technical resources. Additionally, it lacks graphical interfaces, which requires developers to have a good understanding of command-line interfaces and the Blender engine, increasing the learning curve for beginners. Moreover, MORSE is no longer being updated or maintained, which makes it less recommended for developers who want to build a long-term project. Therefore, we recommend developers to carefully consider their specific requirements before choosing a simulation platform, as each platform has its own strengths and weaknesses.

4. Conclusions

This paper provides a comprehensive review of seven open-source simulation platforms commonly used for simulating multi-copter UAV swarms, including Webots, Gazebo, CoppeliaSim, ARGoS, MRDS, MORSE, and USARSim. After reviewing the basic features of them, we presented the requirements of multi-copter UAVs swarm simulation, and then conducted a detailed analysis, proposed our own evaluation criteria, and finally compared of each platform from eight aspects: (i) multi-sensors; (ii) various types of built-in model libraries; (iii) realism of the simulated environments; (iv) computation performance and accuracy; (v) multiple programming languages and OS; (vi) usability; (vii) stability and maintenance; and (viii) commonly used functions or applications. Such a survey provides scientific researchers, developers, and users with a reference when choosing a simulation platform. Based on our evaluation, we conclude that Webots is the most suitable platform for simulating multi-copter UAV swarms, and we provide specific steps and simulation results to validate the effectiveness of the swarm control algorithm in an urban environment.

This research is of great value to researchers, developers, educators, and engineers who are seeking appropriate simulation platforms for their application developments, not only for multi-copter UAV swarms but also for other types of robots. Proper platforms can help them save expenses on testing and speed up the development progress. However, it should be noted that there have been few recent review papers on this topic, so some of the literature cited in this paper may be outdated, and some simulation platforms are updated frequently, which may lead to some discrepancies between our research and the current state of the platforms. Therefore, developers should follow the latest versions of the platforms to ensure their research and applications are up to date.

Author Contributions: Concept and methodology, J.Y., Z.C. and B.M.; formal analysis, Z.C. and J.Y.; investigation, Z.C., J.Y. and B.M.; resources, J.Y.; writing—original draft preparation, Z.C. and J.Y.; writing—review and editing, J.Y., Q.Y. and W.Y.; visualization, Z.C. and K.S.; supervision, J.Y.; project administration, J.Y. All authors have read and agreed to the published version of the manuscript.

Funding: The financial support for this work comes from the Fundamental Research Funds for the Central Universities (3072022FSC0401), Research on 3D path planning for low-speed miniature UAV swarms (KY00220023/005).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Tice, B.P. Unmanned aerial vehicles: The force multiplier of the 1990s. *Airpower J.* **1991**, *5*, 41–55.
2. Zhou, X.; Zhang, X. Individual tree parameters estimation for plantation forests based on UAV oblique photography. *IEEE Access* **2020**, *8*, 96184–96198. [[CrossRef](#)]
3. Panda, K.G.; Das, S.; Sen, D.; Arif, W. Design and deployment of UAV-aided post-disaster emergency network. *IEEE Access* **2019**, *7*, 102985–102999. [[CrossRef](#)]
4. Chen, Y.; Chen, M.; Chen, Z.; Cheng, L.; Yang, Y.; Li, H. Delivery path planning of heterogeneous robot system under road network constraints. *Comput. Electr. Eng.* **2021**, *92*, 107197. [[CrossRef](#)]
5. Wu, J.; Wang, H.; Zhang, M. Urban crowd surveillance in an emergency using unmanned air vehicles. *J. Guid. Control Dyn.* **2020**, *43*, 838–846. [[CrossRef](#)]
6. Fladeland, M.; Sumich, M.; Lobitz, B.; Kolyer, R.; Herlth, D.; Berthold, R.; McKinnon, D.; Monforton, L.; Brass, J.; Bland, G. The NASA SIERRA science demonstration programme and the role of small—Medium unmanned aircraft for earth science investigations. *Geocarto Int.* **2011**, *26*, 157–163. [[CrossRef](#)]
7. Khaloo, A.; Lattanzi, D.; Cunningham, K.; Dell’Andrea, R.; Riley, M. Unmanned aerial vehicle inspection of the Placer River Trail Bridge through image-based 3D modelling. *Struct. Infrastruct. Eng.* **2018**, *14*, 124–136. [[CrossRef](#)]
8. Moon, H.; Martinez-Carranza, J.; Cieslewski, T.; Faessler, M.; Falanga, D.; Simovic, A.; Scaramuzza, D.; Li, S.; Ozo, M.; De Wagter, C.; et al. Challenges and implemented technologies used in autonomous drone racing. *Intell. Serv. Robot.* **2019**, *12*, 137–148. [[CrossRef](#)]
9. Ordoukhianian, E.; Madni, A.M. Toward development of resilient multi-UAV system-of-systems. In Proceedings of the AIAA SPACE 2016, Long Beach, CA, USA, 13–16 September 2016; p. 5414.
