

UTSim: A framework and simulator for UAV air traffic integration, control, and communication

*International Journal of Advanced
Robotic Systems*
September-October 2019: 1–19
© The Author(s) 2019
DOI: 10.1177/1729881419870937
journals.sagepub.com/home/arx



Amjed Al-Mousa¹ , Belal H Sababha¹, Nailah Al-Madi²,
Amro Barghouthi¹ and Remah Younis¹

Abstract

The interest in unmanned systems especially unmanned aerial vehicle is continuously increasing. Unmanned aerial vehicles started to become of great benefit in many different fields. It is anticipated that unmanned aerial vehicles will soon become a main component of the future urban air traffic. The integration of unmanned aerial vehicles within existing air traffic environments has started getting the attention of researchers. Integrating unmanned systems in the real-world urban air traffic requires the development of tools and simulators to enable researchers in their ongoing efforts. In this article, a simulator called UTSim is introduced. The proposed simulator is built using the Unity platform. UTSim is capable of simulating unmanned aerial vehicle physical specification, navigation, control, communication, sensing and avoidance in environments with static and moving objects. The simulator enables studying and exploring several unmanned aerial vehicle air traffic integration issues like sense and avoid, communication protocols, navigation algorithms, and much more. UTSim is designed and developed to be easily used. The user can specify the properties of the environment, the number and types of unmanned aerial vehicles in the environment, and specify the algorithm to be used for path planning and collision avoidance. The simulator outputs a log file with a lot of useful information such as the number of sent and received messages, the number of detected objects and collided unmanned aerial vehicles. Three scenarios have been implemented in this article to present the capabilities of UTSim and to illustrate how it can benefit researchers in the field of integrating unmanned aerial vehicles in urban air traffic.

Keywords

Air traffic integration, control and communication, sense and avoid (S&A), UAV, simulation, Unity, unmanned aerial systems

Date received: 29 April 2019; accepted: 29 July 2019

Topic: Mobile Robots and Multi-Robot Systems

Topic Editor: Andrey V Savkin

Associate Editor: Istvan Harmati

Introduction

An unmanned aerial vehicle (UAV) is an aircraft that does not have a human pilot onboard. UAVs are either manually operated through a remote human pilot or autonomously, where an onboard autopilot has a predefined mission or receives mission commands from a ground station. UAVs can operate in hazardous environments and perform some dangerous missions, where human life can be at risk. Compared to manned systems, UAVs are usually of lightweight, able to fly at lower altitudes, and provide a relatively lower

cost alternative.¹ UAVs have been of interest in various fields such as military, policing, firefighting, natural disaster rescue, and even monitoring missions to document

¹Computer Engineering Department, PSUT, Amman, Jordan

²Computer Science Department, PSUT, Amman, Jordan

Corresponding author:

Amjed Al-Mousa, Computer Engineering Department, PSUT, Amman 111941, Jordan.

Email: a.almousa@psut.edu.jo



Creative Commons CC BY: This article is distributed under the terms of the Creative Commons Attribution 4.0 License

(<http://www.creativecommons.org/licenses/by/4.0/>) which permits any use, reproduction and distribution of the work without further permission provided the original work is attributed as specified on the SAGE and Open Access pages (<https://us.sagepub.com/en-us/nam/open-access-at-sage>).

wildlife abundance, behavior, and habitat.²⁻⁵ These different types of UAVs have different characteristics and capabilities, such as size, speed, power, energy source, communication methods and protocols, sensing range, payload, flight duration, maneuvering, and much more. These characteristics are subject to optimization based on the intended application.⁶

UAV research is rising in many different areas and in many different directions. One field of UAV research is trajectory motion planning.⁷ In the case of UAVs, the typical robotics path planning algorithms get more complicated, as motion becomes a three dimensional (3-D) problem instead of a two dimensional (2-D), which increases the degrees of freedom. However, UAVs do not have the limitations of ground robots, which usually cannot overcome rocks, climb stairs, or get access to ceilings. Nonetheless, UAVs need to exhibit a notable degree of awareness and exactness to accomplish their navigation and obstacle avoidance tasks successfully.⁸ Another field of research is the integration of computer vision algorithms, visual localization and mapping, obstacle detection, and target tracking with UAVs.⁹⁻¹²

Although governmental entities' efforts are still undergoing to develop governance and rules to fly small UAVs for various commercial and other purposes, some of the main guidelines and rules started appearing recently. The Federal Aviation Administration (FAA) released in the second half of 2016 the rules (Part 107) for operating small UAVs weighing below 55 lbs (25 kg).¹³ Some of the operation rules may be subject to waiver by the administration. However, until recently, other rules such as the "Visual line of sight aircraft operation (§ 107.31)" rule could not have been waived.¹⁴ The rule required the unmanned aircraft to remain within the visual line of sight of a visual observer. A beyond visual line of sight waiver has been recently granted by FAA to a Google's spin-off drone delivery business (Wing Aviation LLC) to start operating in two rural communities in Virginia.¹⁵

Despite the strict rules of operating UAVs, some commercial delivery companies already started the R&D and testing to deliver via Drones. Amazon.com introduced (Amazon Prime Air)¹⁶ and United Parcel Service of America, Inc. (UPS) recently started residential delivery tests via Drone launched from the top of a truck.¹⁷ In addition, Wing Aviation has started its operations in Australia already within selected suburbs.¹⁸ While these limited licenses are for UAVs operating in rural and less populated areas, soon drone deliveries will come closer to urban areas, and we will start having more dense skies with small size UAV systems. This would require developing air traffic control (ATC) and management algorithms and protocols. And this would, in turn, require proper and easy to use simulators to implement and test the developed algorithms with multiple real-life scenarios before implementing them on hardware prototypes.

The availability of computer simulation resources benefited UAV development and integration in real-life

applications as they reduce cost, risks, development and testing time, as well as increase safety levels.¹⁹ Some researchers develop their own simulation environments for their own custom special UAVs and applications.²⁰ However, using multiple UAV simulation environments makes it difficult to build sustainable research, as it becomes hard to compare different algorithms that were simulated using different simulation platforms. In addition, it is very time-consuming to resimulate an algorithm, which sometimes is not well documented, on a new platform to compare performance.

Therefore, there is a real need to develop a standard and easy to use simulator that can simulate different scenarios and applications using any type of UAV, with different properties. The simulator should allow for implementing different algorithms, such as path planning, obstacle avoidance, communication protocols, and much more.

The proposed simulator is developed using a well known and well-maintained game engine called Unity.²¹ This popular, easy to use game engine is used to develop simulators and video games for computers, game consoles, and mobile devices. It supports 2-D and 3-D graphics. The user-friendliness, availability of resources, and the popularity of Unity will make the proposed simulator of interest for a wide variety of researchers. In fact, Unity has been used to develop simulators for other similar applications, like connected vehicles^{22,23,68} and remote vehicle piloting.²⁴ It is also used to simulate intelligent agents,²⁵ robotics,²⁶ some medical application,^{27,28} and much more.²⁹⁻³¹

The contribution of the article is easy to use and customizable UAV simulator called UTSim. UTSim is capable of simulating UAV navigation in 3-D space with the ability to produce enough data that would allow for the evaluation of the implemented algorithms. The simulator supports the integration of multiple algorithms like trajectory planning and collision avoidance algorithms, as well as the interaction with active and passive objects. UTSim can be used as a standard simulator for any UAV application that may consist of any number and any type of UAVs. In addition, the fact that UTSim is built on top of an existing well-maintained game engine "Unity," would allow for utilizing any additional features that get added to the engine.

The rest of the article is organized as follows: The second section overviews related work. The third section presents the simulator design in detail, the platform, and the architecture. Then, in the fourth section, the capabilities of the proposed simulator are discussed. Three different test case scenarios are presented in the fifth section to showcase some of the results that can be achieved by UTSim. Finally, the sixth section concludes the article.

Literature review

Simulation is a very important and cost-effective tool, especially if an actual real-life implementation is costly or associated with hazardous conditions. Moreover, simulators

have other advantages like paving the way to the adoption of new technologies, lowering training costs, and, most importantly, enabling researchers to experiment with much fewer resources and time. For example, the research presented in the study by Visser et al.³² showed the importance of developing a simulator, called “AR.Drone,” with advanced navigation capability using realistic sensor and motion models. Therefore, the literature is rich with simulators that were proposed to simulate machines and environments for training, testing, research and development purposes.^{33–38} Robots behavior and motion are of high importance, thus multiple simulators focused on addressing robots simulation.^{39–42}

Another type of simulators is flight simulators, which is a software-based virtual reality system that enables pilots, engineers, and researchers to practice, test, experiment, and research. Such simulators are used in training pilots in a virtual environment which is extremely safer and less expensive. They are also used by engineers to design and test aircraft models and characteristics. Researchers also benefit from such simulators in developing and testing aircraft controls and air traffic management protocols and communication systems.

From the application perspective, flight simulators simulate the behavior of an aircraft considering the aircraft design, the input commands, and the environment. Behavior is predicted through a computation model that is based either on empirical data in forms of lookup tables or based on a live physics engine such as the “blade element theory.”^{43,44} Examples of such simulators include X-Plane by Laminar Research,⁴⁵ FlightGear an open-source project,⁴⁶ and Flight Simulator X (FSX) by Microsoft.⁴⁷ Some flight simulators such as X-Plane, FSX, and FlightGear are interfaceable with network and Internet-based flight simulation networks such as the Virtual Air Traffic Simulation Network.^{48,49}

The authors in Rodriguez-Fernandez et al.⁵⁰ presented a low-cost and easily distributable simulator focused on simulating missions carried out by multiple UAVs while allowing for extraction at the same time. The simulator is called Drone Watch and Rescue (DWR) and focuses on the simulation of missions involving multiple UAVs. DWR was developed using modern web development technologies from the field of video games and designed with a 2-level architecture (server–client). The limitations of using modern web technologies for UAV simulations are the high system requirements needed, in which current JavaScript engines have notorious performance troubles when running compute-intensive jobs.

A Real-time Multi-UAV Simulator (RMUS) was presented in the study by Goktogan et al.⁵¹ with implementation to test and validate mechanisms for real demonstration of multiple UAVs data collection and control. In addition to the included mechanisms (off-line complex simulations, Hardware-in-the-Loop (HIL) tests, validation tests, and online mission control system), the article presented a

communication framework called CommLibX that allows simulation modules to communicate over virtual channels and can be easily ported onto real hardware.

AirSim^{52,53} is a relatively new open-source cross-platform (Linux and Windows) simulator built on top of “Unreal Engine,” which offers physically and visually realistic simulations. It includes a physics engine that can operate at high frequency for real-time HIL simulations with support for popular protocols. It also enables the simulation of a host of physical phenomena, such as gravity, magnetism, atmospheric conditions, and provides sensor models that attempt to mimic real-life. AirSim was proposed to enable designers and developers of robotic systems to generate training data to be used by machine learning algorithms. It is designed to be extensible to accommodate different types of vehicles, hardware platforms, and software protocols. The core components of AirSim include environment model, vehicle model, physics engine, sensor models, rendering interface, public Application Programming Interface (API) layer, and an interface layer for vehicle firmware. Several things can be improved in AirSim, as it currently does not:

- Simulate richer collision response nor advanced ground interaction models.
- Simulate various oddities in camera sensors except those directly available in “Unreal engine.”
- Have advanced noise models and lens models.
- Simulate the degradation of GPS signals.
- Simulate advanced wind effects and thermal simulations for fixed-wing vehicles.

Gazebo⁵⁴ is one of the most popular simulation platforms. It is distinguished by including a physics engine, a host of sensor models, and the ability to create 3-D virtual worlds. Moreover, it can be used to build different types of robots such as manipulator’s arms. However, it has limitations in terms of photorealism and a legacy physics engine.

RotorS⁵⁵ is a modular simulation framework intended for the design of micro aerial vehicles (MAVs). It enables developing and simulating control and state estimation algorithms for MAVs. RotorS is based on Gazebo plugins and the Gazebo physics engine. A key advantage is that it created a modular way of assembling MAVs.

A multi-agent simulation system called Ether was designed and developed in the study by Lundell et al.⁵⁵ to facilitate the evaluation of mission planning models for teams of UAVs. Ether has been built as an agent-based system, where the drones and the environment are separately developed and interlaced at runtime through a simple pluggable component called Emods. While using Emods provides flexibility, it also introduces overhead for users who are trying to build their Emods to test their algorithms.

Garcia and Barnes presented a framework that would enable researchers to study the distributed control algorithms in a multi-UAV scenario using models of realistic

unmanned and manned aerial vehicles in real-world modeled environments.⁵⁶

The authors in Meyer et al.⁵⁷ presented the Hector system which is integrated with Robot Operating System (ROS)⁵⁸ and the Gazebo simulator. This comprehensive approach allows simultaneous simulation of diverse aspects such as flight dynamics, onboard sensors, external imaging sensors, and complex environments. It focuses on quadrotor UAVs and has detailed dynamical models. It also has the same limitations as Gazebo.

The authors in Ganoni and Mukundan⁵⁹ proposed a simulation architecture that uses the Unreal Engine 4 for generating both optical and depth sensor outputs from any position and orientation within the environment. As well as providing several key domain-specific simulation capabilities. For example, it allows the simulation of multiple robots and drones in highly realistic models. However, it is focused on implementing and testing computer vision algorithms used by UAVs. Users can test and validate computer vision algorithms involving different drone configurations under many types of environmental effects such as wind gusts.

Other research presented the design of a HIL simulation framework and its actual implementation on a custom constructed UAV helicopter system.²⁰ Such simulation is important because it enables the verification of the overall control performance and safety of the UAV before conducting test flights. The proposed framework contains four modules: onboard hardware, flight control, ground station, and software. All integrated into a HIL simulation environment. The UAVs in the presented research are utilized for implementing newly developed automatic control techniques and missions such as ground target detection and tracking and multi-UAV cooperation. Simulation results were compared to those obtained from actual test flights and showed that the simulator is capable of efficiently predicting the real flight situations.

Also, many researchers used MATLAB tools^{60–64} to simulate their UAVs and scenarios, with most of the work being very limited to certain types or aspects of UAVs. However, many of these simulators such as X-Plane, FlightGear, and FSX take the aircraft input commands through regular computer input devices such as keyboards, joysticks, or Software APIs. Some APIs such as MathWorks FlightGear to MATLAB Simulink interface⁶⁵ enable researchers to develop aircraft control algorithms and ATC protocols and interface the developed algorithms and protocols in the simulation environment.

The Virtual Air Traffic Simulation Network (VATSIM),^{48,49} founded in July 2001, is a completely free online platform, which allows virtual pilots, wherever they are in the world, to connect their flight simulators into a single shared virtual world. VATSIM world helps to build and maintain communities across the globe, providing resources, training and a place to share the experience of VATSIM. It provides users with virtual skies and the ability to fly all kinds of aircraft all over the world. Users can be

pilots or air traffic controllers, who just need to install a software application that connects the Flight Simulator with the VATSIM servers. Clients are available for FSX, FS9, P3D, and X-Plane. These applications also enable users to communicate with ATC and other aircraft as well.

Lockheed Martin Prepar3D⁶⁶ is a commercial visual simulation platform that allows users to create training scenarios across aviation, maritime, and ground domains. It is used by pilots, academic, commercial organizations, and military. Prepar3D can be used for different learning scenarios including vehicle procedures training, cockpit familiarization, flight planning, air traffic controller training, and emergency response preparation. Prepar3D main features include whole earth training environment, customizable atmosphere, easy configuration of hardware, ATC, flight planner, and expandable vehicle library.

Table 1 presents a summary of the main relevant simulators with their highlights and limitations. The UTSim proposed in this article is designed to be a general UAV simulator that can be used to test different UAV aspects such as path planning, communication protocols, object detection and other high-level controls and algorithms related to the mission and environment rather than specific vehicle-related controls. UTSim allows developers and researchers to implement their motion, navigation, and path planning scenarios easily and without any limitations. At the same time, its infrastructure—Unity—is well maintained and developed continuously for more features by the gaming community.

Proposed simulator design

The simulator proposed in this article is designed to enable researchers and allow them to easily create automated and flexible UAV-related testing environments. It permits the creation of different UAV simulation scenarios and capable of testing algorithms like sense and avoid (S&A) and collision avoidance algorithms. The proposed simulator gives the user—in each simulation session—full control of the number of UAVs, the type of each UAV, the initial and target locations, and much more. It provides users with the ability to specify the environment attributes such as dimensions of the environment and passive objects number, dimensions, and location.

UAVs can be controlled to move from starting points to target positions with predefined altitudes and speeds. The change of altitude, speed, rotation, or even target position is also allowed after UAV takeoff and start of a mission. The simulator platform, customization, configuration, and architecture are discussed in detail in the following subsections as well as in Appendix 1.

The simulator platform: Unity

The proposed simulator is developed using the Unity game engine.²¹ Unity is a platform independent game engine developed by Unity Technologies.⁶⁷ It is used to develop

Table 1. Summary of main relevant simulators and tools.

Simulator	Features	Limitations
DWR ⁵⁰	<ul style="list-style-type: none"> • Low-cost • Easily distributable simulator (built using web technologies) • 2-Level architecture (server–client) • Simulating missions carried out by multiple UAVs, while allowing for data extraction 	<ul style="list-style-type: none"> • High system requirements needed
RMUS ⁵¹	<ul style="list-style-type: none"> • Supports multiple UAVs • Data collection and control • Support the direct communication links between UAVs • Functions as both a testing and validation mechanism 	<ul style="list-style-type: none"> • Separate framework for communication simulation; CommLibX • Constrained environment
AirSim ⁵²	<ul style="list-style-type: none"> • Proposed to collect data to be used by machine learning algorithms • Cross-platform support (Linux and Windows) • Open-source • Enables simulation of a host of physical phenomena • Enables the definition of different types of vehicles 	<ul style="list-style-type: none"> • Underlying platform “Unreal Engine” is not well supported • Does not simulate collision response nor advanced ground interaction models • Does not simulate various oddities in camera sensors except those directly available in Unreal engine • Does not have advanced noise and lens models, which are needed to simulate the degradation of GPS signals • Does not simulate advanced wind effects and thermal simulations for fixed-wing vehicles
Gazebo ⁵⁴	<ul style="list-style-type: none"> • Includes a physics engine, a host of sensor models and the ability to create 3-D virtual worlds • It can be used to build different types of robots such as manipulator’s arms 	<ul style="list-style-type: none"> • Limited legacy physics engine • It has limitations in terms of photorealism
RotorS ³⁵	<ul style="list-style-type: none"> • Enables developing and simulating control and state estimation algorithms for MAVs • It is based on Gazebo plugins and the Gazebo physics engine 	<ul style="list-style-type: none"> • Intended for the design of MAVs
Ether ⁵⁵	<ul style="list-style-type: none"> • Multi-agent simulation • Evaluation of mission planning models for teams of UAVs 	<ul style="list-style-type: none"> • The overhead for users to build their own Emods of their tested algorithms
VATSIM ^{48,49}	<ul style="list-style-type: none"> • Free online platform • Provides the user with virtual skies, flying all kinds of aircraft all over the world • Provides resources, training, and software 	<ul style="list-style-type: none"> • More for pilots training and flying skills improvements not UAVs testing
Lockheed Martin Prepar3D ⁶⁶	<ul style="list-style-type: none"> • Allows users to create training scenarios across aviation, maritime and ground domains • Used by pilots, academic, commercial organizations and military • Customizable atmosphere • Easy configuration of hardware across laptops, desktops, and multi-monitor environments • Multiple view and multiplayer 	<ul style="list-style-type: none"> • A commercial software that needs to be purchased

UAV: unmanned aerial vehicle; VATSIM: Virtual Air Traffic Simulation Network; DWR: Drone Watch and Rescue; RMUS: Real-time Multi-UAV Simulator; MAV: micro aerial vehicles.

simulators and video games for computers, game consoles, and mobile devices. It supports 2-D and 3-D graphics, as shown in Figure 1. In addition, the game engine provides scripting through C#, JavaScript, Unity Script, or BOO coding languages.

The simulator is implemented on top of the Unity game engine because it is easy to generate simulation environments in this platform. It provides a simple cooperative environment that facilitates controlling the attributes of any desired environment component. Thus, allowing to easily create simple

rural areas as well as sophisticated city environments. The dimensions of the simulation area, number of UAVs, passive objects density, dimensions and distribution through the area can all be set and controlled easily through Unity. In Unity, it is also easy to load run files and retrieve simulation sessions parameters and data for further analysis and study. This is enabled through C# scripts. The performance of the simulator can also be monitored visually through Unity’s main screen. In addition, UAVs movement, passive objects and ground stations are all visual on Unity main screen.