10. Tang, J.; Duan, H.; Lao, S. Swarm intelligence algorithms for multiple unmanned aerial vehicles collaboration: A comprehensive review. *Artif. Intell. Rev.* **2022**. [[CrossRef](#)]

11. Wang, R.; Du, J.; Xiong, Z.; Chen, X.; Liu, J. Hierarchical collaborative navigation method for UAV swarm. *J. Aerosp. Eng.* **2021**, *34*, 04020097. [[CrossRef](#)]
12. Sørensen, L.Y.; Jacobsen, L.T.; Hansen, J.P. Low cost and flexible UAV deployment of sensors. *Sensors* **2017**, *17*, 154. [[CrossRef](#)] [[PubMed](#)]
13. How, J.P.; Behihke, B.; Frank, A.; Dale, D.; Vian, J. Real-time indoor autonomous vehicle test environment. *IEEE Control Syst. Mag.* **2008**, *28*, 51–64. [[CrossRef](#)]
14. Rao, B.; Gopi, A.G.; Maione, R. The societal impact of commercial drones. *Technol. Soc.* **2016**, *45*, 83–90. [[CrossRef](#)]
15. Bu, Q.; Wan, F.; Xie, Z.; Ren, Q.; Zhang, J.; Liu, S. General simulation platform for vision based UAV testing. In Proceedings of the 2015 IEEE International Conference on Information and Automation, Lijiang, China, 8–10 August 2015; pp. 2512–2516.
16. Altawy, R.; Youssef, A.M. Security, privacy, and safety aspects of civilian drones: A survey. *ACM Trans. Cyber-Phys. Syst.* **2016**, *1*, 1–25. [[CrossRef](#)]
17. Castillo-Pizarro, P.; Arredondo, T.V.; Torres-Torriti, M. Introductory survey to open-source mobile robot simulation software. In Proceedings of the 2010 Latin American Robotics Symposium and Intelligent Robotics Meeting, São Bernardo do Campo, Brazil, 23–28 October 2010; pp. 150–155.
18. Dierks, T.; Jagannathan, S. Output feedback control of a quadrotor UAV using neural networks. *IEEE Trans. Neural Netw.* **2009**, *21*, 50–66. [[CrossRef](#)] [[PubMed](#)]
19. Zeng, Y.; Xu, X.; Zhang, R. Trajectory design for completion time minimization in UAV-enabled multicasting. *IEEE Trans. Wirel. Commun.* **2018**, *17*, 2233–2246. [[CrossRef](#)]
20. Bell, N.; Yu, Y.; Mucha, P.J. Particle-based simulation of granular materials. In Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, Los Angeles, CA, USA, 29–31 July 2005; pp. 77–86.
21. Cook, D.; Vardy, A.; Lewis, R. A survey of AUV and robot simulators for multi-vehicle operations. In Proceedings of the 2014 IEEE/OES Autonomous Underwater Vehicles (AUV), Oxford, MS, USA, 6–9 October 2014; pp. 1–8.
22. Collins, J.; Chand, S.; Vanderkop, A.; Howard, D. A Review of Physics Simulators for Robotic Applications. *IEEE Access* **2021**, *9*, 51416–51431. [[CrossRef](#)]
23. Pitonakova, L.; Giuliani, M.; Pipe, A.; Winfield, A. Feature and performance comparison of the V-REP, Gazebo and ARGoS robot simulators. In Proceedings of the Annual Conference Towards Autonomous Robotic Systems, Bristol, UK, 22 July 2018; pp. 357–368.
24. Michel, O. Cyberbotics Ltd. Webots™: Professional mobile robot simulation. *Int. J. Adv. Robot. Syst.* **2004**, *1*, 5. [[CrossRef](#)]
25. Rivera, Z.B.; De Simone, M.C.; Guida, D. Unmanned ground vehicle modelling in Gazebo/ROS-based environments. *Machines* **2019**, *7*, 42. [[CrossRef](#)]
26. Rohmer, E.; Singh, S.P.; Freese, M. V-REP: A versatile and scalable robot simulation framework. In Proceedings of the 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems, Tokyo, Japan, 3–7 November 2013; pp. 1321–1326.
27. Pincioli, C.; Trianni, V.; O’Grady, R.; Pini, G.; Brutschy, A.; Brambilla, M.; Mathews, N.; Ferrante, E.; Di Caro, G.; Ducatelle, F.; et al. ARGoS: A modular, parallel, multi-engine simulator for multi-robot systems. *Swarm Intell.* **2012**, *6*, 271–295. [[CrossRef](#)]
28. Jackson, J. Microsoft robotics studio: A technical introduction. *IEEE Robot. Autom. Mag.* **2007**, *14*, 82–87. [[CrossRef](#)]
29. Zhibao, S.; Haojie, Z.; Sen, Z. A robotic simulation system combined USARSim and RCS library. In Proceedings of the 2017 2nd Asia-Pacific Conference on Intelligent Robot Systems (ACIRS), Wuhan, China, 16–18 June 2017; pp. 240–243.
30. Marhefka, D.W.; Orin, D.E. XAnimate: An educational tool for robot graphical simulation. *IEEE Robot. Autom. Mag.* **1996**, *3*, 6–14. [[CrossRef](#)]
31. Case, K.; Porter, M. SAMMIE, a computer-aided ergonomics design system. *Engineering* **1980**, 21–25.
32. Miller, A.T.; Allen, P.K. Graspit!: A versatile simulator for grasp analysis. In Proceedings of the ASME International Mechanical Engineering Congress and Exposition, Orlando, FL, USA, 5–10 November 2000; American Society of Mechanical Engineers: New York, NY, USA, 2000; Volume 26652, pp. 1251–1258. [[CrossRef](#)]
33. Gerkey, B.; Vaughan, R.T.; Howard, A. The player/stage project: Tools for multi-robot and distributed sensor systems. In Proceedings of the 11th International Conference on Advanced Robotics, Coimbra, Portugal, 30 June–3 July 2003; Volume 1, pp. 317–323.
34. Shimizu, M.; Takahashi, T. Simulation platform for performance test for robots and human operations. In Proceedings of the 2011 AAAI Fall Symposium Series, Francisco, CA, USA, 7–11 August 2011; pp. 61–66.
35. Glette, K.; Hovin, M. Evolution of artificial muscle-based robotic locomotion in PhysX. In Proceedings of the 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems, Taipei, Taiwan, 18–22 October 2010; pp. 1114–1119.
36. Maciel, A.; Halic, T.; Lu, Z.; Nedel, L.P.; De, S. Using the PhysX engine for physics-based virtual surgery with force feedback. *Int. J. Med. Robot. Comput. Assist. Surg.* **2009**, *5*, 341–353. [[CrossRef](#)] [[PubMed](#)]
37. Marcu, C.; Lazea, G.; Robotin, R. An OpenGL application for industrial robots simulation. In Proceedings of the 2006 IEEE International Conference on Automation, Quality and Testing, Robotics, Cluj-Napoca, Romania, 25–28 May 2006; Volume 2, pp. 254–259.
38. Hugues, L.; Bredeche, N. Simbad: An autonomous robot simulation package for education and research. In Proceedings of the International Conference on Simulation of Adaptive Behavior, Rome, Italy, 25–29 September 2006; pp. 831–842.