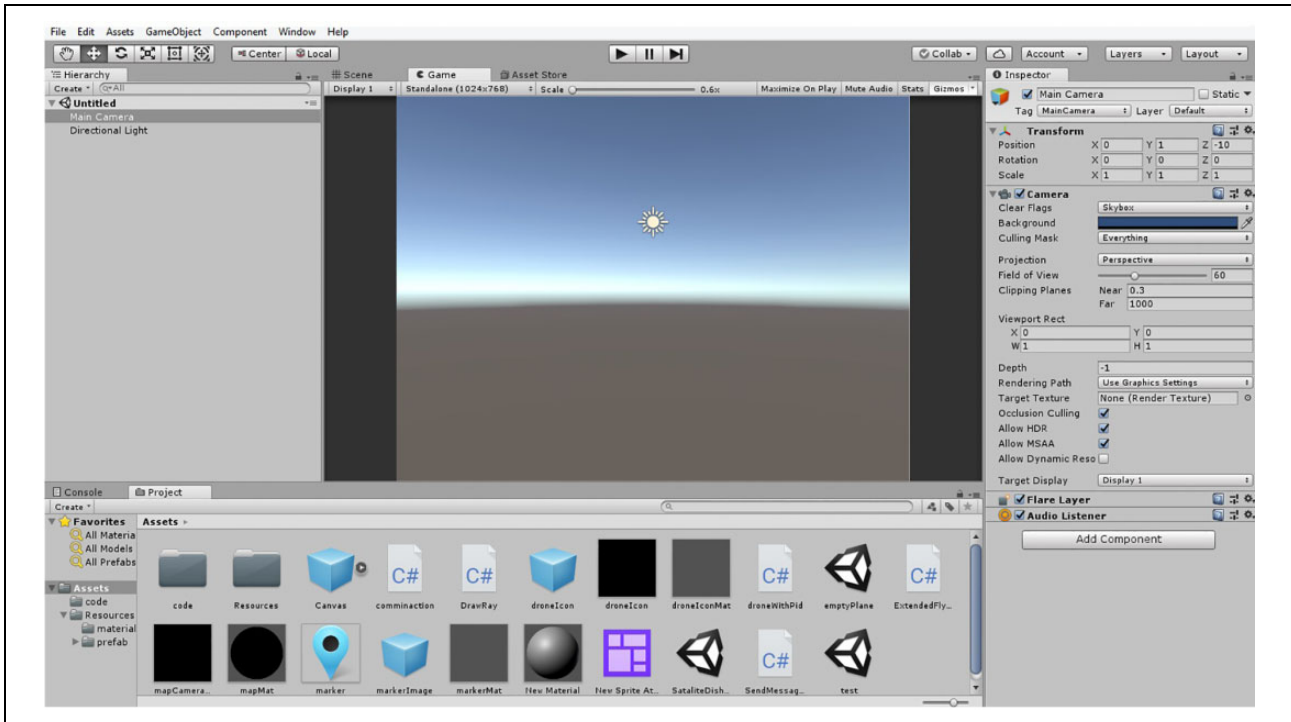


Figure 1. Unity main window.

Figure 1 shows the default layout of the Unity main window. In the middle lies the scene viewer window, where the items included in the scenario can be visualized by the programmer. Programmers can edit the locations of objects in the scene viewer or programmatically. The hierarchy window on the left contains the names of all the objects in the scene. When an object is selected in the hierarchy window, its properties appear in the inspector window to the right. The properties of the object can be viewed and edited either via the GUI or via a script. Game objects needed for any game or the UTSim, in this case, can be added manually in the main screen or using scripts. Game objects are the core items making up any Unity project. They are 2-D or 3-D geometric forms like a sphere or a square. These objects possess many components, but the most important component of any game objects is the transform component that controls the scale, 3-D position and rotation of the object as will be discussed in detail later.

Unity platform customization and configuration

For any object to be part of the Unity environment, it has to be of type Game-Object. Game-Objects can have many components, some for rendering and others for physics and movement control. Scripts are also considered components of Game-Objects, and that is where most of the customization happens as other components of the project are initialized and controlled in scripts. Game-Objects and their components were utilized in UTSim to implement UAVs and give them proper physical properties.

The main components that were used are the following:

1. **Transform:** The transform component of any Game-Object keeps track of its position. This is used when implementing basic UAV movements, path planning, and collision avoidance algorithms.
2. **Rigid-Body:** This component adds Newtonian physics rules to a Game-Object. For example, when implementing UAVs, any UAV can be modeled as a rigid body affected by gravity force. To change the UAV vertical position, a vertical upward force needs to be applied to the UAV. The resulting movement of UAVs during simulation sessions is determined based on the resulting summation of forces affecting the UAV.
3. **Colliders:** Colliders allow Game-Objects to collide, and not act as a transparent object. This component is of a major benefit for collision detection implementation.
4. **Mesh render:** This component handles the rendering of Game-Objects. Without Mesh rendering a Game-Object will be invisible during simulation sessions.
5. **Scripts:** Scripts in the Game-Objects must inherit what is called a mono-behavior class. A typical Unity Script has the following functions:
 - **Start:** This function is called when the Game-Object is initialized.
 - **Update:** This is called once in every time-frame. This function does not get called at a fixed rate, it depends on the GPU and mostly gets associated with the rendering and animation.

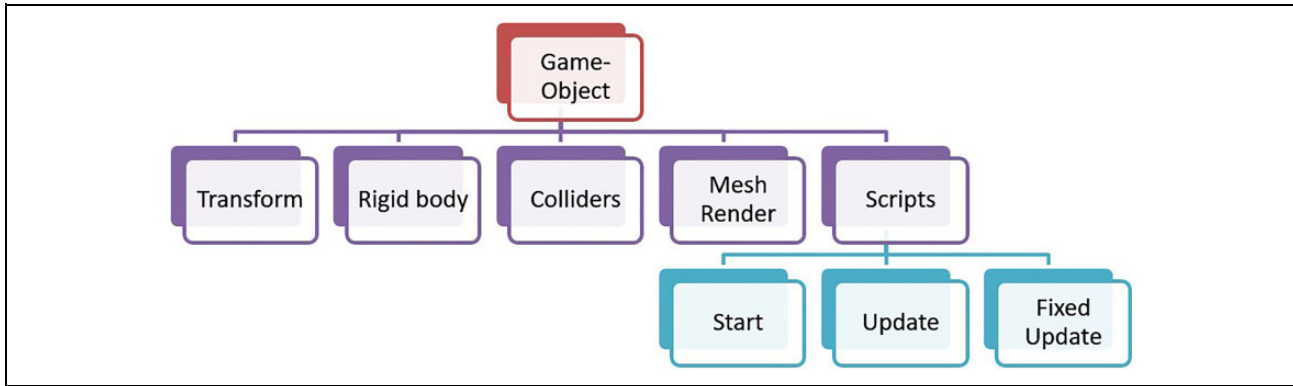


Figure 2. Unity Game-Objects components.

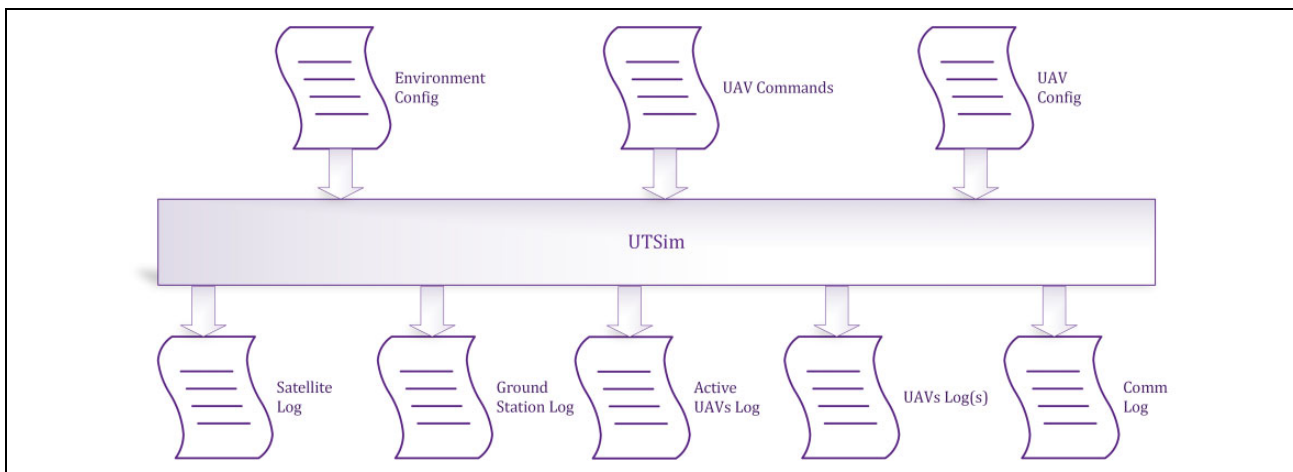


Figure 3. UTSim input configuration and output files.

- **Fixed-Update:** The Fixed-Update function is called at a fixed rate, with a default of 0.02 seconds; this is where the physics and control algorithms are implemented.

The game-object components are illustrated in Figure 2. The status of the game objects and the values of their attributes can be saved to XML files, for logging purposes. This enables later study and analysis of the efficiency of certain algorithms under test.

The simulator architecture

When UTSim starts, it reads multiple configuration files, shown in Figure 3. These XML files are responsible for configuring the environment, the UAVs, and the simulation scenario as follows:

- **UAV Config:** This XML file contains the configuration of the UAV types that are going to be used.
- **Environment Config:** The Environment Config XML file contains all necessary information to initialize the environment and important information to

instantiate and control UAVs such as their type, current location, and target location.

- **UAV Tasks:** This file describes the details of the simulation scenario in the form of a set of predefined actions that the UAVs are required to perform. Actions are a set of orders UTSim is capable of decoding and performing. The main benefit of using actions is to be able to build a complicated simulation scenario and flight plans involving multiple UAVs out of smaller detailed commands. These actions are decoded in C# into instructions to be performed.

Once the simulator starts, the GUI starts showing the desired commands in action. However, for performing detailed analysis, the simulated data need to be logged. UTSim provided several types of logs as follows:

- **Communication log:** This log file keeps track of all messages communicated between the different entities in the simulator.
- **UAVs log:** There is one UAV log generated per UAV. The log file documents actions encountered

from the perspective of that UAV as well as messages the respective UAV sends or receives.

- Active UAVs log: This log file records the number of active UAVs during each time step. Collided UAVs are removed by Unity and are not counted as part of the active UAVs.
- Satellite log: The satellite log file documents messages sent/received by the satellite station.
- Ground station log: The ground station log file documents messages sent/received by each ground station.

Finally, C# scripts are used to specify how UAVs are controlled and driven towards targets based on the actions UTSim supports and algorithms and protocols being tested or performed by UTSim.

Figure 4 shows the flow of a UTSim simulation session. First, the XML configuration file is read. Objects such as UAVs, ground stations, and passive objects are instantiated based on the XML file data. The XML file contains the necessary information to initialize these objects; data such as UAV types, current position, rotation, and speed are set, and passive objects' dimensions and locations are initialized as well. Generally, all objects included in any simulation session are instantiated and their attributes are set right after reading the XML file. Then commands are assigned to each UAV, these commands can be viewed as a flight plan for UAVs, just been created and initialized. Execution then begins. Unity enables a simple time management functionality that can be utilized in parallel execution of commands assigned to each UAV, during execution control algorithms are applied based on the scenario being run. Data such as the number of collisions, the number of messages exchanged can be logged during execution for further analysis and evaluation.

UTSim supported capabilities

In this section, the proposed UTSim framework and its capabilities are discussed. These supported capabilities relate to several areas like the environment, the UAVs, the communication model and flight planning algorithms. The requirements are set to not restrict the simulator with a certain type of UAV or environment. The capabilities also include reporting and data logging to ease extracting simulated flight data parameters for analysis. UTSim also supports existing cooperative S&A satellite and ground-based technologies such as Traffic Alert and Collision Avoidance System (TCAS) and Automatic Dependent Surveillance-Broadcast (ADS-B) already used in manned ATC. Moreover, it shall support noncooperative objects (passive objects) detected via various sensing methods.

General capabilities

The following is a list of general capabilities of UTSim:

1. Custom environment: UTSim enables researchers to test their algorithms and ideas in various environments. This could be a city landscape or a rural area. Professionally built environments could be acquired from the Unity market store. On the other hand, researchers have the option to build their own custom environments from scratch. The simulator provides options to define the simulation environment's area, what objects to be added along with their location. The physical properties of each object, and so on. This flexibility allows for creating a wide range of custom environments.
2. Custom scenarios: UTSim allows the creation of unlimited testing scenarios. A scenario is defined using a custom configuration file, which can be used to configure the number of UAVs in each simulation session, initialize certain parameters for each UAV, define the path plan for each UAV via a set of commands that specify what the UAV is tasked to do during the course of the scenario. Details of the commands will be discussed in the following sections. These scenario configuration files are XML-formatted files that are loaded and decoded at the beginning of each simulation session.
3. Custom algorithms: The simulator is designed to allow testing of collision S&A, and path planning algorithms and protocols. This is supported by having the classes of UTSim written and organized in a modular way to ease editing and adding custom code.
4. Flexible navigation: The simulator allows specifying a starting point and a destination target for each UAV. It also supports maneuverability in 3-D and allows UAVs to be able to move at different speeds.
5. Ability to save and retrieve simulation sessions data: UTSim is developed for educational and research purposes. Hence, a researcher or developer is able to repeatedly test the same scenario with different algorithms. The simulator is designed to be able to retrieve data during simulation sessions and save it for later analysis and evaluation of tested algorithms.

UTSim environment elements

To accurately simulate UAV movements, the environment where these UAVs are flying is an essential part of the simulation. The environment needs to be as real as possible. Taking into account both static objects like buildings and trees, as well as moving objects like other flying objects like UAVs and possible birds. The following is a list of the supported environment elements:

- UAVs: Any number of UAVs can be included, they could be of various types.
- Passive objects: These are objects that are not able to communicate with UAVs but exist in the

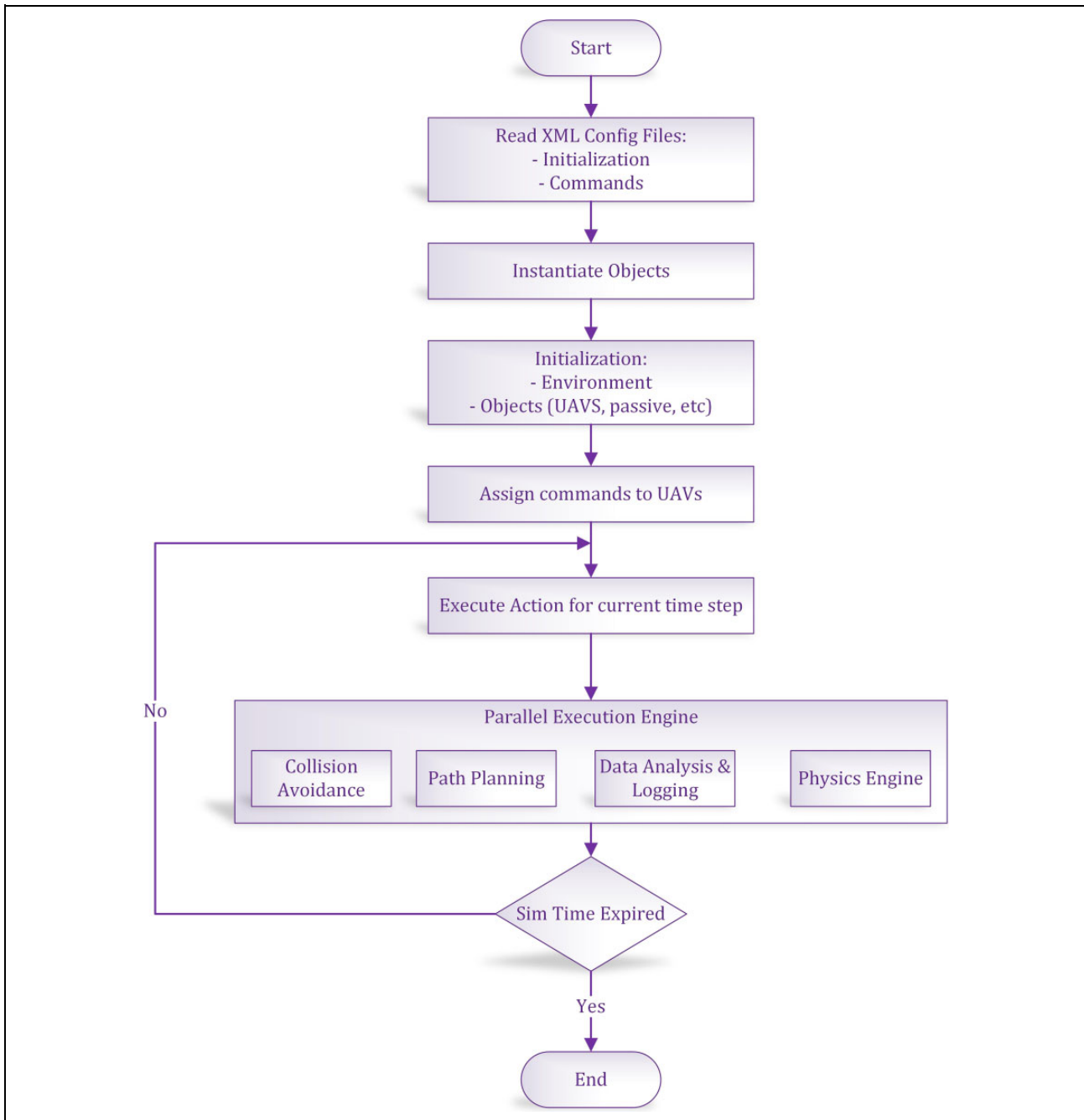


Figure 4. Simulator flow diagram.

surrounding environment. Passive objects could be static, like buildings and towers, and they may be moving like birds. These objects are crucial in order to allow for creating scenes with custom built landscapes based on the scenario being simulated.

- Stations: These are a subset of passive objects where all UAVs can start their missions from and end it at.
- Space communication medium (SCM): This object serves as the medium for exchanging messages between different objects in the environment.

- Satellite: This type of object is used to emulate Satellites and can be used to broadcast information to UAVs.
- Control ground station (CGS): This object simulates the use of certain ground stations as control hubs for UAV traffic control. These CGSs support two-way communication with UAVs.
- Weather conditions: Weather conditions, such as wind and rain, can be added to the UTSim environment. Unity natively takes care of the dynamics resulting from such weather conditions.

The Simulation environment can include any number of the previous objects, except the SCM where there is only one. Details on customizing UTSim are included in Appendix 1.

UAV capabilities

The simulator handles various types of UAVs flying simultaneously at different routes. Each of these UAVs has different physical, electrical properties. The following is a list of the UAV capabilities in UTSim:

1. UAV type configuration: UTSim is capable of simulating many types of UAVs. UTSim has the capability to define a UAV type, where each type shares certain properties like:
 - Aircraft type
 - Dimensions
 - UAV weight
 - Sensing range
 - Max speed
 - Max elevation

Once a UAV type is defined and created these parameters take their default values. Users can create multiple instances of a specific UAV type.

2. UAV Specific Configuration: In addition to the UAV type parameters, there are some parameters that are custom per UAV, these can be configured using the configurable run file before the simulation session starts and subject to change during the simulation. The following is a sample of such properties:
 - Speed
 - Current location
 - Next location
 - Target location
 - UAV ID, a unique ID associated with a UAV during each simulation session.
 - Current battery charge
 - Command list: The set of actions to be performed during a simulation session for each UAV.
 - Sensing range: This attribute defines the capability of each UAV to detect other objects around it. UAVs should be able to detect objects within this defined distance. This attribute depends on the type of sensors on the simulated UAV. It is assumed that working sensors are employed that satisfy such range.
 - Objects within sensing range: Every UAV is able to keep track of all objects within its sensing range. This attribute holds a list of all objects detected by the UAV and removes objects from the list when they leave the sensing range.