39. Yıldırım, Ş.; Arslan, E. ODE (Open Dynamics Engine) based stability control algorithm for six legged robot. *Measurement* **2018**, *124*, 367–377. [[CrossRef](#)]

40. Montemerlo, M.; Roy, N.; Thrun, S. Perspectives on standardization in mobile robot programming: The Carnegie Mellon navigation (CARMEN) toolkit. In Proceedings of the Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No. 03CH37453), Las Vegas, NV, USA, 27–31 October 2003; Volume 3, pp. 2436–2441.
41. Staranowicz, A.; Mariottini, G.L. A survey and comparison of commercial and open-source robotic simulator software. In Proceedings of the 4th International Conference on PErvasive Technologies Related to Assistive Environments, New York, NY, USA, 25–27 May 2011; pp. 1–8.
42. Guo, W.; Gao, Y.; Wang, Y. Design and realization of the interactive virtual laboratory based on VRML. In Proceedings of the 2012 2nd International Conference on Consumer Electronics, Communications and Networks (CECNet), Yichang, China, 21–23 April 2012; pp. 2510–2513.
43. Erez, T.; Tassa, Y.; Todorov, E. Simulation tools for model-based robotics: Comparison of bullet, havok, mujoco, ode and physx. In Proceedings of the 2015 IEEE International Conference on Robotics and Automation (ICRA), Seattle, WA, USA, 26–30 May 2015; pp. 4397–4404.
44. ArduPilot Dev Team. SITL with Webots. Available online: <https://ardupilot.org/dev/docs/sitl-with-webots.html> (accessed on 12 March 2023).
45. Vajta, L.; Juhasz, T. 3D Simulation in the advanced robotic design, test and control. *Int. J. Simul. Model.* **2015**, *4*, 105–117. [CrossRef]
46. Singh, A.; Jha, S.S. Learning safe cooperative policies in autonomous multi-uav navigation. In Proceedings of the 2021 IEEE 18th India Council International Conference (INDICON), Guwahati, India, 19–21 December 2021; pp. 1–6.
47. Lei, Z.; Hao, L.; Yu-fei, L.; Shuai, Z. Study on Simulation Optimization of Gazebo Based on Asynchronous Mechanism. *Comput. Sci.* **2020**, *47*, 593–598.
48. Kumar, A.S.; Manikutty, G.; Bhavani, R.R.; Couceiro, M.S. Search and rescue operations using robotic darwinian particle swarm optimization. In Proceedings of the 2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI), Udupi, India, 13–16 September 2017; pp. 1839–1843.
49. He, H.; Zheng, J.; Sun, Q.; Li, Z. Simulation of realistic particles with bullet physics engine. In *E3S Web of Conferences*; EDP Sciences: Les Ulis, France, 2019; Volume 92, p. 14004.
50. Lee, J.; Grey, M.X.; Ha, S.; Kunz, T.; Jain, S.; Ye, Y.; Srinivasa, S.S.; Stilman, M.; Liu, C.K. Dart: Dynamic animation and robotics toolkit. *J. Open Source Softw.* **2018**, *3*, 500. [CrossRef]
51. Mingo Hoffman, E.; Traversaro, S.; Rocchi, A.; Ferrati, M.; Settimi, A.; Romano, F.; Natale, L.; Bicchi, A.; Nori, F.; Tsagarakis, N.G. Yarp based plugins for gazebo simulator. In *Modelling and Simulation for Autonomous Systems. MESAS 2014*; Hodicky, J., Ed.; Lecture Notes in Computer Science; Springer: Cham, Switzerland, 2014; Volume 8906, pp. 371–382. [CrossRef]
52. Comparative analysis between gazebo and v-rep robotic simulators. *Semin. Interno Cognitaco Artif.-SICA* **2014**, *2*.
53. Bernardeschi, C.; Fagiolini, A.; Palmieri, M.; Scrima, G.; Sofia, F. Ros/gazebo based simulation of co-operative uavs. In Proceedings of the Modelling and Simulation for Autonomous Systems: 5th International Conference, MESAS 2018, Prague, Czech Republic, 17–19 October 2018; pp. 321–334.
54. Nguyen, K.D.; Nguyen, T.T. Vision-based software-in-the-loop-simulation for Unmanned Aerial Vehicles using gazebo and PX4 open source. In Proceedings of the 2019 International Conference on System Science and Engineering (ICSSE), Dong Hoi, Vietnam, 20–21 July 2019; pp. 429–432.