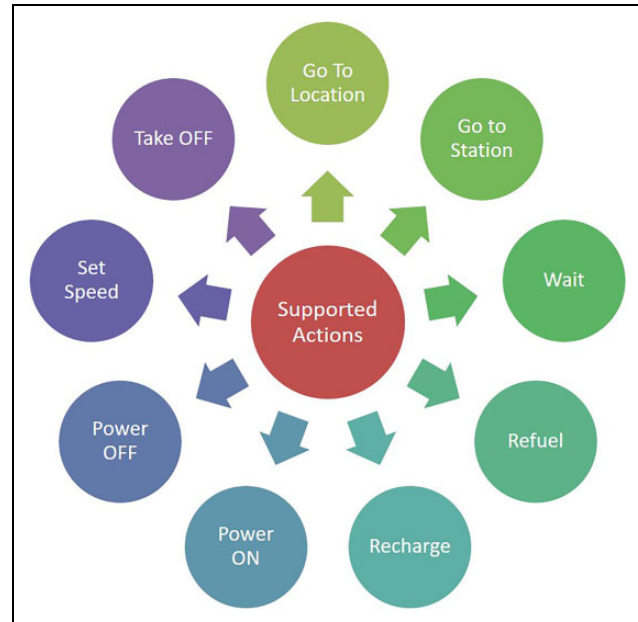


Figure 5. UAV actions supported by UTSim. UAV: unmanned aerial vehicle.

3. UAV supported actions: Actions, shown in Figure 5, describe the possible basic elements a UAV is capable of doing during a simulation session in a human-readable language. During the simulation, these actions are decoded and transformed into C# code that Unity can handle. Actions are loaded for each UAV from the configurable run XML file. Actions were chosen to ensure flexibility in creating a wide variety of simulation scenarios and in a way that allows their generic usage. The following is a list of actions that UTSim supports:
 - Goto location: This action instructs the UAV to head towards a point in space, defined by the 3-D coordinates (x, y, z) . During any simulation session, this command may be used more than once, so the flight plan is set using this command.
 - Power ON: Every UAV needs to be powered ON before it can start operating, this command is used for this purpose.
 - Power OFF: This command is used to turn OFF the UAV during a simulation session.
 - Goto station: This command is used to order UAVs to head to certain stations, making it easy to plan the path without focusing on the location of the target stations.
 - Wait: This command is used to make UAVs wait in their current location for a certain amount of time.
 - Set speed: This command is used to set the speed of a UAV and change it during a simulation session.

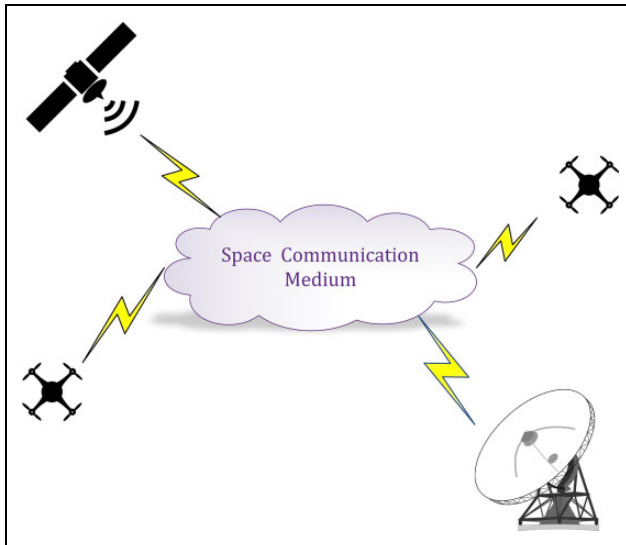


Figure 6. The UTSim communication model.

- Land: This command is used to make a UAV land at its current location.
- Take OFF: When the UAV is on the ground or on top of a building and has to start moving, this command is used to order a UAV to take off to a certain altitude.
- Recharge: This command is used to recharge the batteries of electrically powered UAVs.
- Refuel: This command is used to refuel a UAV assuming that some types of UAVs are fuel powered.

Messaging protocol capabilities

UTSim communication model. A basic assumption for the communication of messages is that each message is assumed to be broadcasted to a radius of n meters, where n is a configurable parameter per UAV type. And that each UAV needs to process messages broadcasted within a certain predefined radius (R_{sensing}). This parameter (R_{sensing}) should be configurable per UAV type and speed, as faster speeds require processing a larger radius.

In order to implement the aforementioned requirement, the exchange of messages between different objects in the environment happens through a new object called ‘Space Communication Medium’ (SCM), shown in Figure 6. This medium serves as an intermediate broker in serving the messages. It has access to all messages being sent, and it is in charge of delivering the messages to the intended recipient. The role of the SCM is important in modeling two factors:

- Message fading: In the physical world, as objects become far from each other messages might fade to the extent the recipient might not be able to detect the message. Thus, the SCM checks the distance

between every two objects before deciding to deliver or drop a message. A more complicated fading model can be developed by incorporating weather conditions, such as humidity and wind.

- Message congestion: whenever there is a large number of objects in the same proximity, it becomes likely that there will be congestion and messages could interfere with each other. Thus the simulator could drop messages based on predefined parameters to simulate this effect.

It is also important to mention that UAVs broadcast periodic messages containing their ID, current position, speed, and other route-related information. Any UAV can receive messages from other UAVs in its sensing range, this feature is valuable in implementing collision avoidance scenarios.

Supported messages. The simulator supports the exchange of various types of messages between UAVs. The following are the supported types of messages:

- A. Periodic messages: Periodic messages are required to be broadcasted by all certified UAVs. With the period being a system wide configurable parameter. The format is expandable to allow for adding any future fields yet being backward compatible with older versions of the protocol. The proposed version of the broadcast messages include the following fields:
 - Unique UAV ID
 - Current UAV coordinates
 - 3-D Directional vector
 - Current UAV speed
 - Priority level—assigned during certification and licensing stage. The level is set based on the cargo type the UAV is expected to be loading (Urgent Medical Supplies vs. Pizza)
- B. Status change messages: Status change messages are meant to broadcast important events triggered by a UAV. They are broadcasted the same way as broadcast messages to every object around. The events that trigger Status Change Messages are:
 - Detecting another UAV in range
 - Changing altitude
 - Changing direction
 - Changing speed
- C. Request messages: These messages have not been implemented yet in UTSim. However, request messages are meant to be communicated between UAVs and control ground station or even between UAVs and are triggered when a UAV is asked to perform a

certain action. A sample of such actions may include:

- Change altitude
- Change speed
- Change direction

Request messages need to be acknowledged by the target UAV that they have been received and whether they will be able to comply or not.

D. Emergency messages: These are messages broadcasted when a UAV is in an emergency situation. Emergency messages get prioritized in handling by other UAVs and by control stations as well. Initial Emergency cases include:

- Critical battery levels.
- Flight equipment malfunction.

These messages are to be supported by UTSim.

Data logging and reporting capabilities

The simulator has extensive data logging capabilities. The logs provided by the simulator can be used for further analysis and evaluation of used algorithms. Details of output logs have been covered in ‘The simulator architecture’ section.

Simulation scenarios

To demonstrate the capabilities of the simulator, three simulation scenarios are presented. The three scenarios are meant to demonstrate some important features of the proposed simulator, but not all of its capabilities.

First scenario: Basic operation

In the first scenario, the basic functionality and operation of the UTSim are demonstrated. The scenario introduces the different types of active and passive objects, as well as the sensing range for UAVs. As illustrated in Figure 7, three UAVs (1, 2, and 3) are commanded to fly to three different target points. The scene also includes two passive objects, a tall building, and a house. The scenario is set up such that there will be no collision. While each UAV records events individually from its own perspective, focus will be on the journey of UAV 1. Figure 8 shows the timeline of the messages logged by UAV 1. Initially, at the time ($t = 0.5$), an object is detected to be within the sensing range of UAV 1, which is the tall building. However, at ($t = 4.5$) and object out of range message is logged to denote that the UAV has lost sight of the tall building. It can also be seen that there are messages, denoted in blue, sent by UAV 1 at ($t = 5, 10, 15, 20$). These are the periodic messages a UAV periodically broadcasts based on a predefined time interval. At ($t = 7.3$), UAV 1 detected an object (which is UAV 2) and almost immediately received a message

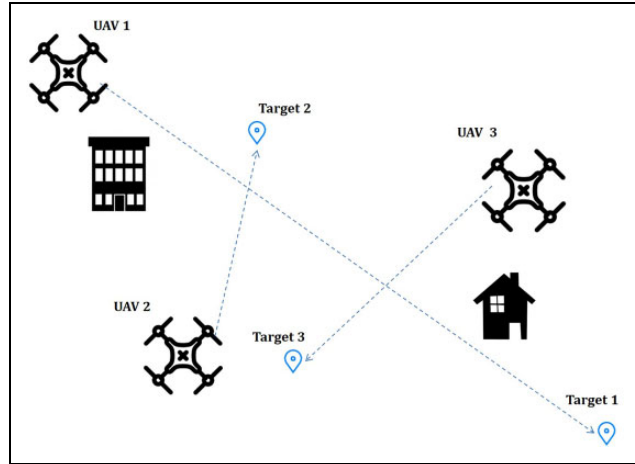


Figure 7. First Scenario: Three UAVs with multiple passive objects. UAV: unmanned aerial vehicle.

from UAV 2, indicating where it is heading. Note that since for the purpose of this simulation all UAVs created are of the same type, and consequently have the same sensing range. Thus each of the UAVs detected each other at the same time. The same repeats at ($t = 9.2$) when UAV 1 detects UAV 3. At ($t = 12$), UAV 2 goes out of range of UAV 1, thus the object out of range event is logged. Afterward, UAV 1 detects the house in the lower right corner of Figure 8 at ($t = 13.48$). Finally, out of range messages are logged for UAV 3 and for the house at ($t = 14.5$) and ($t = 17.3$), respectively.

It is clear that the existence of these logs allows for detailed checking of what is happening during flight simulation and figure out while studying collisions what exactly went wrong.

Second scenario: Periodic messages

In this scenario, the concept of periodic update messages is demonstrated. The scenario is focused on showing the volume of periodic messages as UAVs take off and land. In the scenario, 10 UAVs are launched from a single point, but at different times. They take off in an outward direction, as shown in Figure 9. They all travel at the same speed. Because each of the 10 UAVs flies into different directions, they will never crash. Each of these UAVs has a predefined Target. The aim of this scenario is to inspect the communication log which records messages seen by the medium from all UAVs. The UAVs in this scenario have been configured to broadcast periodic messages every 10 s.