55. Hummel, J.; Wolff, R.; Stein, T.; Gerndt, A.; Kuhlen, T. An evaluation of open source physics engines for use in virtual reality assembly simulations. In Proceedings of the International Symposium on Visual Computing, San Diego, CA, USA, 3–5 October 2012; pp. 346–357.
56. Tursynbek, I.; Shintemirov, A. Modeling and simulation of spherical parallel manipulators in CoppeliaSim (V-REP) robot simulator software. In Proceedings of the 2020 International Conference Nonlinearity, Information and Robotics (NIR), Innopolis, Russia, 3–6 December 2020; pp. 1–6.
57. Obdržálek, Z. Mobile agents and their use in a group of cooperating autonomous robots. In Proceedings of the 2017 22nd International Conference on Methods and Models in Automation and Robotics (MMAR), Miedzyzdroje, Poland, 28–31 August 2017; pp. 125–130.
58. Freese, M.; Singh, S.; Ozaki, F.; Matsuhira, N. Virtual robot experimentation platform v-rep: A versatile 3d robot simulator. In Proceedings of the International Conference on Simulation, Modeling, and Programming for Autonomous Robots, Darmstadt, Germany, 15–18 November 2010; pp. 51–62.
59. Coppelia Robotics, L. Kinematics. Available online: <https://www.coppeliarobotics.com/helpFiles/en/kinematics.htm> (accessed on 22 March 2022).
60. Coppelia Robotics, L. Proximity Sensor Types and Mode of Operation. Available online: <https://www.coppeliarobotics.com/helpFiles/index.html> (accessed on 22 March 2022).
61. Coppelia Robotics, L. Vision Sensor Types and Mode of Operation. Available online: <https://www.coppeliarobotics.com/helpFiles/index.html> (accessed on 22 March 2022).
62. Wang, C.; Shi, Z.; Gu, M.; Luo, W.; Zhu, X.; Fan, Z. Revolutionary entrapment model of uniformly distributed swarm robots in morphogenetic formation. *Def. Technol.* **2022**. [CrossRef]

63. Virbora, N.; Sokoeun, U.; Saran, M.; Channareth, S.; Saravuth, S. Implementation of Matrix Drone Show Using Automatic Path Generator with DJI Tello Drones. In Proceedings of the 2022 International Conference on Engineering and Emerging Technologies (ICEET), Kuala Lumpur, Malaysia, 27–28 October 2022; pp. 1–5.
64. Huang, K.; Chen, J.; Oyekan, J. Decentralised aerial swarm for adaptive and energy efficient transport of unknown loads. *Swarm Evol. Comput.* **2021**, *67*, 100957. [[CrossRef](#)]
65. Pincioli, C. ARGoS Core. Available online: <https://www.argos-sim.info/core.php> (accessed on 22 March 2022).
66. Allwright, M.; Bhalla, N.; Pincioli, C.; Dorigo, M. Simulating multi-robot construction in ARGoS. In Proceedings of the International Conference on Swarm Intelligence, Shanghai, China, 17–22 June 2018; pp. 188–200.
67. Lu, Q.; Fricke, G.M.; Erickson, J.C.; Moses, M.E. Swarm foraging review: Closing the gap between proof and practice. *Curr. Robot. Rep.* **2020**, *1*, 215–225. [[CrossRef](#)]
68. Matta-Gómez, A.; Del Cerro, J.; Barrientos, A. Multi-robot data mapping simulation by using microsoft robotics developer studio. *Simul. Model. Pract. Theory* **2014**, *49*, 305–319. [[CrossRef](#)]
69. Wang, D.; Zhang, L.; Wang, M.; Xiao, T.; Hou, Z.; Zou, F. A simulation system based on ogre and physx for flexible aircraft assembly. In Proceedings of the 2012 ACM/IEEE/SCS 26th Workshop on Principles of Advanced and Distributed Simulation, Zhangjiajie, China, 15–19 July 2012; pp. 171–173.