Figure 10 shows the number of update messages between simulation time ($t = 0$) and ($t = 1100$). The figure aggregates the number of periodic messages broadcasted every 10 s. It can be seen that the number of periodic messages increases steadily between ($t = 0$ and $t = 130$) as UAVs are taking off. Then during the period where all 10 UAVs are in flight, the number of periodic messages settles at 10. Finally (at $t = 560$), and as UAVs start reaching their destinations the

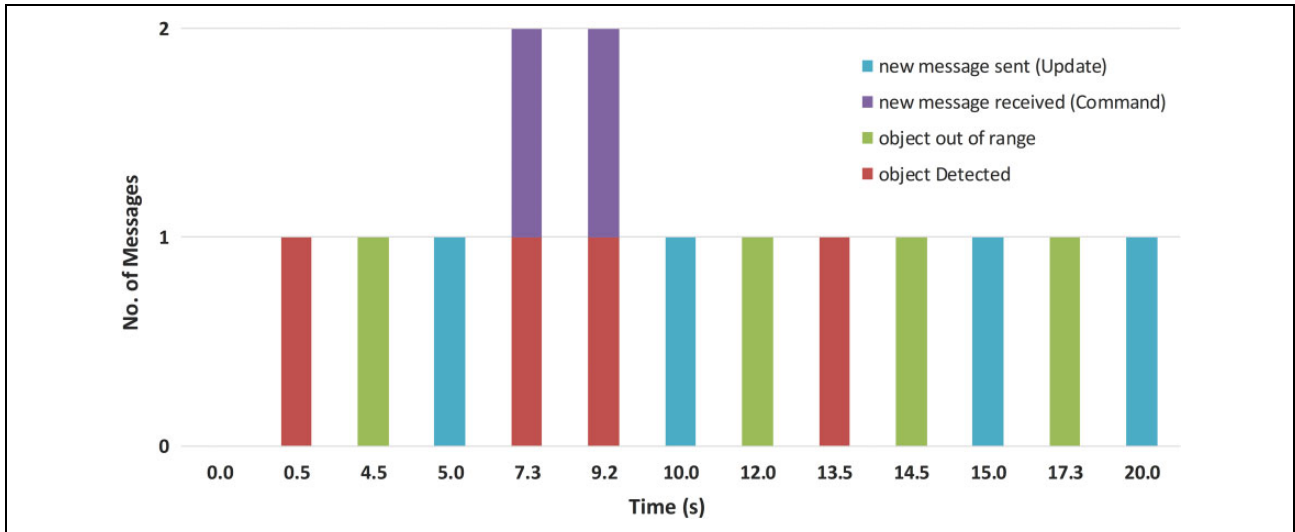


Figure 8. Message log for UAV I in the first Scenario. UAV: unmanned aerial vehicle.

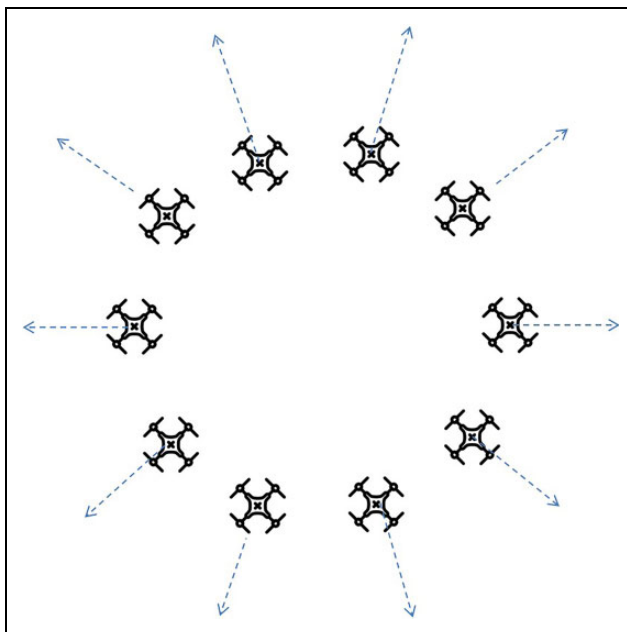


Figure 9. Second Scenario: 10 UAVs. UAV: unmanned aerial vehicle.

number of periodic messages starts declining. The Communication log is an excellent, high-level source to know what has been going on during the simulation.

Third scenario: Scalability and collisions

In this scenario, as shown in Figure 11, the scalability of the UTSim is demonstrated by attempting to simulate as many as 1500 UAVs to fly simultaneously. In addition, the previous two scenarios have been orchestrated carefully to avoid collision between UAVs. However, in this scenario, each UAV is given a target and if two UAVs were too close to each other, they were allowed to crash and collide. This

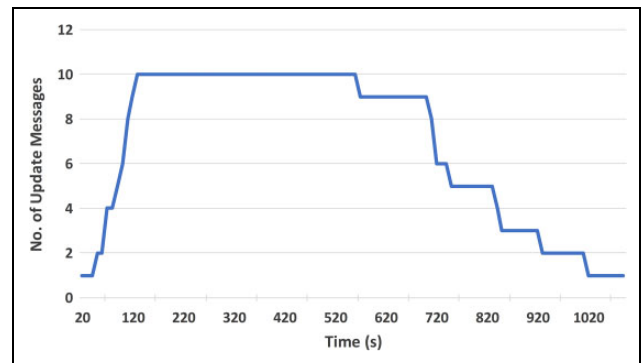


Figure 10. An aggregate of the number of periodic update message during 10 s interval.

scenario was repeated multiple times, each time with a different number of UAVs. Indeed the scenario was run for these number of UAVs: {20, 40, 60, 80, 100, 150, 200, 300, 400, 500, 750, 1000, 1500}. The area where this scenario has been conducted was fixed to be 1 Km × 1 Km. The starting and ending points of the UAVs were completely random.

Figure 12 shows the absolute number of collisions recorded for each run with a different number of UAVs. For example, when there were 750 UAVs, slightly less than 600 UAVs have collided with each other. The figure shows that the number of collisions is linear and directly proportional to the number of UAVs.

Figure 13, illustrates the percentage of the UAVs that have ended up colliding out of the initial starting UAVs. It can be seen that the percentage rises exponentially as the number of UAVs increases, and afterward, this percentage saturates. For example, if there are 200 UAVs, half of this is expected to collide. Of course, it is worth noting that in this scenario all UAVs travel at the same altitude and all have no collision avoidance techniques.

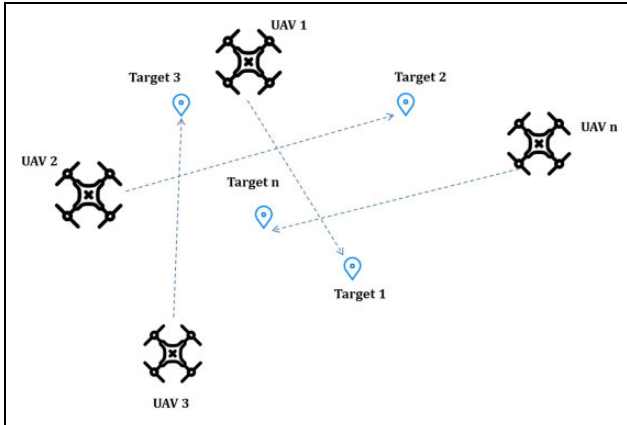


Figure 11. Third Scenario: Colliding UAVs. UAV: unmanned aerial vehicle.

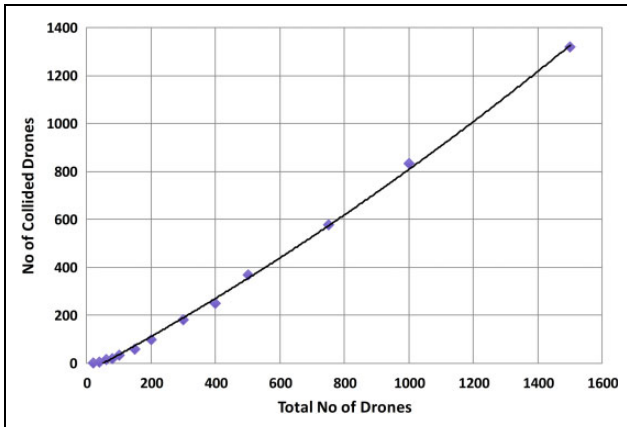


Figure 12. Number of collided UAVs versus number of total starting UAVs. UAV: unmanned aerial vehicle.

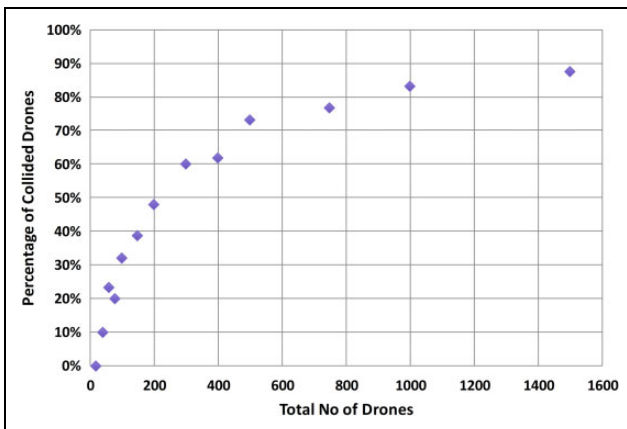


Figure 13. Percentage of collided UAVs versus total number of UAVs. UAV: unmanned aerial vehicle.

Finally, in the same scenario, Figure 14 shows the number of messages exchanged between UAVs as the number of UAVs increases. It can be seen that the number of Update Messages does not increase much as it is tied to

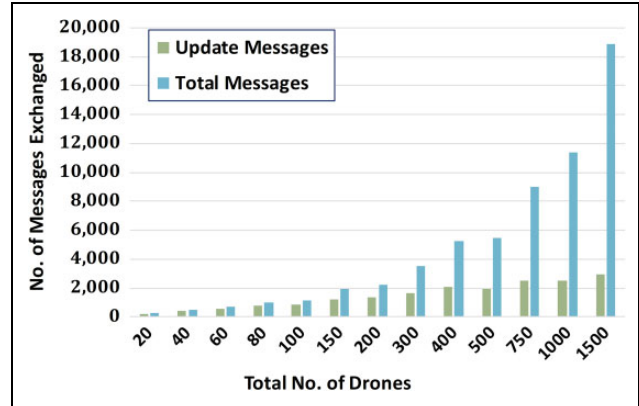


Figure 14. Number of periodic update and total messages versus the number of UAVs. UAV: unmanned aerial vehicle.

the number of UAVs. However, the Total Number of messages drastically increases as the number of UAVs increases. This is due to the interaction between different UAVs as they come in and out of each others' sensing range. The total number of messages is expected to be in the order of n^2 , while the number of update messages is in the order of n , where n is the number of UAVs.

Conclusion

As unmanned aerial systems are continuously increasing and becoming part of more applications, the embedding of UAVs in the national aerospace has become a must. Researchers have started working on the integration of UAVs in current air traffic. This kind of research is quite expensive if it is physically performed. The need for simulators that are easy to use and provide researchers with tools an capabilities to evaluate their proposed algorithms and retrieve useful information to assist in this task, the need for such simulators is vital. In this work, a simulator called UTSim is proposed. The simulator enables researchers to simulate air traffic integration scenarios and empowers them with the tools to design, develop and test air traffic integration and navigation algorithms. UTSim enables researchers to specify properties of the environment, the number, and the type of UAVs, in addition to protocols and algorithms to manage air traffic integration issues like communication, path planning, collision avoidance and more. The proposed simulator has been explained in detail and several simulation scenarios have been presented to illustrate the capabilities of the simulator.


Declaration of conflicting interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The author(s) received no financial support for the research, authorship, and/or publication of this article.

ORCID iD

Amjed Al-Mousa  <https://orcid.org/0000-0002-6427-1008>

References

- Ping JTK, Ling AE, Quan TJ, et al. Generic unmanned aerial vehicle (UAV) for civilian application—a feasibility assessment and market survey on civilian application for aerial imaging. In: *2012 IEEE conference on sustainable utilization and development in engineering and technology (STUDENT)*, Kuala Lumpur, Malaysia, October 2012, pp. 289–294. Piscataway, NJ: IEEE.
- Christie KS, Gilbert SL, Brown CL, et al. Unmanned aircraft systems in wildlife research: current and future applications of a transformative technology. *Front Ecol Environ* 2016; 14(5): 241–251.
- Adams SM and Friedland CJ. A survey of unmanned aerial vehicle (UAV) usage for imagery collection in disaster research and management. In: *9th international workshop on remote sensing for disaster response*, vol. 8, Stanford, CA, USA, September 2011.
- Achille C, Adami A, Chiarini S, et al. UAV-based photogrammetry and integrated technologies for architectural applications—methodological strategies for the after-quake survey of vertical structures in Mantua (Italy). *Sensors* 2015; 15(7): 15520–15539.
- Yang H, AbouSleiman R, Sababha B, et al. Implementation of an autonomous surveillance quadrotor system. In: *Proceedings of AIAA unmanned unlimited conference*, Seattle, Washington, 6–9 April 2009, pp. AIAA–2009–2047. Washington: AIAA.
- Kontogiannis SG and Ekaterinaris JA. Design, performance evaluation and optimization of a UAV. *Aerosp Sci Technol* 2013; 29(1): 339–350.
- Goerzen C, Kong Z, and Mettler B. A survey of motion planning algorithms from the perspective of autonomous UAV guidance. *J Intell Robot Syst* 2010; 57(1-4): 65.
- Bonin-Font F, Ortiz A, and Oliver G. Visual navigation for mobile robots: a survey. *J Intell Robot Syst* 2008; 53(3): 263–296.
- Kanellakis C and Nikolakopoulos G. Survey on computer vision for UAVs: current developments and trends. *J Intell Robot Syst* 2017; 87(1): 141–168.
- Abughalieh KM, Sababha BH, and Rawashdeh NA. A video-based object detection and tracking system for weight sensitive UAVs. *Multimed Tools Appl* 2019; 78(7): 9149–9167.
- Rawashdeh NA, Rawashdeh OA, and Sababha BH. Vision-based sensing of UAV attitude and altitude from downward in-flight images. *J Vib Control* 2017; 23(5): 827–841.
- AbouSleiman R, Sababha B, Yang HC, et al. Real-time estimation of UAV attitude from aerial fisheye video. In: *Proceedings of AIAA Infotech@Aerospace conference*. Seattle, Washington, 6–9 April 2009, pp. AIAA–2009–1933. Washington: AIAA.
- Federal Aviation Administration. Summary of small unmanned aircraft rule (Part 107), https://www.faa.gov/uas/media/Part_107_Summary.pdf (accessed 13 June 2019).
- Federal Aviation Administration. *Waivers to certain small UAS operating rules*, https://www.faa.gov/uas/beyond_the_basics (accessed 13 June 2019).
- Richter JL and Aitken MA. United States: FAA issues first national waiver for beyond-visual-line-of-sight drone operations, <https://bit.ly/2ZjtflL> (accessed 13 June 2019).
- Wikipedia. Amazon Prime Air, https://en.wikipedia.org/wiki/Amazon_Prime_Air (accessed 13 June 2019).
- UPS Pressroom. UPS Tests residential delivery via drone launched from atop package car, <https://bit.ly/2xPMunO> (accessed 13 June 2019).
- Evan Ackerman. Wing officially launches Australian drone delivery service, <https://bit.ly/2IDm3sK>. IEEE Spectrum, April 2019 (accessed 13 June 2019).
- Hofstede GJ, De Caluwé L, and Peters V. Why simulation games work-in search of the active substance: a synthesis. *Simulat and Gaming* 2010; 41(6): 824–843.
- Cai G, Chen BM, Lee TH, et al. Design and implementation of a hardware-in-the-loop simulation system for small-scale UAV helicopters. *Mechatronics* 2009; 19(7): 1057–1066.
- Unity Technologies. The world’s leading content-creation engine. 2018, <https://unity3d.com/unity> (accessed 19 July 2018).
- Olaverri-Monreal C, Errea-Moreno J, Daz-Álvarez A, et al. Connection of the SUMO microscopic traffic simulator and the Unity 3D game engine to evaluate V2X communication-based systems. *Sensors* 2018; 18(12): 4399.
- Chan MT, Chan JT, Chan C, et al. An artificial intelligence-based vehicular system simulator. *Int J Soft Sci Comput Intell (IJSSCI)* 2017; 9(1): 55–68.
- Chin CS, Kamsani NB, Zhong X, et al. Unity3D serious game engine for high fidelity virtual reality training of remotely-operated vehicle pilot. In: *2018 10th international conference on modelling, identification and control (ICMIC)*, Guiyang, China, 2 July–4 July 2018, pp. 1–6. NJ, USA: IEEE.
- Juliani A, Berges VP, Vckay E, et al. Unity: a general platform for intelligent agents. 2018. arXiv preprint arXiv:1809.02627.
- Andaluz VH, Chicaiza FA, Gallardo C, et al. Unity3D-MatLab simulator in real time for robotics applications. In: *International conference on augmented reality, virtual reality and computer graphics*, Otranto, Italy, 15 June–18 June 2016, pp. 246–263. Cham: Springer.
- Zhang J, Lyu Y, Wang Y, et al. Development of laparoscopic cholecystectomy simulator based on unity game engine. In: *Proceedings of the 15th ACM SIGGRAPH European conference on visual media production*, London, United Kingdom, 13–14 December 2018. New York: ACM. DOI 10.1145/3278471.3278474.
- Jones B, Rohani SA, Ong N, et al. A virtual-reality training simulator for cochlear implant surgery. *Simulat and Gaming* 2019; 50(2): 243–258.