70. Gechter, F.; Contet, J.M.; Galland, S.; Lamotte, O.; Koukam, A. Virtual intelligent vehicle urban simulator: Application to vehicle platoon evaluation. *Simul. Model. Pract. Theory* **2012**, *24*, 103–114. [[CrossRef](#)]
71. Cepeda, J.S.; Chaimowicz, L.; Soto, R. Exploring Microsoft Robotics Studio as a mechanism for service-oriented robotics. In Proceedings of the 2010 Latin American Robotics Symposium and Intelligent Robotics Meeting, São Bernardo do Campo, Brazil, 23–28 October 2010; pp. 7–12.
72. Michal, D.S.; Etzkorn, L. A comparison of player/stage/gazebo and microsoft robotics developer studio. In Proceedings of the 49th Annual Southeast Regional Conference, Kennesaw, GA, USA, 24–26 March 2011; pp. 60–66.
73. Echeverria, G.; Lassabe, N.; Degroote, A.; Lemaignan, S. Modular open robots simulation engine: Morse. In Proceedings of the 2011 IEEE International Conference on Robotics and Automation, Shanghai, China, 9–13 May 2011; pp. 46–51.
74. Echeverria, G.; Lemaignan, S.; Degroote, A.; Lacroix, S.; Karg, M.; Koch, P.; Lesire, C.; Stinckwich, S. Simulating complex robotic scenarios with MORSE. In Proceedings of the International Conference on Simulation, Modeling, and Programming for Autonomous Robots, Tsukuba, Japan, 5–8 November 2012; pp. 197–208.
75. Casas, V.; Mitschele-Thiel, A. On the impact of communication delays on UAVs flocking behavior. In Proceedings of the 2018 IEEE Wireless Communications and Networking Conference Workshops (WCNCW), Barcelona, Spain, 15–18 April 2018; pp. 67–72.
76. Casas, V.; Mitschele-Thiel, A. From simulation to reality: A implementable self-organized collision avoidance algorithm for autonomous UAVs. In Proceedings of the 2020 International Conference on Unmanned Aircraft Systems (ICUAS), Athens, Greece, 1–4 September 2020; pp. 822–831.
77. Dias, A.; Fernandes, T.; Almeida, J.; Martins, A.; Silva, E. 3D path planning methods for unmanned aerial vehicles in search and rescue scenarios. In *Human-Centric Robotics: Proceedings of the CLAWAR 2017: 20th International Conference on Climbing and Walking Robots and the Support Technologies for Mobile Machines*, Porto, Portugal, 11–13 September 2017; World Scientific: Singapore, 2018; pp. 213–220.
78. Balakirsky, S.; Scrapper, C.; Carpin, S.; Lewis, M. USARSim: Providing a framework for multi-robot performance evaluation. In Proceedings of the Performance Metrics for Intelligent Systems (PerMIS) Workshop, Gaithersburg, MD, USA, 14–16 August 2006; pp. 98–102.
79. Carpin, S.; Lewis, M.; Wang, J.; Balakirsky, S.; Scrapper, C. USARSim: A robot simulator for research and education. In Proceedings of the 2007 IEEE International Conference on Robotics and Automation, Rome, Italy, 10–14 April 2007; pp. 1400–1405.
80. Lewis, M.; Wang, J.; Hughes, S. USARSim: Simulation for the study of human-robot interaction. *J. Cogn. Eng. Decis. Mak.* **2007**, *1*, 98–120. [[CrossRef](#)]
81. Mendes, J.; Ventura, R. Safe teleoperation of a quadrotor using FastSLAM. In Proceedings of the 2012 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR), College Station, TX, USA, 5–8 November 2012; pp. 1–6.
82. Drews, S.; Lange, S.; Protzel, P. Validating an active stereo system using USARSim. In Proceedings of the Simulation, Modeling, and Programming for Autonomous Robots: Second International Conference, SIMPAR 2010, Darmstadt, Germany, 15–18 November 2010; pp. 387–398.
83. Yu, J.; Dong, X.; Li, Q.; Ren, Z. Distributed cooperative encirclement hunting guidance for multiple flight vehicles system. *Aerospace Sci. Technol.* **2019**, *95*, 105475. [[CrossRef](#)]
84. Zhou, Z.; Luo, D.; Shao, J.; Xu, Y.; You, Y. Immune genetic algorithm based multi-UAV cooperative target search with event-triggered mechanism. *Phys. Commun.* **2020**, *41*, 101103. [[CrossRef](#)]
85. Jia, N.; Yang, Z.; Yang, K. An operational effectiveness evaluation method of the swarming UAVs air combat system. In *MATEC Web of Conferences*; EDP Sciences: Les Ulis, France, 2019; Volume 277, p. 02010.
86. Sharma, A.; Shoval, S.; Sharma, A.; Pandey, J.K. Path Planning for Multiple Targets Interception by the Swarm of UAVs based on Swarm Intelligence Algorithms: A Review. *IETE Tech. Rev.* **2022**, *39*, 675–697. [[CrossRef](#)]
87. Jia, Z.; Wan, Y.H.; Zhou, Y.J.; Jiang, G.P.; Zhang, D. Formation shape control of multi-UAV with collision avoidance. In Proceedings of the 2018 33rd Youth Academic Annual Conference of Chinese Association of Automation (YAC), Nanjing, China, 18–20 May 2018; pp. 305–310.

88. Akhloufi, M.A.; Arola, S.; Bonnet, A. Drones chasing drones: Reinforcement learning and deep search area proposal. *Drones* **2019**, *3*, 58. [CrossRef]
89. Craighead, J.; Murphy, R.; Burke, J.; Goldiez, B. A survey of commercial open source unmanned vehicle simulators. In Proceedings of the 2007 IEEE International Conference on Robotics and Automation, Rome, Italy, 10–14 April 2007; pp. 852–857.
90. Ivaldi, S.; Padois, V.; Nori, F. Tools for dynamics simulation of robots: A survey based on user feedback. *arXiv* **2014**, arXiv:1402.7050.
91. Cyberbotics. Webots User Guide. Available online: <https://www.cyberbotics.com/doc/guide/sensors> (accessed on 8 January 2022).
92. Open Source Robotics Foundation. Category: Sensors. Available online: <http://gazebosim.org/tutorials?cat=sensors> (accessed on 8 January 2022).
93. Coppelia Robotics, L. CoppeliaSim User Manual. Available online: <https://www.coppeliarobotics.com/helpFiles/index.html> (accessed on 8 January 2022).
94. Microsoft Corporation. Robotics Common Overview. Available online: [https://docs.microsoft.com/en-us/previous-versions/microsoft-robotics/cc998481\(v=msdn.10\)](https://docs.microsoft.com/en-us/previous-versions/microsoft-robotics/cc998481(v=msdn.10)) (accessed on 8 January 2022).
95. Components Library. Components Library. Available online: https://www.openrobots.org/morse/doc/latest/components_library.html (accessed on 8 January 2022).
96. Wang, J. USARSim-Manual-3.1.3. Available online: <https://sourceforge.net/projects/usarsim/files/Documentation/3.1.3/> (accessed on 28 March 2023).
97. Fei, W.; Jin-Qiang, C.; Ben-Mei, C.; Tong, H.L. A comprehensive UAV indoor navigation system based on vision optical flow and laser FastSLAM. *Acta Autom. Sin.* **2013**, *39*, 1889–1899.
98. Dai, X.; Mao, Y.; Huang, T.; Qin, N.; Huang, D.; Li, Y. Automatic obstacle avoidance of quadrotor UAV via CNN-based learning. *Neurocomputing* **2020**, *402*, 346–358. [CrossRef]
99. Bretschneider, T.; Shetti, K.R. UAV-based gas pipeline leak detection. In Proceedings of the ARCS 2015, Porto, Portugal, 24–27 March 2015.
100. Hentati, A.I.; Krichen, L.; Fourati, M.; Fourati, L.C. Simulation tools, environments and frameworks for UAV systems performance analysis. In Proceedings of the 2018 14th International Wireless Communications & Mobile Computing Conference (IWCMC), Limassol, Cyprus, 25–29 June 2018; pp. 1495–1500.
101. Rönnau, A.; Sutter, F.; Heppner, G.; Oberländer, J.; Dillmann, R. Evaluation of physics engines for robotic simulations with a special focus on the dynamics of walking robots. In Proceedings of the 2013 16th International Conference on Advanced Robotics (ICAR), Montevideo, Uruguay, 25–29 November 2013; pp. 1–7.