29. Horton BK, Kalia RK, Moen E, et al. Game-engine-assisted research platform for scientific computing (gears) in virtual reality. *SoftwareX* 2019; 9(2019): 112–116.
30. Li B, Gong GH, and Zhao YP. Physically-based facial modeling and animation with Unity3D game engine. In: *Asian Simulation Conference*, Melaka, Malaysia, 27–29 August 2017, pp. 393–404. Springer Link.
31. Hu W, Si S, and Wang Y. Chemistry experiment simulation based on game engine. In: *2018 IEEE/ACIS 17th international conference on computer and information science (ICIS)*, Singapore, 6–8 June 2018, pp. 776–780. Piscataway, NJ: IEEE.
32. Visser A, Dijkshoorn N, Van der Veen M, et al. Closing the gap between simulation and reality in the sensor and motion models of an autonomous AR.Drone. In: *Proceedings of the international micro air vehicle conference and flight competition (IMAV11)*, vol 108, 't Harde, the Netherlands, 12–15 September. Delft, Netherlands: Delft University of Technology.
33. Dosovitskiy A, Ros G, Codevilla F, et al. CARLA: an open urban driving simulator. 2017. arXiv preprint arXiv:1711.03938.
34. Mueller M, Smith N, and Ghanem B. A benchmark and simulator for UAV tracking. In: *European conference on computer vision*, Amsterdam, The Netherlands, 8–16 October, pp. 445–461. Cham: Springer.
35. Furrer F, Burri M, Achtelik M, et al. RotorS—a modular gazebo MAV simulator framework. In: Koubaa A (ed) *Robot Operating System (ROS)*. Studies in Computational Intelligence, vol 625. Cham: Springer, 2016.
36. Santana E and Hotz G. Learning a driving simulator. 2016. arXiv preprint arXiv:1608.01230.
37. Zeng X, Garg SK, Strazdins P, et al. IOTSim: a simulator for analysing IoT applications. *J Syst Architect* 2017; 72(2017): 93–107.
38. Manhaes M, Marcusso M, Scherer SA, et al. UUV simulator: a Gazebo-based package for underwater intervention and multi-robot simulation. In: *OCEANS 2016 MTS/IEEE Monterey*, CA, USA, 19–23 September 2016, pp. 1–8. Piscataway, NJ: IEEE.
39. Rohmer E, Singh SP, and Freese M. V-REP: a versatile and scalable robot simulation framework. In: *2013 IEEE/RSJ international conference on Intelligent Robots and Systems (IROS)*, Tokyo, Japan, 3–7 November 2013, pp. 1321–1326. Piscataway, NJ: IEEE.
40. Hugues L and Bredeche N. Simbad: an autonomous robot simulation package for education and research. In: *International conference on simulation of adaptive behavior*, Rome, Italy, 25–29 September 2006, pp. 831–842. Springer Link.
41. Rohrmeier M. Interactive simulation using virtual systems: web based robot simulation using VRML. In: *Proceedings of the 32nd conference on Winter simulation*, Orlando, Florida, 10–13 December 2000, pp. 1525–1528. USA, CA: Society for Computer Simulation International.
42. Michel O. Cyberbotics Ltd. webots™: professional mobile robot simulation. *Int J Adv Robot Syst* 2004; 1(1): 5.
43. X-Plane. <https://www.x-plane.com/desktop/how-x-plane-works/> (accessed 20 August 2018).
44. Blade Element Theory. https://en.wikipedia.org/wiki/Blade_element_theory (accessed: 20 August 2018).
45. X-Plane. <https://www.x-plane.com/press-kit/> (accessed 20 August 2018).
46. FlightGear. <http://home.flightgear.org/about/> (accessed 20 August 2018).
47. Microsoft Flight Simulator X. https://en.wikipedia.org/wiki/Microsoft_Flight_Simulator_X (accessed 20 August 2018).
48. VATSIM. <https://www.vatsim.net/about> (accessed 25 August 2018).
49. VATSIM. https://en.wikipedia.org/wiki/Virtual_Air_Traffic_Simulation_Network (accessed 20 August 2018).
50. Rodriguez-Fernandez V, Menéndez HD, and Camacho D. Design and development of a lightweight multi-UAV simulator. In: *2015 IEEE 2nd international conference on Cybernetics (CYBCONF)*, Gdynia, Poland, 24–26 June 2015, pp. 255–260. Piscataway, NJ: IEEE.
51. Goktogan AH, Nettleton E, Ridley M, et al. Real time multi-UAV simulator. In: *Proceedings ICRA'03. IEEE international conference on robotics and automation, 2003*, vol 2, Taiwan, 14–19 September 2003, pp. 2720–2726. Piscataway, NJ: IEEE.
52. Shah S, Dey D, Lovett C, et al. AirSim: high-fidelity visual and physical simulation for autonomous vehicles. In: Hutter M, and Siegwart R (eds) *Field and service robotics*. Springer Proceedings in Advanced Robotics, vol 5. Cham: Springer, 2018.
53. AirSim. <https://github.com/Microsoft/AirSim> (accessed 25 August 2018).
54. Koenig N and Howard A. Design and use paradigms for Gazebo, an open-source multi-robot simulator. In: *Proceedings of 2004 IEEE/RSJ international conference on Intelligent Robots and Systems, 2004 (IROS 2004)*, vol. 3. Sendai, Japan, 28 September–2 October 2004, pp. 2149–2154. Piscataway, NJ: IEEE.
55. Lundell M, Tang J, Hogan T, et al. An agent-based heterogeneous UAV simulator design. In: *Proceedings of the 5th WSEAS international conference on artificial intelligence, knowledge engineering and data bases*, Madrid, Spain, 15–17 February 2006, pp. 453–457. World Scientific and Engineering Academy and Society (WSEAS).
56. Garcia R and Barnes L. Multi-UAV simulator utilizing X-plane. In: *Selected papers from the 2nd International Symposium on UAVs*, Reno, Nevada, USA, 8–10 June 2009. Reno, Nevada: Springer, pp. 393–406.
57. Meyer J, Sendobry A, Kohlbrecher S, et al. Comprehensive simulation of quadrotor UAVs using ROS and Gazebo. In: *International conference on simulation, modeling, and programming for autonomous robots*, Tsukuba, Japan, 5–8 November 2012, pp. 400–411. Springer Link.
58. Quigley M, Conley K, Gerkey B, et al. ROS: an open-source robot operating system. In: *ICRA workshop on open source software*, vol. 3, Kobe, Japan, 17 May 2009, p. 5. Piscataway, NJ.
59. Ganoni O and Mukundan R. A framework for visually realistic multi-robot simulation in natural environment. 2017. arXiv preprint arXiv:170801938.

60. Richards A, Schouwenaars T, How JP, et al. Spacecraft trajectory planning with avoidance constraints using mixed-integer linear programming. *J Guid Control Dynam* 2002; 25(4): 755–764.
61. Mujumdar A and Padhi R. Evolving philosophies on autonomous obstacle/collision avoidance of unmanned aerial vehicles. *J Aerosp Comp Inf Com* 2011; 8(2): 17–41.
62. Fu Y, Zhang Y, and Yu X. An advanced sense and collision avoidance strategy for unmanned aerial vehicles in landing phase. *IEEE Aero El Sys Mag* 2016; 31(9): 40–52.
63. Sahawneh LR, Mackie J, Spencer J, et al. Airborne radar-based collision detection and risk estimation for small unmanned aircraft systems. *J Aerosp Inf Syst* 2015; 12(12): 756–766.
64. Fu Y, Yu X, and Zhang Y. Sense and collision avoidance of unmanned aerial vehicles using Markov decision process and flatness approach. In: *2015 IEEE International Conference on Information and Automation*, Lijiang, China, 8–10 August 2015, pp. 714–719. Piscataway, NJ: IEEE.
65. Aerospace Blockset. <https://www.mathworks.com/help/aeroblks/index.html;jsessionid=7f23b5dc0d517d02eb01d5c663d3> (accessed 20 August 2018).
66. Lockheed Martin Prepar3D. <https://www.prepar3d.com> (accessed 25 August 2018).
67. Wikipedia contributors. Unity (game engine)—Wikipedia, the free encyclopedia, [https://en.wikipedia.org/w/index.php?title=Unity_\(game_engine\)&oldid=850638164](https://en.wikipedia.org/w/index.php?title=Unity_(game_engine)&oldid=850638164) (2018, accessed 19 July 2018).
68. Wang Z, Kim B, Kobayashi H, et al. Agent-Based Modeling and Simulation of Connected and Automated Vehicles Using Game Engine: A Cooperative On-Ramp Merging Study. *CoRR*. arXiv preprint arXiv:1810.09952. 2018 October 23. <https://arxiv.org/abs/1810.09952>

Appendix I

Simulator customization

Introducing new drone types. UTSim allows users to create their own type of drones, with custom properties. Users can edit the properties of the GameObject ‘basicDrone’ in the Unity environment to match the required properties. Alternatively, if users needed to create multiple drone types, they can copy the “basicDrone” GameObject and rename it (e.g. “basicDrone1”) and customize the new object as needed.

Implementing flight scenarios. UTSim gives users the ability to create custom scenarios. Each scenario needs an XML-based configuration file. In the configuration file the objects, whether they are UAVs or static objects (like trees or buildings) are created. The XML configuration file gives the user the ability to instantiate drones in the session and specify the type and name of that drone instant and then initialize the attributes related to that instant. Also, when creating a UAV, a flight plan on how and where to move is associated with that UAV.

The following is part of the configuration file used to run the first test case scenario, described in the ‘Stimulation scenarios’ section and shown in Figure 7. The configuration file creates 5 objects: 3 UAVS and 2 static objects. The code for UAV 1 and static object 1 is shown below.

When creating the UAVs, its position in 3-D space is defined, then its scale and rotation, as shown in the comments in the code. Next, the UAV is assigned a unique name and is associated with a specific type of Drones ‘basicDrone.’ By default, the instance inherits all the default values of the drone type. After that, the command section is responsible for defining the flight plan for drone 1. Through a list of commands, the speed is set and then a specific direction is given to the drone. The list could contain as many commands as needed. The list of possible actions is explained in the ‘UAV supported actions’ section. It can be seen also that the instance can override values inherited from the type. Finally, the two static objects are defined. Their position, scale, and rotation are set. Also, names are assigned and the type of the static objects are defined.

An example of scenario configuration file for UTSim:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Define an Array of list of objects in the simulation -->
<ArrayOfObjectStatexmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <!-- Define a UAV Object UAV 1 -->
  <!-- ===== -->
  <objectStatexsi:type="uavObjectState">

    <!-- Initialize the Position of UAV 1 -->
    <position>
      <x>-5</x> <y>50</y> <z>16.6</z>
    </position>
```

```

<!-- Initialize the Scale of UAV 1 -->
<scale>
  <x>1</x> <y>1</y> <z>1</z>
</scale>
<!-- Initialize the Rotation of UAV 1 -->
<rotation>
  <x>0</x> <y>0</y> <z>0</z>
</rotation>

<!-- Initialize the Name of UAV 1 -->
<name>drone 1</name>

<!-- Define the type of the Drone -->
<prefabName>basicDrone</prefabName>

<!-- A Set of Commands assigned to the UAV 1 -->
<cmdList>
  <string>setSpeed 5</string>
  <string>move x88 y50 z-76.4</string>
</cmdList>

<!-- Define Data for the UAV 1 -->
<uavData>
  <KeyValuePairOfStringString>
    <Key>mass</Key>
    <Value>4</Value>
  </KeyValuePairOfStringString>
</uavData>
</objectState>

<!-- Define a UAV Object UAV 2 & 3 -->
<!--.....-->

<!-- Define a Passive Object 1 -->
<!-- ===== -->
<objectState xsi:type="visibleObjectState">
  <!-- Initialize the Position of Obj 1 -->
  <position>
    <x>0</x> <y>50</y> <z>0</z>
  </position>

  <!-- Initialize the Scale of Obj 1 -->
  <scale>
    <x>4</x> <y>100</y> <z>4</z>
  </scale>
  <!-- Initialize the Rotation of Obj 1 -->
  <rotation>
    <x>0</x> <y>0</y> <z>0</z>
  </rotation>

  <!-- Initialize the Name of Obj 1 -->
  <name>Passive Object 1</name>

  <!-- Define the type of the Obj 1 -->
  <prefabName>Cube</prefabName>
</objectState>

  <!-- Code to configure Passive Obj 2 -->
  <!--.....-->
</ArrayOfObjectState>

```

Implementing custom S&A algorithms. UTSim allows for implementing custom S&A algorithms. The core of such algorithms is related to translation in the position of the UAVs and the velocity by which they move in the 3D space. These can be defined as part of the **Move** action of any UAV. In the UTSim, the function **decode()** is responsible for decoding the actions dictated by the user in the simulation configuration, including the **Move**. Within the scope of the **Move** action, the algorithm has access to information provided by the assumed sensors, which are crucial to the decision-making process. In the current native version of the simulator, the **Move** action code employs a simple PID controller to provide a force responsible for moving the UAV from the current position to the target position. The force decreases gradually as the UAV approaches the destination.

When a custom sense and avoid algorithm is to be implemented, the code should be part of decoding the **Move** action in the **decode()** function. Users can alter the decision to move the UAVs based on the S&A algorithm being implemented.