102. Mouret, J.B.; Chatzilygeroudis, K. 20 years of reality gap: A few thoughts about simulators in evolutionary robotics. In Proceedings of the Genetic and Evolutionary Computation Conference Companion, Berlin, Germany, 15–19 July 2017; pp. 1121–1124.
103. Noori, F.M.; Portugal, D.; Rocha, R.P.; Couceiro, M.S. On 3D simulators for multi-robot systems in ROS: MORSE or Gazebo? In Proceedings of the 2017 IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR), Shanghai, China, 11–13 October 2017; pp. 19–24.
104. Farley, A.; Wang, J.; Marshall, J.A. How to pick a mobile robot simulator: A quantitative comparison of CoppeliaSim, Gazebo, MORSE and Webots with a focus on accuracy of motion. *Simul. Model. Pract. Theory* **2022**, *120*, 102629. [CrossRef]
105. Components Library. MORSE Limitations. Available online: https://www.openrobots.org/morse/doc/stable/what_is_morse.html (accessed on 14 March 2023).
106. Qiu, W.; Yuille, A. Unrealcv: Connecting computer vision to unreal engine. In Proceedings of the Computer Vision–ECCV 2016 Workshops, Amsterdam, The Netherlands, 8–10 October 2016; pp. 909–916.
107. Torres-Torriti, M.; Arredondo, T.; Castillo-Pizarro, P. Survey and comparative study of free simulation software for mobile robots. *Robotica* **2016**, *34*, 791–822. [CrossRef]
108. Li, S.; Wan, Y.; He, P.; Wang, C.; Sun, J.; Zhang, Y.; Li, X.; Xie, G. Heros: A simulation platform for heterogeneous robotic swarms. In Proceedings of the 2019 Chinese Control Conference (CCC), Guangzhou, China, 27–30 July 2019; pp. 7223–7228.
109. Coppelia Robotics, L. CoppeliaSim Version History. Available online: <https://www.coppeliarobotics.com/helpFiles/en/versionInfo.htm#coppeliaSim4.2.0> (accessed on 7 June 2022).
110. Gustiana, M.; Indrawaty, Y.; Febriandi, A. Perancangan Mobile Manipulator Robot Secara Simulasi Menggunakan Microsoft Robotics Developer Studio. *MIND (Multimed. Artif. Intell. Netw. Database) J.* **2018**, *3*, 15–23. [CrossRef]
111. Askarpour, M.; Rossi, M.; Tiraykiler, O. Co-simulation of human-robot collaboration: From temporal logic to 3D simulation. *arXiv* **2020**, arXiv:2007.11737.
112. Hong, A.; Igharoro, O.; Liu, Y.; Niroui, F.; Nejat, G.; Benhabib, B. Investigating human-robot teams for learning-based semi-autonomous control in urban search and rescue environments. *J. Intell. Robot. Syst.* **2019**, *94*, 669–686. [CrossRef]
113. Cyberbotics. Webots Homepage. Available online: <https://www.cyberbotics.com> (accessed on 12 December 2022).
114. Open Source Robotics Foundation. Gazebo Download. Available online: <http://www.gazebosim.cn/download.html> (accessed on 8 January 2022).
115. Pincioli, C. ARGoS Homepage. Available online: <https://www.argos-sim.info/> (accessed on 7 June 2022).
116. Components Library. Latest News. Available online: <http://morse-simulator.github.io/> (accessed on 8 January 2022).

117. Quigley, M.; Conley, K.; Gerkey, B.; Faust, J.; Foote, T.; Leibs, J.; Wheeler, R.; Ng, A.Y. ROS: An open-source Robot Operating System. In Proceedings of the ICRA Workshop on Open Source Software, Kobe, Japan, 12–17 May 2009; Volume 3, p. 5.
118. Vardy, A. ARGoS-ROS. Available online: <https://www.argos-sim.info/extensions.php> (accessed on 7 June 2022).
119. Hao-Jie, Z.; Zhi-Bao, S.; Tian-Tian, Y. Design of Team Formation Simulation System for Unmanned Ground Vehicles Based on USARSim and ROS. *Acta Autom. Sin.* **2021**, *47*, 1390–1400.

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.